

Klaus Löffelmann

# **Visual Basic 2005 – Das Entwicklerbuch**

**Microsoft®**  
*Press*

Klaus Löffelmann: Visual Basic 2005 – Das Entwicklerbuch  
Microsoft Press Deutschland, Konrad-Zuse-Str. 1, 85716 Unterschleißheim  
Copyright © 2006 by Microsoft Press Deutschland

Das in diesem Buch enthaltene Programmmaterial ist mit keiner Verpflichtung oder Garantie irgendeiner Art verbunden. Autor, Übersetzer und der Verlag übernehmen folglich keine Verantwortung und werden keine daraus folgende oder sonstige Haftung übernehmen, die auf irgendeine Art aus der Benutzung dieses Programmmaterials oder Teilen davon entsteht. Die in diesem Buch erwähnten Software- und Hardwarebezeichnungen sind in den meisten Fällen auch eingetragene Marken und unterliegen als solche den gesetzlichen Bestimmungen. Der Verlag richtet sich im Wesentlichen nach den Schreibweisen der Hersteller.

Das Werk, einschließlich aller Teile, ist urheberrechtlich geschützt. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtsgesetzes ist ohne Zustimmung des Verlags unzulässig und strafbar. Das gilt insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen und die Einspeicherung und Verarbeitung in elektronischen Systemen.

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1  
08 07 06

ISBN 3-86063-537-9

© Microsoft Press Deutschland  
(ein Unternehmensbereich der Microsoft Deutschland GmbH)  
Konrad-Zuse-Str. 1, D-85716 Unterschleißheim  
Alle Rechte vorbehalten

Satz: Silja Brands, Klaus Löffelmann, ActiveDevelop, Lippstadt (<http://ActiveDevelop.de>)  
Fachlektorat: Ruprecht Dröge, Ratingen (<http://BeConstructed.de>)  
Testing: Jürgen Heckhuis, Lippstadt  
Umschlaggestaltung: Hommer Design GmbH, Haar ([www.HommerDesign.com](http://www.HommerDesign.com))  
Layout und Gesamtherstellung: Kösel, Krugzell ([www.KoeselBuch.de](http://www.KoeselBuch.de))

# Teil E

## Die wichtigsten Datentypen des Frameworks

---

- 439 Primitive Datentypen**
  - 497 Kulturabhängiges Formatieren von Zahlen- und Datumswerten**
  - 533 Enums (Aufzählungen)**
  - 539 Arrays und Auflistungen (Collections)**
  - 595 Arbeiten mit generischen Typen und generischen Auflistungen**
  - 613 Reguläre Ausdrücke (Regular Expressions)**
  - 657 Serialisierung von Objekten**
  - 687 Attribute und Reflection**
- 

Das .NET-Framework kennt nicht weniger als ca. 8.000 verschiedene Klassen. Sie alle im Rahmen eines solchen Buches aufzuarbeiten ist ein Ding der Unmöglichkeit – und auch gar nicht notwendig, denn: Die Online-Hilfe von Visual Studio ist inzwischen von brauchbarer Qualität, und mithilfe von IntelliSense und dynamischer Hilfe ist Gesuchtes schnell gefunden.

Es gibt allerdings einige ganz essenzielle Datentypen, dazugehören beispielsweise alle primitiven Datentypen wie Integer, String, Date, Double, Single oder Decimal oder auch die zahlreichen Auflistungsklassen zur Verwaltung von Objektmengen, die Sie im täglichen Entwicklungseinsatz immer und immer wieder verwenden. Sie aus dem Effeff zu beherrschen ist wichtig, denn das spart beim Entwickeln größerer Projekte wertvolle Zeit. Und vor allen Dingen: Wissen Sie diese wichtigsten Datentypen des Frameworks richtig anzuwenden, stellen alle anderen Klassen auch kein Problem mehr dar.



# 16 Primitive Datentypen

---

- 440 **Einführung**
  - 440 **.NET-Äquivalente primitiver Datentypen**
  - 442 **Numerische Datentypen**
  - 459 **Der Datentyp Char**
  - 460 **Der Datentyp String**
  - 482 **Der Datentyp Boolean**
  - 488 **Der Datentyp Date**
- 

Nun wissen Sie spätestens seit dem Durcharbeiten des letzten Buchteils, was Klassen sind, wie Sie sie entwickeln, wie Sie eigene Datentypen entwerfen können und dass alle Klassen von *Object* abgeleitet sind. Sie haben auch schon viele der .NET-Datentypen kennen gelernt. Allerdings fehlen bislang immer noch genaue Informationen über das Konzept einiger spezieller Datentypen, die in .NET fest integriert sind und die Sie nahezu ständig beim Entwickeln eigener Applikationen verwenden müssen.

Das sind vor allen die schon erwähnten primitiven Datentypen, wie *Integer*, *Double*, *Date*, *String*, etc. Sie sind in den .NET-Sprachen von C# oder Visual Basic fest verankert, was Sie im Übrigen auch daran feststellen können, dass der Editor sie blau markiert, wenn Sie die Schlüsselwörter ausgeschrieben haben.

Aus Framework-Sicht sind die primitiven Datentypen ebenfalls von *Object* abgeleitet, einigen von ihnen kommt jedoch durch das Framework zur Laufzeit eine Sonderbehandlung zu: Es wäre natürlich unsinnig, beispielsweise einen Additionsbefehl für den *Integer*-Typen durch neuen Framework-Code zu programmieren, bietet doch jeder Prozessor eigene Befehle an, um eine 16, 32 oder 64 Bit breite Integer-Addition selbst durchzuführen.

# Einführung

Unter primitive Datentypen fallen alle Typen, die in einer Programmiersprache unter .NET fest im Sprachschatz verankert sind. Das heißt genauer:

- Ein konstanter Wert jedes primitiven Datentyps kann in Schriftform angegeben werden. Die Angabe 123.324D bezeichnet so beispielsweise einen Wert bestimmter Größe vom Typ Decimal.
- Es ist möglich, einen primitiven Datentyp als Konstante zu deklarieren. Wenn ein bestimmter Ausdruck ausschließlich als Konstante definiert wird (also beispielsweise beim Ausdruck 123.32D\*2+100.23D), kann er schon beim Kompilieren ausgewertet werden.
- Viele Operationen und Funktionen bestimmter primitiver Datentypen können vom Framework zur Ausführung direkt an den Prozessor delegiert werden. Dazu gehören die meisten der Operationen der Datentypen Byte, Short, Integer, Long, Single, Double und Boolean.

## .NET-Äquivalente primitiver Datentypen

Trotz ihrer festen Verankerung in der Sprache gibt es für jeden der primitiven Datentypen ein .NET-Äquivalent. Die folgende Tabelle gibt Ihnen Auskunft darüber:

Primitiver Datentyp in VB	.NET-Datentyp Äquivalent
Byte	System.Byte
SByte	System.SByte
Short	System.Int16
UShort	System.UInt16
Integer	System.Int32
UInteger	System.UInt32
Long	System.Int64
ULong	System.UInt64
Single	System.Single
Double	System.Double
Decimal	System.Decimal
Boolean	System.Boolean
Date	System.DateTime
Char	System.Char
String	System.String

**Tabelle 16.1:** Primitive Visual Basic-Datentypen und ihre .NET-Äquivalente

Das bedeutet: Es ist vollkommen egal, ob Sie beispielsweise einen 32-Bit-Integer mit

```
Dim loc32BitInteger as Integer
```

oder mit

```
Dim loc32BitInteger as System.Int32
```

deklarieren. Die Objektvariable loc32BitInteger ist in beiden Fällen nicht nur vom gleichen, sondern vom selben<sup>1</sup> Typ. Wenn Sie sich den erzeugten IML-Code aus dem kleinen Programm

```
Public Shared Sub main()
```

```
    Dim locDate As Date = #12/14/2003#
    Dim locDate2 As DateTime = #12/14/2003 12:13:22 PM#
    If locDate > locDate2 Then
        Console.WriteLine("locDate ist größer als locDate2")
    Else
        Console.WriteLine("locDate2 ist größer als locDate")
    End If
```

anschauen, werden Sie anhand der generierten Codezeilen

```
.method public static void main() cil managed
{
    // Code size      75 (0x4b)
    .maxstack 2
    .locals init ([0] valuetype [mscorlib]System.DateTime locDate,
                 [1] valuetype [mscorlib]System.DateTime locDate2,
                 [2] bool VB$CG$t_bool$S0)
    IL_0000: nop
    IL_0001: ldc.i8    0x8c58fec59f98000
    IL_000a: newobj     instance void [mscorlib]System.DateTime::ctor(int64)
    IL_000f: nop
    IL_0010: stloc.0
    IL_0011: ldc.i8    0x8c59052cd35dd00
    IL_001a: newobj     instance void [mscorlib]System.DateTime::ctor(int64)
    IL_001f: nop
    IL_0020: stloc.1
    IL_0021: ldloc.0
    IL_0022: ldloc.1
    .
    .
}
```

feststellen, dass diese Tatsache zutrifft. Beide lokalen Variablen sind, wie in den fett markierten Zeilen zu sehen, als `System.DateTime`-Typ deklariert worden.

---

<sup>1</sup> Und zwar – auch auf die Gefahr hin, mich zu wiederholen – in der ursprünglichen Bedeutung des Unterschiedes von »selber« und »gleicher«.

# Numerische Datentypen

Für die Verarbeitung von Zahlen bietet Ihnen Visual Basic die primitiven Datentypen `Byte`, `Short`, `Integer`, `Long`, `Single`, `Double` und `Decimal` an. Neu in Visual Basic 2005 sind darüber hinaus die Datentypen `SByte`, `UShort`, `UInteger` und `ULong`. Sie unterscheiden sich durch den Wertebereich, den sie abdecken können, die Präzision, mit der sie rechnen (Anzahl Nachkommastellen – auch Skalierung genannt) und den Speicherbedarf, den sie benötigen.

## Numerische Datentypen deklarieren und definieren

Alle numerischen Datentypen werden wie alle primitiven Datentypen ohne das Schlüsselwort `New` deklariert; Zuweisungen von konstanten Wert können direkt im Programm erfolgen, wobei es bestimmte Markierungszeichen gibt, von welchem Typ ein konstanter Wert ist, der durch eine Ziffernfolge angegeben wird. Eine Variable vom Typ `Double` kann beispielsweise mit der Anweisung

```
Dim locDouble As Double
```

deklariert und sofort verwendet werden. Die Instanzbildung des `Double`-Objektes geschieht auf IML- bzw. Framework-Ebene.

Numerische Datentypen werden mit im Programmcode verankerten Konstanten definiert, indem Sie ihnen eine Ziffernfolge zuweisen, der im Bedarfsfall das Typliteral folgt, wie im folgenden Beispiel:

```
locDouble = 123.3D
```

Genau wie andere primitive Datentypen können Deklaration und Zuweisung in einer Anweisung erfolgen. So könnten Sie natürlich die beiden oben stehenden einzelnen Anweisungen durch die folgende ersetzen:

```
Dim locDouble As Double = 123.3D
```

Das hier gezeigte Beispiel gilt für alle anderen numerischen Datentypen äquivalent – wobei sich die Typliterale natürlich von Typ zu Typ unterscheiden können.

## Delegation numerischer Berechnungen an den Prozessor

Die Eigenschaft, einige mathematische Operationen dem Prozessor direkt zu überlassen, zeigt das folgende Beispiel eindrucksvoll. Dazu müssen Sie wissen: Der `Decimal`-Typ wird, anders als `Double` oder `Single`, auf Grund seiner Rechengenauigkeit nicht alleine durch die Fließkommaeinheit des Prozessors, sondern durch entsprechenden Programmcode der Base Class Library berechnet.

---

**BEGLEITDATEIEN:** Bevor Sie den folgenden Beispielcode (zu finden unter `.\VB 2005 - Entwicklerbuch\E - Datentypen\Kap16\Primitives01`) ausführen, setzen Sie mit der Taste **F9** einen Haltepunkt in der folgenden, fett markierten Codezeile:

---

```
Public Class Primitives
    Public Shared Sub main()
        Dim locDouble1, locDouble2 As Double
        Dim locDec1, locDec2 As Decimal
```

```

locDouble1 = 123.434D
locDouble2 = 321.121D
locDouble2 += 1
locDouble1 += locDouble2
Console.WriteLine("Ergebnis der Double-Berechnung: {0}", locDouble1)

locDec1 = 123.434D
locDec2 = 321.121D
locDec2 += 1
locDec1 += locDec2
Console.WriteLine("Ergebnis der Double-Berechnung: {0}", locDec1)

End Sub
End Class

```

Wenn Sie dieses Programm anschließend starten, unterbricht das Programm in der Zeile, in der Sie zuvor den Haltepunkt gesetzt haben. Wählen Sie aus dem Menüs *Debuggen/Fenster* den Punkt *Disassembly*. Mit Hilfe dieses Fensters können Sie sehen, was der JITter<sup>2</sup> aus dem zunächst in die IML kompilierten Programm gemacht hat:

```

locDouble2 += 1
00000048 fld1
0000004a fadd    qword ptr [ebp-68h]
0000004d fstp     qword ptr [ebp-68h]
locDouble1 += locDouble2
00000050 fld      qword ptr [ebp-68h]
00000053 fadd    qword ptr [ebp-60h]
00000056 fstp     qword ptr [ebp-60h]

```

Hier werden, entgegen der Erwartung, keine weiteren Methoden der Double-Struktur aufgerufen; die Addition wird vielmehr durch die Fließkommafunktionalität des Prozessors (**fadd**, im Listing fett markiert) selbst erledigt. Ganz anders ist das weiter unten in der Disassembly, wenn die gleichen Operationen mit dem Decimal-Datentyp ausgeführt werden:

```

locDec2 += 1
000000df lea      eax,[ebp-58h]
000000e2 sub     esp,10h
000000e5 movq    xmm0,mmword ptr [eax]
000000e9 movq    mmword ptr [esp],xmm0
000000ee movq    xmm0,mmword ptr [eax+8]
000000f3 movq    mmword ptr [esp+8],xmm0
000000f9 mov      eax,dword ptr ds:[023910B8h]
000000fe add     eax,4
00000101 cmp     ecx,dword ptr [eax]
00000103 sub     esp,10h
00000106 movq    xmm0,mmword ptr [eax]
0000010a movq    mmword ptr [esp],xmm0
0000010f movq    xmm0,mmword ptr [eax+8]

```

---

<sup>2</sup> Das Disassembly-Fenster kann übrigens nur Debug-Assembler-Code anzeigen, der nicht sehr gut optimiert ist. Selbst wenn Sie die Option *Starten ohne Debuggen* auswählen, sehen Sie immer nur den Debug- und nicht den optimierten Code.

```

00000114 movq    mmword ptr [esp+8],xmm0
0000011a lea     ecx,[ebp+FFFFFF68h]
00000120 call   7865DD88
00000125 lea     edi,[ebp-58h]
00000128 lea     esi,[ebp+FFFFFF68h]
0000012e movq    xmm0,mmword ptr [esi]
00000132 movq    mmword ptr [edi],xmm0
00000136 movq    xmm0,mmword ptr [esi+8]
0000013b movq    mmword ptr [edi+8],xmm0
locDec1 += locDec2
00000140 lea     eax,[ebp-48h]
00000143 sub    esp,10h
00000146 movq    xmm0,mmword ptr [eax]
0000014a movq    mmword ptr [esp],xmm0
0000014f movq    xmm0,mmword ptr [eax+8]
00000154 movq    mmword ptr [esp+8],xmm0
0000015a lea     eax,[ebp-58h]
0000015d sub    esp,10h
00000160 movq    xmm0,mmword ptr [eax]
00000164 movq    mmword ptr [esp],xmm0
00000169 movq    xmm0,mmword ptr [eax+8]
0000016e movq    mmword ptr [esp+8],xmm0
00000174 lea     ecx,[ebp+FFFFF58h]
0000017a call   7865DD88

```

Ungleich mehr Vorbereitungen zur Addition sind hier nötig, da die notwendigen Operanden zunächst auf den Stack kopiert werden müssen. Und hier wird die eigentliche Addition auch nicht durch den Prozessor selbst erledigt, sondern durch entsprechende Routinen der BCL, die, im Disassembly zu sehen, mit `Call` aufgerufen werden (im Listing fett markiert).

---

**TIPP:** Das ist übrigens auch der Grund, weshalb die Performance des Decimal-Datentyps auch nur etwa einem Zehntel der Performance des Double-Datentyps entspricht. Decimal sollten Sie nur dann einsetzen, wenn Sie absolut genaue Berechnungen durchführen müssen und sich keine Rundungsfehler erlauben können (lesen Sie dazu bitte auch die Ausführungen im ► Abschnitt »Rundungsfehler bei der Verwendung von Single und Double« auf Seite 451).

---

## Hinweis zur CLS-Konformität

Einige der neuen Datentypen in Visual Basic 2005 entsprechen nicht der so genannten CLS-Compliance<sup>3</sup>. Dazu gehören generische Datentypen als auch einige neue primitive Datentypen, die vorzeichenlose Integerwerte speichern (sowie der neue primitive Datentyp `SByte`). Methoden, die Typen benötigen oder zurückliefern, die als nicht *CLS-Compliant* gelten, sollten nicht offen gelegt werden, also nicht in Komponenten zur Verfügung gestellt werden, die auch andere .NET-Programmiersprachen verwenden können. Sie dürfen nämlich nicht davon ausgehen, dass diese Typen auch in allen .NET-Programmiersprachen »erreichbar« sind. Im Gegensatz zu Beta-Versionen von Visual Basic 2005 prüft Visual Basic übrigens nicht mehr automatisch auf CLS-Konformität. Falls Sie Komponenten vom VB-Compiler auf Konformität überprüfen lassen wollen, können Sie das so genannte `CLSCompliant`-Attribut auf Klassen-Ebene einsetzen, also etwa so:

---

<sup>3</sup> Etwa: »Entsprechung der Common Language Specification«.

```

<CLSCompliant(True)>
Public Class EineKlasse

    Private myMember As UShort

    Public Property NichtCLSEntsprechend() As UShort
        Get
            Return myMember
        End Get
        Set(ByVal value As UShort)
            myMember = value
        End Set
    End Property
End Class

```

Falls Sie eine einzelne Methode auf CLS-Compliance überprüfen wollen, funktioniert das äquivalent:

```

<CLSCompliant(True)> _
Public Shared Function EineNichtCLSComplianceMethode() As UShort
    Dim locTest As ClassLibrary1.EineKlasse
    locTest = New ClassLibrary1.EineKlasse
End Function

```

## Die numerischen Datentypen auf einen Blick

Die Verwendung numerischer Datentypen und ihre darstellbaren Wertebereiche finden Sie in den folgenden kurzen Abschnitten beschrieben.

---

**HINWEIS:** Der Punkt *Arithmetik durch den Prozessor* in der nachstehenden Aufstellung sagt aus, dass bestimmte arithmetische oder boolesche Operationen nicht durch Prozeduren des Frameworks, sondern durch den Prozessor ausgeführt werden; das äußert sich in einer äußerst schnellen Verarbeitung.<sup>4</sup>

---

### Byte

**.NET-Datentyp:** System.Byte

**Stellt dar:** Integer-Werte (Zahlen ohne Nachkommastellen) im angegebenen Wertebereich

**Wertebereich:** 0 bis 255

**Typliteral:** nicht vorhanden

**Speicherbedarf:** 1 Byte

**Arithmetik durch den Prozessor:** ja

**Deklaration und Beispielzuweisung:**

```

Dim einByte As Byte
einByte = 123

```

---

<sup>4</sup> Diese Aussage gilt unter Umständen *nicht* für andere als die Intel Pentium Plattform. Auf Pocket-PCs beispielsweise, auf denen Applikationen unter .NET ebenfalls entwickelt werden können, kann sich das unter Umständen anders verhalten.

**Anmerkung:** Dieser Datentyp speichert nur vorzeichenlose, positive Zahlen im angegebenen Zahlbereich.

**CLS-Compliant:** ja

**Konvertierung von anderen Zahlentypen:** CByte(objVar) oder Convert.ToByte(objVar)

```
einByte = CByte(123.45D)  
einByte = Convert.ToByte(123.45D)
```

## SByte

**.NET-Datentyp:** System.SByte

**Stellt dar:** Integer-Werte (Zahlen ohne Nachkommastellen) im angegebenen Wertebereich

**Wertebereich:** -128 bis 127

**Typliteral:** nicht vorhanden

**Speicherbedarf:** 1 Byte

**Arithmetik durch den Prozessor:** ja

**Deklaration und Beispielzuweisung:**

```
Dim einByte As SByte  
einByte = 123
```

**Anmerkung:** Dieser Datentyp speichert negative und positive Zahlen im angegebenen Zahlbereich.

**CLS-Compliant:** nein

**Konvertierung von anderen Zahlentypen:** CSByte(objVar) oder Convert.ToSByte(objVar)

```
einByte = CByte(123.45D)  
einByte = Convert.ToByte(123.45D)
```

## Short

**.NET-Datentyp:** System.Int16

**Stellt dar:** Integer-Werte (Zahlen ohne Nachkommastellen) im angegebenen Wertebereich

**Wertebereich:** -32.768 bis 32.767

**Typliteral:** S

**Speicherbedarf:** 2 Byte

**Delegation an den Prozessor:** ja

**Deklaration und Beispielzuweisung:**

```
Dim einShort As Short  
einShort = 123S
```

**Anmerkung:** Dieser Datentyp speichert vorzeichenbehaftete, also negative und positive Zahlen im angegebenen Zahlbereich. Bei der Konvertierung in den Datentyp Byte kann durch den größeren Wertebereich von Short eine OutOfRangeException erzeugt werden.

**CLS-Compliant:** ja

**Konvertierung von anderen Zahlentypen:** CShort(objVar) oder Convert.ToInt16(objVar)

```
'Nachkommastellen werden abgeschnitten  
einShort = CShort(123.45D)  
einShort = Convert.ToInt16(123.45D)
```

## UShort

**.NET-Datentyp:** System.UInt16

**Stellt dar:** positive Integer-Werte (Zahlen ohne Nachkommastellen) im angegebenen Wertebereich

**Wertebereich:** 0 bis 65.535

**Typliteral:** US

**Speicherbedarf:** 2 Byte

**Delegation an den Prozessor:** ja

**Deklaration und Beispielzuweisung:**

```
Dim einUShort As UShort  
einUShort = 123US
```

**Anmerkung:** Dieser Datentyp speichert vorzeichenlose, also nur positive Zahlen im angegebenen Zahlenbereich. Bei der Konvertierung in den Datentyp Byte oder Short kann durch den (teilweise) größeren Wertebereich von UShort eine OutOfRangeException erzeugt werden.

**CLS-Compliant:** Nein

**Konvertierung von anderen Zahlentypen:** CUShort(objVar) oder Convert.ToUInt16(objVar)

```
'Nachkommastellen werden abgeschnitten  
einUShort = CUShort(123.45D)  
einUShort = Convert.ToUInt16(123.45D)
```

## Integer

**.NET-Datentyp:** System.Int32

**Stellt dar:** Integer-Werte (Zahlen ohne Nachkommastellen) im angegebenen Wertebereich

**Wertebereich:** -2.147.483.648 bis 2.147.483.647

**Typliteral:** I

**Speicherbedarf:** 4 Byte

**Delegation an den Prozessor:** ja

**Deklaration und Beispielzuweisung:**

```
Dim einInteger As Integer  
Dim einAndererInteger%      ' auch als Integer deklariert  
einInteger = 123I
```

**Anmerkung:** Dieser Datentyp speichert vorzeichenbehaftete, also negative und positive Zahlen im angegebenen Zahlenbereich. Bei der Konvertierung in die Datentypen Byte, Short und UShort kann durch den größeren Wertebereich von Integer eine OutOfRangeException erzeugt werden. Durch Anhängen des Zeichens »%« an eine Variable kann der Integer-Typ für die Variable erzwungen werden (darauf sollten Sie allerdings zugunsten eines besseren Programmierstils lieber verzichten).

**CLS-Compliant:** ja

**Konvertierung von anderen Zahlentypen:** CInt(objVar) oder Convert.ToInt32(objVar)

```
einInteger = CInt(123.45D)  
einInteger = Convert.ToInt32(123.45D)
```

## UInteger

**.NET-Datentyp:** System.UInt32

**Stellt dar:** positive Integer-Werte (Zahlen ohne Nachkommastellen) im angegebenen Wertebereich

**Wertebereich:** 0 bis 4.294.967.295

**Typliteral:** UI

**Speicherbedarf:** 4 Byte

**Delegation an den Prozessor:** ja

**Deklaration und Beispielzuweisung:**

```
Dim einUInteger As UInteger  
einUInteger = 123UI
```

**Anmerkung:** Dieser Datentyp speichert vorzeichenlose, also nur positive Zahlen im angegebenen Zahlenbereich. Bei der Konvertierung in die Datentypen Byte, Short, UShort und Integer kann durch den (teilweise) größeren Wertebereich von UInteger eine OutOfRangeException erzeugt werden.

**CLS-Compliant:** nein

**Konvertierung von anderen Zahlentypen:** CUInt(objVar) oder Convert.ToUInt32(objVar)

```
einUInteger = CUInt(123.45D)  
einUInteger = Convert.ToUInt32(123.45D)
```

## Long

**.NET-Datentyp:** System.Int64

**Stellt dar:** Integer-Werte (Zahlen ohne Nachkommastellen) im angegebenen Wertebereich

**Wertebereich:** -9.223.372.036.854.775.808 bis 9.223.372.036.854.775.807

**Typliteral:** L

**Speicherbedarf:** 8 Byte

**Delegation an den Prozessor:** ja

### Deklaration und Beispielzuweisung:

```
Dim einLong As Long  
Dim einAndererLong& ' auch als Long definiert  
einLong = 123L
```

**Anmerkung:** Dieser Datentyp speichert vorzeichenbehaftete, also negative und positive Zahlen im angegebenen Zahlenbereich. Bei der Konvertierung in alle anderen Integer-Datentypen kann durch den größeren Wertebereich von Long eine OutOfRangeException erzeugt werden. Durch Anhängen des Zeichens »&« an eine Variable kann der Long-Typ für die Variable erzwungen werden (darauf sollten Sie allerdings zugunsten eines besseren Programmierstils lieber verzichten).

**CLS-Compliant:** ja

**Konvertierung von anderen Zahlentypen:** CLng(objVar) oder Convert.ToInt64(objVar)

```
einLong = CLng(123.45D)  
einLong = Convert.ToInt64(123.45D)
```

### ULong

**.NET-Datentyp:** System.UInt64

**Stellt dar:** positive Integer-Werte (Zahlen ohne Nachkommastellen) im angegebenen Wertebereich

**Wertebereich:** 0 bis 18.446.744.073.709.551.615

**Typliteral:** UL

**Speicherbedarf:** 8 Byte

**Delegation an den Prozessor:** ja

### Deklaration und Beispielzuweisung:

```
Dim einULong As ULong  
einULong = 123L
```

**Anmerkung:** Dieser Datentyp speichert vorzeichenlose, also nur positive Zahlen im angegebenen Zahlenbereich. Bei der Konvertierung in alle anderen Integer-Datentypen kann durch den größeren Wertebereich von Long eine OutOfRangeException erzeugt werden.

**CLS-Compliant:** nein

**Konvertierung von anderen Zahlentypen:** CULng(objVar) oder Convert.ToUInt64(objVar)

```
einULong = CULng(123.45D)  
einULong = Convert.ToUInt64(123.45D)
```

### Single

**.NET-Datentyp:** System.Single

**Stellt dar:** Fließkommawerte (Zahlen mit Nachkommastellen, deren Skalierung<sup>5</sup> mit Anwachsen des Wertes kleiner wird) im angegebenen Wertebereich

---

<sup>5</sup> Skalierung in diesem Zusammenhang bezeichnet die Anzahl der Nachkommastellen einer Fließkommazahl.

**Wertebereich:** Die Werte reichen von  $-3,4028235 \cdot 10^{38}$  bis  $-1,401298 \cdot 10^{-45}$  für negative Werte und von  $1,401298 \cdot 10^{-45}$  bis  $3,4028235 \cdot 10^{38}$  für positive Werte.

**Typliteral:** F

**Speicherbedarf:** 4 Byte

**Delegation an den Prozessor:** ja

**Deklaration und Beispielzuweisung:**

```
Dim einSingle As Single  
Dim einAndererSingle! ' auch als Single definiert  
einSingle = 123.0F
```

**Anmerkung:** Dieser Datentyp speichert vorzeichenbehaftete, also negative und positive Zahlen im angegebenen Zahlenbereich. Durch Anhängen des Zeichens »!« an eine Variable kann der Single-Typ für die Variable erzwungen werden (darauf sollten Sie allerdings zugunsten eines besseren Programmierstils lieber verzichten).

**CLS-Compliant:** ja

**Konvertierung von anderen Zahlentypen:** CSng(objVar) oder Convert.ToSingle(objVar)

```
einSingle = CSng(123.45D)  
einSingle = Convert.ToSingle(123.45D)
```

## Double

**.NET-Datentyp:** System.Double

**Stellt dar:** Fließkomma-Werte (Zahlen mit Nachkommastellen, deren Skalierung mit Anwachsen des Wertes kleiner wird) im angegebenen Wertebereich

**Wertebereich:** Die Werte reichen von  $-1,79769313486231570 \cdot 10^{308}$  bis  $-4,94065645841246544 \cdot 10^{-324}$  für negative Werte und von  $4,94065645841246544 \cdot 10^{-324}$  bis  $1,79769313486231570 \cdot 10^{308}$  für positive Werte

**Typliteral:** R

**Speicherbedarf:** 8 Byte

**Delegation an den Prozessor:** ja

**Deklaration und Beispielzuweisung:**

```
Dim einDouble As Double  
Dim einAndererDouble# ' auch als Double definiert  
einDouble = 123.0R
```

**Anmerkung:** Dieser Datentyp speichert vorzeichenbehaftete, also negative und positive Zahlen im angegebenen Zahlenbereich. Durch Anhängen des Zeichens »#« an eine Variable kann der Double-Typ für die Variable erzwungen werden (darauf sollten Sie allerdings zugunsten eines besseren Programmierstils lieber verzichten).

**CLS-Compliant:** ja

**Konvertierung von anderen Zahlentypen:** CDbL(objVar) oder Convert.ToDouble(objVar)

```
einDouble = CDb1(123.45D)  
einDouble = Convert.ToDouble(123.45D)
```

## Decimal

**.NET-Datentyp:** System.Decimal

**Stellt dar:** Fließkomma-Werte (Zahlen mit Nachkommastellen, deren Skalierung mit Anwachsen des Wertes kleiner wird) im angegebenen Wertebereich

**Wertebereich:** Der Wertebereich hängt von der Anzahl der verwendeten Dezimalstellen ab. Werden keine Dezimalstellen verwendet – man spricht dabei von einer Skalierung von 0 –, liegen die maximalen/minimalen Werte zwischen  $+79.228.162.514.264.337.593.543.950.335$ . Bei der Verwendung einer maximalen Skalierung (28 Stellen hinter dem Komma – es können dann nur noch Werte zwischen  $>-1$  und  $<+1$  dargestellt werden) liegen die maximalen/minimalen Werte zwischen  $+0.99999999999999999999999999999999$ .

**Typliteral:** D

Speicherbedarf: 16 Byte

### **Delegation an den Prozessor: nein**

## Deklaration und Beispielzuweisung:

```
Dim einDecimal As Decimal
```

```
Dim einAndererDouble@ ' auch als Decimal definiert  
einDecimal = 123.23D
```

**Anmerkung:** Dieser Datentyp speichert vorzeichenbehaftete, also negative und positive Zahlen im angegebenen Zahlbereich. Durch Anhängen des Zeichens »@« an eine Variable kann der Decimal-Typ für die Variable erzwungen werden (darauf sollten Sie allerdings zugunsten eines besseren Programmierstils lieber verzichten). Wichtig: Bei sehr hohen Werten müssen Sie das Typliteral an eine Literalkonstante anhängen, um eine Overflow-Fehlermeldung zu vermeiden.

**HINWEIS:** Bitte beachten Sie ebenfalls, dass beim Decimal-Datentyp keine Delegation arithmetischer Funktionen an den Prozessor erfolgt, dieser Datentyp also im Vergleich zu den Fließkommadatentypen Single und Double sehr viel langsamer verarbeitet wird. Gleichzeitig treten aber keine Rundungsfehler durch die interne Darstellung von Werten im Binärsystem auf. Darüber erfahren Sie im folgenden Abschnitt Näheres.

**CLS-Compliant:** ja

**Konvertierung von anderen Zahlentypen:** CDec(objVar) oder Convert.ToDecimal(objVar)

```
einDecimal = CDec(123.45F)  
einDecimal = Convert.ToDecimal(123.45F)
```

## Rundungsfehler bei der Verwendung von Single und Double

Es ist eigentlich eine ganz normale Sache, dass ein bestimmtes Zahlensystem einige Brüche nicht genau darstellen kann. Dennoch gibt es immer wieder Programmierer, die glauben, einen Fehler in einer Programmiersprache gefunden zu haben, oder behaupten, der Computer könne nicht richtig

rechnen. Dabei kennen Sie Rundungs- bzw. Konvertierungsfehler von einem Zahlensystem in das andere aus dem täglichen Leben auch beim 10er-System: Wenn Sie die Zahl 1 durch 3 teilen, erhalten Sie eine Zahl mit unendlichen Nachkommastellen, nämlich 0,333333333333. Im Dreiersystem ein Drittel darzustellen, benötigt wesentlich weniger Ziffern. Es ist schlicht 0,1.

Nun ist es ganz gleich, wie viele Ziffern Sie für die Darstellung eines für ein Zahlensystem problematischen Bruches verwenden; solange Sie eine endliche Anzahl von Ziffern in einem Zahlensystem verwenden, das einen Bruch nur periodisch darstellen kann, erhalten Sie beim Addieren dieser Zahlen Rundungsfehler.

Ein Beispiel:  $3*1/3$  im Dreiersystem führt zur Berechnung von:

0.1  
+0.1  
+0,1  
=====

Und das entspricht im Dezimalsystem ebenfalls 1,0. Das Ausrechnen dieser Addition im Dezimalsystem ist ungenau, denn selbst wenn Sie über 60 Nachkommastellen für die Darstellung der Zahlen verwenden, so erreichen Sie in der Addition dennoch niemals den Wert 1:

Dieser Wert ist zwar ziemlich nah dran an 1, aber eben nicht ganz 1. Und wenn Sie mehrere Ergebnisse im Laufe einer Berechnung haben, können sich diese Darstellungsfehler schnell zu größeren Fehlern summieren, die auch irgendwann relevant werden.

Das gleiche Problem hat der Computer bei bestimmten Zahlen, wenn er im Binärsystem rechnet. Während wir beispielsweise die Zahl 69,82 im Dezimalsystem ganz genau mit einer endlichen Anzahl von Ziffern darstellen können, bekommt der Computer mit dem Binärsystem Probleme:

Die Umwandlung von 69 funktioniert noch einwandfrei, aber bei der 0,82 wird es schwierig:

Wenn Sie wissen, dass Nachkommastellen durch negative Potenzen der Basiszahl dargestellt werden, dann ergibt sich folgende Rechnung:

0.5	$1*2^{-1}$	Zwischenergebnis: 0.5
0.25	$1*2^{-2}$	Zwischenergebnis: 0.75
0.125	$1*2^{-3}$	Zwischenergebnis: 0.8125
0.0625	$0*2^{-4}$	Zwischenergebnis: 0.8125
0.03125	$0*2^{-5}$	Zwischenergebnis: 0.8125
0.015625	$0*2^{-6}$	Zwischenergebnis: 0.8125
0.0078125	$0*2^{-7}$	Zwischenergebnis: 0.8125
0.00390625	$1*2^{-8}$	Zwischenergebnis: 0.81640625
0.001953125	$1*2^{-9}$	Zwischenergebnis: 0.818359375
0.0009765625	$1*2^{-10}$	Zwischenergebnis: 0.8193359375
0.00048828125	$1*2^{-11}$	Zwischenergebnis: 0.81982421875

Wir sind inzwischen bei der Zahl 0,11100001111 angelangt und haben das gewünschte Ziel immer noch nicht erreicht. Die Wahrheit ist: Sie können dieses Spielchen bis in alle Ewigkeit weiterspielen.

Sie werden die Zahl 0,82 des Dezimalsystems mit einer endlichen Anzahl an Ziffern im Binärsystem niemals darstellen können.

---

**BEGLEITDATEIEN:** Was hat das für Auswirkungen auf die Programmierung unter Visual Basic? Nun, schauen Sie sich dazu einmal das folgende kleine Beispielprogramm an, das Sie im Verzeichnis `\VB 2005 - Entwicklerbuch\E - Datentypen\Kap16\Primitives02` finden können. (Starten Sie das Programm mit **Strg + F5**.)

---

```
Public Class Primitives
    Public Shared Sub main()

        Dim locDouble1, locDouble2 As Double
        Dim locDec1, locDec2 As Decimal

        locDouble1 = 69.82
        locDouble2 = 69.2
        locDouble2 += 0.62

        Console.WriteLine("Die Aussage locDouble1=locDouble2 ist {0}", locDouble1 = locDouble2)
        'Console.WriteLine("locDouble1 lautet {0}; locDouble2 lautet {1}", locDouble1, locDouble2)

        locDec1 = 69.82D
        locDec2 = 69.2D
        locDec2 += 0.62D
        Console.WriteLine("Die Aussage locDec1=locDec2 ist {0}", locDec1 = locDec2)
    End Sub
End Class
```

Auf den ersten Blick sollte man meinen, dass beide `WriteLine`-Methoden den gleichen Text ausgeben. Sie brauchen keinen Taschenrechner zu bemühen, um zu sehen, dass der erste Wert und damit die erste Variable innerhalb des Programms die Addition des zweiten und dritten Wertes darstellt und die beiden Variablenwerte aus diesem Grund gleich sein sollten. Leider ist dem nicht so. Während Sie mit dem `Decimal`-Datentyp im zweiten Teil des Programms den richtigen Wert herausbekommen, versagt der `Double`-Typ im ersten Part des Programms. Der Grund dafür ist genau der zuvor beschriebene.

Noch verwirrender wird es, wenn Sie die zweite, auskommentierte `WriteLine`-Methode wieder ins Programm nehmen: Beide Variablen enthalten nämlich augenscheinlich den gleichen Wert, bis auf die letzte Nachkommastelle genau. Das Geheimnis darum ist aber schnell gelüftet: Bei der Umwandlung in eine Zeichenkette findet eine Rundung statt, die über das wahre Ergebnis hinwegtäuscht.

Aus dieser Tatsache leiten sich folgende Grundsätze ab:

- Vermeiden Sie es nach Möglichkeit, innerhalb von Schleifen gebrochene `Double`- oder `Single`-Werte zu verwenden. Sie laufen sonst Gefahr, dass sich Ihr Programm auf Grund der beschriebenen Ungenauigkeiten in Endlosschleifen verrennt.
- Verwenden Sie `Single`- und `Double`-Datentypen nur dort, wo es nicht auf die x-te Stelle hinter dem Komma ankommt. Bei der Berechnung von Grafiken beispielsweise, wo Rundungsfehler durch eine geringere Bildschirmauflösung als die Rechenpräzision ohnehin keine Rolle spielen, sollten Sie immer die schnelleren, prozessorberechneten Datentypen `Single` und `Double` dem manuell berechneten `Decimal`-Datentyp vorziehen.

- Bei finanztechnischen Anwendungen sollten Sie in jedem Fall den `Decimal`-Datentyp einsetzen. Nur mit ihm ist gewährleistet, dass Additionen und andere Berechnungen von nicht exakt darstellbaren Zahlen nicht in größere Fehler münden.
- Verwenden Sie andererseits den `Decimal`-Datentyp, wenn es eben geht, nie in Schleifen und setzen Sie ihn schon gar nicht als Zählvariable ein. Er erfährt nämlich keine Unterstützung durch den Prozessor und bremst ihr Programm extrem aus! Verwenden Sie dort nach Möglichkeit nur eine der zahlreichen Integer-VariablenTypen.
- Wenn Sie – aus Geschwindigkeitsgründen – dennoch `Double`- oder `Single`-Typen auf Gleichheit testen müssen, arbeiten Sie besser mit einer Abfrage des Deltas, etwa:

```
If Math.Abs(locDouble1 - locDouble2) < 0.0001 then
    'Werte sind annähernd dieselben, also quasi gleich.
End If
```

## Besondere Funktionen, die für alle numerischen Datentypen gelten

Alle numerischen Datentypen verfügen über Methoden, die bei allen Typen nach der gleichen Vorgehensweise verwendet werden. Sie dienen zur Umwandlung einer Zeichenkette (eine Ziffernfolge) in den entsprechenden Wert sowie zur Umwandlung des Wertes in eine Zeichenkette. Mit anderen Funktionen können Sie den größten oder kleinsten Wert ermitteln, den ein Datentyp darstellen kann.

### Zeichenketten in Werte Wandeln und Vermeiden von kulturabhängigen Fehlern

Zum Umwandeln einer Zeichenkette in einen Wert dienen die statischen Funktion `Parse` oder `TryParse`, die jedem numerischen Datentyp zur Verfügung steht. Um beispielsweise die Ziffernfolge »123« in den Integerwert 123 umzuwandeln, genügen die folgenden Anweisungen:

```
Dim locInteger As Integer
locInteger = Integer.Parse("123")
```

**WICHTIG:** Allerdings können Sie seit Visual Basic 2005 nicht mehr

```
locInteger = locInteger.Parse("123") ' Geht nicht mehr!
```

verwenden, denn `Parse` ist eine statische Funktion und sollte nicht mehr über eine Objektvariable sondern nur über den entsprechenden Klassennamen angesprochen werden. Zwar ließe sich das Programm beim Ansprechen der statischen Funktion über eine Objektvariable noch kompilieren, bereits im Codeeditor würden Sie aber eine Warnung in der Fehlerliste sehen.

Es gibt auch die Möglichkeit des Umwandlungsversuchs einer Zeichenkette in einen numerischen Wert, wie es das folgende Beispiel zeigt:

```
Dim locInteger As Integer
If Integer.TryParse("123", locInteger) Then
    'Umwandlung war erfolgreich
Else
    'Umwandlung war nicht erfolgreich
End If
```

Bei erfolgreicher Umwandlung steht die umgewandelte Zahl anschließend in der `TryParse` übergebenden Variablen.

Auch das Framework-Äquivalent von Integer ermöglicht die Umwandlung durch

```
locInteger = System.Int32.Parse("123") ' Und auch das ginge.
```

und um die Liste komplett zu machen, geht es natürlich auch über die Convert-Klasse im Framework-Stil mit

```
locInteger = Convert.ToInt32("123") ' Und das ginge.
```

und in alter Visual Basic-Manier täte es die Anweisung

```
locInteger = CInt("123") ' Letzte Möglichkeit, mehr fallen mir nicht ein.
```

ebenfalls.

Aber aufgepasst: Wenn Sie das folgende Programm auf einem deutschen System starten, passiert möglicherweise nicht das, was Sie erwarten:

```
Dim locString As String = "123.23"
Dim locdouble As Double = Double.Parse(locString)
Console.WriteLine(locdouble.ToString)
```

Sie rechnen vielleicht damit, dass die Ziffernfolge korrekt in den Wert 123,23 umgewandelt wird. Anstelle dessen gibt das Programm

12323

aus – definitiv nicht das Ergebnis, das Sie erwartet haben. Starten Sie das Programm auf einem englischen System, ist das Ergebnis korrekt und wie erwartet.<sup>6</sup>

123.23

Na ja, vielleicht nicht ganz. Wir Deutschen haben uns angewöhnt, die Nachkommastellen von den Vorkommastellen mit einem Komma zu trennen (vermutlich heißen sie auch deshalb *Nachkommastellen*). Englischsprachige Länder machen das allerdings mit einem Punkt, und die Ausgabe, die Sie über diesem Absatz sehen, ist eine korrekte englische Formatierung. Welche Auswirkungen hat dieses Verhalten auf Ihre Programme? Nun, zunächst einmal sollten Sie es unbedingt vermeiden, im Programmcode selbst numerische Konstanten als String zu speichern, wenn Sie sie später in einen numerischen Typ wandeln wollen (wie im letzten Beispiel gezeigt). Wenn Sie numerische Datentypen innerhalb Ihres Programms definieren, dann machen Sie es bitte grundsätzlich nur im Code direkt, nicht mit Zeichenketten (in Anführungszeichen) und deren Umwandlungsfunktionen. Sie haben sicherlich schon festgestellt, dass im Code abgelegte Ziffernfolgen (ohne Anführungszeichen) zur Zuweisung eines Wertes grundsätzlich nur im englischen Format abgelegt werden.

Solange Sie keine Dateien mit als Text gespeicherten Informationen, aus denen Ihr Programm Werte generieren muss, über Kulturgrenzen hinweg austauschen müssen, haben Sie nichts zu befürchten: Läuft Ihre Anwendung auf einem englischen System, werden Zahlen mit Punkt als Trennzeichen in die Datei geschrieben, hier im deutschsprachigen Raum eben als Komma. Da die Kultureinstellungen beim Einlesen äquivalent berücksichtigt werden, kann Ihre Anwendung auch die richtigen Werte aus der Textdatei zurückgenerieren.

---

<sup>6</sup> Wobei dieses Verhalten streng genommen nicht dadurch bedingt wird, dass es sich um ein englisches System handelt, sondern dass ein englisches Betriebssystem voreingestellt andere Ländereinstellungen aufweist als ein deutsches. Natürlich könnten Sie auch ein deutsches Betriebssystem so konfigurieren, dass es das gleiche Ergebnis liefert.

Problematisch wird es dann, wenn auch die Dateien mit den Texten über Kulturgrenzen hinweg ausgetauscht werden sollen. Dann würde eine USA-Plattform die Datei mit Punkt als Trennzeichen exportieren und hier in Deutschland würde die Parse-Funktion den Punkt als Tausenderpunkt ansehen, damit unter den Tisch fallen lassen und so fälschlicherweise den Wert verhundertfachen. In diesem Fall müssen Sie dafür sorgen, dass der Export in eine Textdatei kulturreutral erfolgt, und das können Sie auf folgende Weise erreichen:

Sowohl die Parse-Funktion als auch die ToString-Funktion aller numerischen Typen können bei der Umwandlung durch einen so genannten *Format Provider* (etwa: Formatanbieter) spezifisch gesteuert werden. Es gibt die unterschiedlichsten Format Provider in .NET für numerische Typen, nämlich auf der einen Seite solche, mit denen Sie Formate in Abhängigkeit von der Anwendung (finanztechnisch, wissenschaftlich, etc.) oder in Abhängigkeit von der Kultur steuern können. Dazu dienen die Klassen NumberFormatInfo und CultureInfo. Beiden lassen sich, zuvor entsprechend instanziert und aufbereitet, als Parameter sowohl der ToString- als auch der Parse-Funktion übergeben. Genaueres über den Umgang mit diesen Klassen erfahren Sie im nächsten Kapitel; für den Moment soll das folgende Beispiel genügen, das demonstriert, wie Sie erzwingen, dass Umwandlungen nicht von der aktuellen Kultur abhängig gemacht werden, sondern kulturreutral erfolgen.

---

**WICHTIG:** Diese Vorgehensweise, wie sie im Folgenden zu sehen ist, sollten Sie immer bei Anwendungen anwenden, die mit internationalem Anspruch entwickelt werden, um Fehler bei Typkonvertierungen von vorneherein auszuschließen!

---

```
Dim locString As String = "123.23"
Dim locdouble As Double

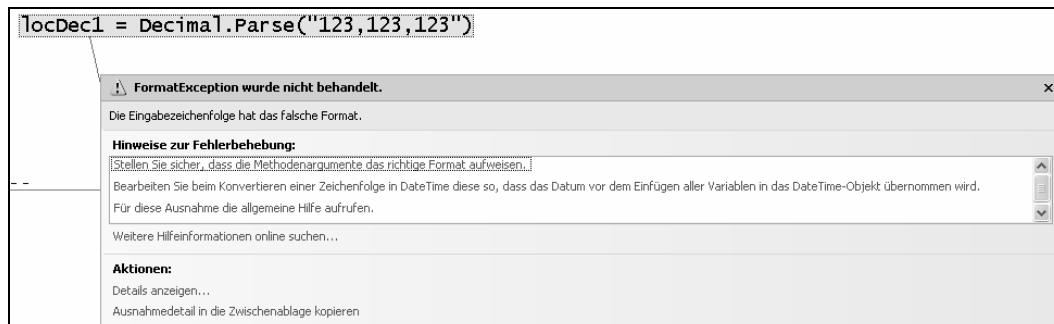
locdouble = Double.Parse(locString, CultureInfo.InvariantCulture)
Console.WriteLine(locdouble.ToString(CultureInfo.InvariantCulture))
Console.ReadLine()
```

---

**HINWEIS:** Damit Sie auf kulturbezogene Klassen und Funktionen zugreifen können, müssen Sie am Anfang des Programms den entsprechenden Namensbereich System.Globalization mit Imports eingebunden haben, etwa so:

---

```
Imports System.Globalization
```



**Abbildung 16.1:** Falls die Zeichenkette zur Umwandlung auf Grund ihres Formates nicht umgewandelt werden kann, generiert das Framework eine Ausnahme

Die statische Eigenschaft `InvariantCulture` liefert direkt eine Instanz einer `CultureInfo`-Klasse zurück, die mit den entsprechenden Eigenschaften bestückt wurde.

---

**HINWEIS:** Wenn der Umwandlungsversuch fehlschlägt, da die Zeichenkette schlicht und ergreifend kein konvertierbares Format enthält und sich deswegen nicht in einen Wert umwandeln lässt, generiert das Framework eine Ausnahme (siehe Abbildung 16.1). Sie können die Ausnahme entweder mit Try/Catch abfangen oder alternativ die statische Funktion `TryParse` (siehe unten) verwenden, die grundsätzlich keine Ausnahme beim Konvertierungsversuch erzeugt.

### Ermitteln von minimal und maximal darstellbarem Wert eines numerischen Typs

Die numerischen Datentypen kennen zwei spezielle statische Eigenschaften, mit denen Sie jeweils den größten und kleinsten darstellbaren Wert ermitteln lassen können. Die Eigenschaften lauten `MinValue` und `MaxValue`, und Sie können sie wie jede statische Funktion entweder durch den Typennamen selbst oder durch eine Objektvariable des Typs aufrufen. Beispiel:

```
Dim locInteger As Integer  
Dim locdouble As Double  
Dim locDecimal As System.Decimal  
  
Console.WriteLine(locInteger.MaxValue)  
Console.WriteLine(Double.MinValue)  
Console.WriteLine(locDecimal.MaxValue)
```

## Spezielle Funktionen der Fließkommatypen

Die Fließkommatypen verfügen über einige besondere Eigenschaften zur Darstellung von speziellen Werten, auf deren Zustand Sie eine Objektvariable mit entsprechenden Funktionen überprüfen können.

### Unendlich (Infinity)

Wenn Sie einen Fließkommawertetyp durch 0 teilen, dann erzeugen Sie damit keine Ausnahme (keinen Fehler). Vielmehr ist das Ergebnis *unendlich (infinity)*, und sowohl `Single` als auch `Double` können dieses Ergebnis darstellen, wie das folgende Beispiel demonstriert:

```
Dim locdouble As Double  
locdouble = 20  
locdouble /= 0  
Console.WriteLine(locdouble)  
Console.WriteLine("Die Aussage locDouble ist +unendlich ist {0} .", locdouble = Double.PositiveInfinity)
```

Wenn Sie dieses Beispiel ausführen, erzeugt es keine Fehlermeldung in Form einer Ausnahme, sondern das Programm schreibt vielmehr ein Ergebnis auf den Bildschirm:

```
+unendlich  
Die Aussage locDouble ist +unendlich ist True.
```

Anstatt den Vergleich auf Unendlichkeit durch den Vergleichsoperator durchzuführen, können Sie auch die statische Funktion `IsInfinity` verwenden:

```
Console.WriteLine("Die Aussage locDouble ist +unendlich ist {0} .", locdouble.IsInfinity(locdouble))
```

Das hat den Vorteil, dass Sie im Vorfeld nicht wissen müssen, ob ein Ergebnis positiv oder negativ unendlich ist. Mit den Funktionen `IsPositiveInfinity` und `IsNegativeInfinity` können Sie dennoch die Differenzierung innerhalb einer Abfrage vornehmen.

Um einer Variable gezielt den Wert unendlich zuzuweisen, verwenden Sie die statischen Funktionen `PositiveInfinity` und `NegativeInfinity`, die entsprechende Konstanten zurückliefern.

### Keine Zahl (`NaN`, Not a Number)

Und noch einen Sonderfall decken die primitiven Fließkommatypen ab: Die Division von 0 und 0, die mathematisch nicht definiert ist und *keine gültige Zahl* ergibt:

```
'Sonderfall: 0/0 ist mathematisch nicht definiert und ergibt "Not a Number"
einDouble = 0
einDouble = einDouble / 0
If Double.IsNaN(einDouble) Then
    Debug.Print("einDouble ist keine Zahl!")
End If
```

Ließen Sie diesen Code laufen, würde der Text in der If-Abfrage ausgegeben werden.

---

**WICHTIG:** Überprüfungen auf diese Sonderwerte lassen sich nur durch die Eigenschaften testen, die statische Funktionen direkt an den Typen »hängen«. Zwar können Sie beispielsweise mit der Konstante der Fließkommatypen `NaN` den Wert »keine gültige Zahl« einer Variablen zuweisen; diese Konstante eignet sich allerdings nicht, auf diesen Zustand (»Wert«) zu testen, wie das folgende Beispiel zeigt:

---

```
Dim einDouble As Double

'Sonderfall: 0/0 ist mathematisch nicht definiert und ergibt "Not a Number"
einDouble = 0
einDouble = einDouble / 0

'Der Text sollte erwartungsgemäß ausgegeben werden,
'wird er aber nicht!
If einDouble = Double.NaN Then
    Debug.Print("Test 1:einDouble ist keine Zahl!")
End If

'Nur so kann der Test erfolgen!
If Double.IsNaN(einDouble) Then
    Debug.Print("Test 2:einDouble ist keine Zahl!")
End If
```

In diesem Beispiel würde nur der zweite Text ausgegeben.

### Versuchte Umwandlungen mit TryParse

Alle numerischen Datentypen kennen die statische Funktion `TryParse`, die versucht, eine Zeichenkette in einen Wert umzuwandeln. Im Gegensatz zu `Parse` erzeugt sie allerdings keine Ausnahme, wenn die Umwandlung nicht gelingt. Vielmehr übergeben Sie ihr eine Variable als Referenz zur Speicherung des Rückgabewertes, und das Funktionsergebnis informiert Sie darüber, ob die Konvertierung gelang (`True`) oder nicht (`False`):

```

Dim locdouble As Double
Dim locString As String = "Einhundertdreiundzwanzig"

'locdouble = Double.Parse(locString) ' Ausnahme
'Keine Ausnahme:
Console.WriteLine("Konvertierung erfolgreich? {0}", _
    Double.TryParse(locString, NumberStyles.Any, New CultureInfo("de-DE"), locdouble))

```

## Spezielle Funktionen des Wertetyps Decimal

Der Wertetyp *Decimal* verfügt ebenfalls über spezielle Funktionen, von denen viele allerdings in Visual Basic nicht zur Anwendung kommen (nichtsdestotrotz können Sie sie verwenden, aber es ergibt wenig Sinn – tatsächlich wurden sie für andere Sprachen, die eine Operatorenüberladung [noch] nicht kennen aufgenommen). Nehmen wir als Beispiel die statische Add-Funktion, die zwei Zahlen vom Typ *Decimal* addiert. Sie liefert als Ergebnis ein *Decimal* zurück. Stattdessen können Sie aber auch auf den + -Operator von Visual Basic zurückgreifen, der ebenfalls zwei Zahlen vom Typ *Decimal* addieren kann – und das natürlich für die spätere Nacharbeitung in einem viel leichter lesbaren Code. Lediglich die Funktionen der folgenden Tabelle sind sinnvoll im Gebrauch:

Funktionsname	Aufgabe
Remainder(Dec1, Dec2)	Ermittelt den Rest der Division der beiden <i>Decimal</i> -Werte.
Round(Dec, Integer)	Rundet einen <i>Decimal</i> -Wert auf die angegebene Anzahl Nachkommastellen.
Truncate(Dec)	Gibt den Vorkomma teil des angegebenen <i>Decimal</i> -Wertes zurück.
Floor(Dec)	Rundet den <i>Decimal</i> -Wert auf die nächste kleinere ganze Zahl.
Negate(Decimal)	Multipliziert den <i>Decimal</i> -Wert mit -1.

**Tabelle 16.2:** Die wichtigsten Funktionen des *Decimal*-Typs

## Der Datentyp Char

Der Char-Datentyp speichert ein Zeichen im Unicode-Format (mehr zu diesem Thema finden Sie im ► Abschnitt »Speicherbedarf von Strings« auf Seite 464) und belegt damit 16 Bit, bzw. 2 Byte. Anders als String ist der Char-Datentyp ein Wertetyp. Die folgende Kurzübersicht klärt nähere Details:

**.NET-Datentyp:** System.Char

**Stellt dar:** ein einzelnes Zeichen

**Wertebereich:** Die Wertebereich beträgt 0–65535, damit Unicode-Zeichen dargestellt werden können.

**Typliteral:** c

**Speicherbedarf:** 2 Byte

**Delegation an den Prozessor:** ja

**CLS-Compliant:** ja

**Anmerkungen:** *Chars* werden häufig in Arrays verwendet, da ihre Verarbeitung in vielen Fällen praktischer ist als die Verarbeitung von *Strings*. Sie können Char-Arrays wie jeden anderen Datentyp mit Konstanten definieren; ein Beispiel dafür finden Sie unter dem nächsten Punkt.

Weitere Beispiele, wie Sie Char-Arrays anstelle von *Strings* beispielsweise zum Verarbeiten Buchstabe für Buchstabe verwenden, finden Sie im Abschnitt über Strings.

Auch wenn Char intern als vorzeichenloser 16-Bit-Wert und damit wie ein Short gespeichert wird, können Sie keine impliziten Konvertierungen in einen numerischen Typ vornehmen. Sie können aber, in Ergänzung zur in der Online-Hilfe beschriebenen Möglichkeit nicht nur die Funktionen AscW und ChrW zur Umwandlung von Char in einen numerischen Datentyp und umgekehrt, sondern auch die Convert-Klasse verwenden. Beispiel:

```
'Normale Deklaration und Definition
Dim locChar As Char
locChar = "1"c
Dim locInteger As Integer = Convert.ToInt32(locChar)
Console.WriteLine("Der Wert von '{0}' lautet {1}", locChar, locInteger)
```

Wenn Sie dieses Beispiel laufen lassen, sehen Sie folgende Ausgabe auf dem Bildschirm:

```
Der Wert von '1' lautet 49
```

Die Funktionen Chr und Asc können Sie ebenfalls verwenden; sie funktionieren, aber nur für Nicht-Unicode-Zeichen (ASCII 0-255). Außerdem haben sie durch etliche interne Bereichsüberprüfungen einen enormen Overhead und sind deswegen lange nicht so schnell wie AscW, ChrW (die am schnellsten sind, weil eine direkte interne Typumwandlung von Char in Integer und umgekehrt stattfindet) oder die Convert-Klasse (die den Vorteil hat, auch von Nicht-Visual-Basic-Entwicklern verstanden zu werden).

#### Deklaration und Beispielzuweisung (auch als Array):

```
'Normale Deklaration und Definition
Dim locChar As Char
locChar = "K"c

'Ein Char-Array mit Konstanten deklarieren und definieren.
Dim locCharArray() As Char = {"A"c, "B"c, "C"c}

'Ein Char-Array in einen String umwandeln.
Dim locStringAusCharArray As String = New String(locCharArray)

'Einen String in ein Char-Array umwandeln.
'Das geht natürlich auch mit einer Stringvariablen.
locCharArray = "Dies ist ein String".ToCharArray
```

## Der Datentyp String

Mit Strings speichern Sie Zeichenketten. Anders als bei Visual Basic 6 gibt es für den Umgang mit Strings einen objektorientierten Ansatz, mit dem die Programmierung der Verarbeitung von Strings viel einfacher wird. Man wird Ihre Programme viel leichter lesen können, wenn Sie dieses OOP-Konzept verwenden.

Im Laufe der vergangenen Kapitel haben Sie Strings schon an vielen Stellen kennen gelernt. Dennoch lohnt es sich, einen weiteren Blick hinter die Kulissen zu werfen, und nicht zuletzt durch die Klasse Regex (Abkürzung von »Regular Expressions« – etwa: »reguläre Ausdrücke«) gibt Ihnen das Framework eine Unterstützung in Sachen Zeichenketten an die Hand, wie sie besser eigentlich nicht mehr sein kann.

Anders als andere primitive Typen handelt es sich bei Strings um Referenztypen. Dennoch ist es nicht notwendig, eine neue String-Instanz mit dem Schlüsselwort `New` zu definieren.

Erreicht wird das durch das Eingreifen des Compilers, der ohnehin anderen Code generieren muss als bei anderen Objekten.

Die folgenden Abschnitte geben Ihnen eine Übersicht über die besonderen Eigenarten der Strings der Base Class Library. Am Ende dieses Abschnittes finden Sie im Referenzstil die Anwendung der wichtigsten String-Funktionen beschrieben.

## Strings – gestern und heute

Durch die neue Implementierung des Datentyps `String`, der ebenso wie andere Objekte durch Instanziierung seiner Klasse entsteht, gibt es seit Visual Studio 2002 und dem Framework 1.0 eine völlig neue Herangehensweise bei der Arbeit mit Zeichenketten.

Fast alle Befehle und Funktionen, die es in Visual Basic noch »allein stehend« gab, gibt es auch noch in den Framework-Versionen von Visual Basic. Allerdings sind sie nicht nur überflüssig, da es sich mit den vorhandenen Methoden und Eigenschaften des String-Objektes viel eleganter zum Ziel kommen lässt, sondern sie bremsen Programme auch unnötig aus, da sie letzten Endes selbst die String-Objektfunktionen aufrufen.

Für (fast) jede der alten String-Funktionen gibt es eine entsprechende Klassenfunktion, die Sie statt ihrer verwenden sollten. Die folgenden Abschnitte demonstrieren Ihnen den Umgang mit Strings anhand kurzer Beispiele.

## Strings deklarieren und definieren

Strings werden, wie alle primitiven Datentypen, ohne das Schlüsselwort `New` deklariert; Zuweisungen können direkt im Programm erfolgen. Eine Zeichenkette kann also beispielsweise mit der Anweisung

```
Dim locString As String
```

deklariert und sofort verwendet werden. Die Instanzbildung des String-Objektes geschieht auf IML-Ebene.

Strings werden definiert, indem Sie ihnen eine Zeichenkette zuweisen, die Sie in Anführungszeichen setzen, wie im folgenden Beispiel:

```
locString = "Miriam Sonntag"
```

Genau wie andere primitive Datentypen können Deklaration und Zuweisung in einer Anweisung erfolgen. So könnten Sie natürlich die beiden oben stehenden einzelnen Anweisungen durch die folgende ersetzen:

```
Dim locString As String = "Miriam Sonntag"
```

## Der String-Konstruktor als Ersatz von String\$

Obwohl Sie Strings wie primitive Datentypen ohne Konstruktor erstellen, haben Sie dennoch die Möglichkeit, einen Konstruktor zu verwenden. Jedoch verwenden Sie den Konstruktor nicht ausschließlich zur Neuinstanzierung eines leeren String-Objektes (der parameterlose Konstruktor ist auch gar nicht erlaubt), sondern emulieren (unter anderem) eigentlich die alte String\$-Funktion aus Visual Basic 6.0.

Mit ihrer Hilfe war es möglich, eine Zeichenfolge programmgesteuert zu wiederholen und in einem String abzuspeichern. Übrigens: Während viele der alten Visual Basic-6.0-Befehle auch noch in den Framework Versionen 1.0, 1.1 und 2.0 vorhanden sind, gibt es die String-Funktion selbst – wohl wegen der Verwendung ihres Schlüsselwortes als Typbezeichner – in Visual Basic .NET nicht mehr.

Um den String-Konstruktor als String\$-Funktionsersatz zu verwenden, verfahren Sie folgendermaßen:

```
Dim locString As String = New String("A"c, 40)
```

Sie sehen am Typliteral »c«, dass Sie im Konstruktor einen Wert vom Typ Char übergeben müssen. Damit beschränkt sich die Wiederholungsfunktion dummerweise auf ein Zeichen, was bei String\$ nicht der Fall war. Eine eigene Repeat-Funktion zu implementieren, die diese Aufgabe löst, ist aber kein wirkliches Problem:

```
Public Function Repeat(ByVal s As String, ByVal repetitions As Integer) As String
    Dim locString As String
    For count As Integer = 1 To repetitions
        locString &= s
    Next
    Return locString
End Function
```

---

**HINWEIS:** Dieses Konstrukt dient in erster Linie als Beispiel, und Sie sollten Zeichenketten auf diese Weise nur »zusammenbauen«, wenn es sich um weniger Zeichen handelt. Für größere Mengen verwenden Sie aus Performance-Gründen besser die StringBuilder-Klasse, die Sie im ► Abschnitt »Stringbuilder vs. String – wenn es auf Geschwindigkeit ankommt« ab Seite 476 beschrieben finden. Warum das so ist, klärt der ► Abschnitt »Strings sind unveränderlich« ab Seite 464.

---

Neben der Fähigkeit des Konstruktors, Strings aus einem sich wiederholenden Zeichen zu generieren, können Sie ihn ebenfalls dazu verwenden, einen String aus einem Char-Array oder einem Teil eines Char-Arrays zu erstellen, wie das folgende Beispiel zeigt:

```
Dim locCharArray() As Char = {"K"c, ".c, "c, "L"c, "ö"c, "f"c, "f"c, "e"c, "l"c, "m"c, "a"c, "n"c, "n"c}
Dim locString As String = New String(locCharArray)
Console.WriteLine(locString)
locString = New String(locCharArray, 3, 6)
Console.WriteLine(locString)
```

Wenn Sie dieses Programm laufen lassen, erscheint im Konsolenfenster folgende Ausgabe:

K. Löffelmann  
Löffel

## Einem String Zeichenketten mit Sonderzeichen zuweisen

Wenn Sie Anführungszeichen im String selbst verwenden wollen, dann bedienen Sie sich doppelter Anführungszeichen. Um die Zeichenkette

Miri sagte, "es ist erst 13.00 Uhr, ich schlafe noch ein wenig".

im Programm zu definieren, würde die Zuweisungsanweisung lauten:

```
locString = "Miriam sagte, ""ich schlafe noch ein wenig!""."
```

Möchten Sie andere Sonderzeichen in Strings einbauen, bedienen Sie sich im Basic-Sprachschatz vorhandener Konstanten. Um beispielsweise einen Absatz in einen String einzubauen, müssen Sie die ASCIIIs für *Linefeed* (Zeilenvorschub) und *Carriage Return* (Wagenrücklauf) in den String einfügen. Sie erreichen das durch die Verwendung der Konstanten, wie im folgenden Beispiel zu sehen:

```
locAndererString = "Miriam sagte ""ich schlafe noch ein wenig!"" + vbCr + vbLf +  
"Sie schlief direkt wieder ein."
```

Weniger Schreibarbeit haben Sie mit der folgenden Version, die exakt das gleiche Resultat liefert:

```
locAndererString = "Miriam sagte ""ich schlafe noch ein wenig!"" + vbNewLine +  
"Sie schlief direkt wieder ein."
```

Ihnen stehen zum Einfügen von Sonderzeichen die folgenden Konstanten in Visual Basic zur Verfügung:

Konstante	ASCII	Beschreibung
vbCrLf oder vbNewLine	13; 10	Wagenrücklaufzeichen/Zeilenvorschubzeichen
vbCr	13	Wagenrücklaufzeichen
vbLf	10	Zeilenvorschubzeichen
vbNullChar	0	Zeichen mit dem Wert 0
vbNullString	Zeichenfolge ""	Zeichenfolge mit dem Wert 0. Entspricht nicht einer Zeichenfolge mit 0-Länge (""); diese Konstante ist für den Aufruf externer Prozeduren gedacht (COM-Interop).
vbTab	9	Tabulatorzeichen
vbBack	8	Rückschrittzeichen
vbFormFeed	12	Wird in Microsoft Windows nicht verwendet.
vbVerticalTab	11	Steuerzeichen für den vertikalen Tabulator, der in Microsoft Windows aber nicht verwendet wird.

**Tabelle 16.3:** Die einfachsten Möglichkeiten, Sonderzeichen in Strings einzubauen

## Speicherbedarf von Strings

Jedes Zeichen, das in einem String gespeichert wird, belegt zwei Bytes an Arbeitsspeicher. Auch wenn Strings in Buchstabenform ausgegeben werden, so hat im Speicher selbst natürlich jedes Zeichen einen bestimmten Wert. Die Codewerte von Strings entsprechen bis 255 dem *American Standard Code for Information Interchange* – kurz ASCII –, wobei nur Werte bis 127 bei jedem verwendeten Ausgabezeichensatz einheitlich sind. Sonderzeichen spezieller Länder sind, abhängig vom verwendeten Zeichensatz, in den Bereichen 128–255 definiert, wobei auch hier in der Regel die Codes für die in europäischen Ländern verwendeten Sonderzeichen wie »öäüÖÄÜâéè« in jedem Font dieselben Codes haben (Ausnahmen bestätigen wie immer die Regel). Werte über 255 stellen Sonderzeichen dar, die beispielsweise für kyrillische, arabische oder asiatische Zeichen verwendet werden. Die Codierungskonvention, nach der ein Zeichen Werte über 255 annehmen darf, und die die Codierung einer größeren Anzahl von Zeichen erlaubt, nennt man übrigens *Unicode*. .NET-Framework-Strings speichern Zeichenketten generell im *Unicode*-Format.

## Strings sind unveränderlich

Strings sind generell Referenztypen aber dafür grundsätzlich konstant, also unveränderlich. Das bedeutet für Sie in der Praxis keine Einschränkung mit dem gewohnten Umgang von Zeichenketten, denn: Wenn es eigentlich so aussieht, als hätten Sie einen String verändert, dann haben Sie in Wahrheit einen neuen erzeugt, der die Veränderungen widerspiegelt. Wissen müssen Sie das nur bei Anwendungen, die sehr viele Stringoperationen beanspruchen. In diesem Fall sollten Sie die so genannte *StringBuilder*-Klasse verwenden, da diese zwar nicht die Flexibilität von Strings und auch nicht den Vorteil von primitiven Datentypen mit sich bringt, aber für diese Fälle deutlich leistungsfähiger ist (der ► Abschnitt »Stringbuilder vs. String – wenn es auf Geschwindigkeit ankommt« ab Seite 476 verrät mehr darüber).

Viel wichtiger ist die Auswirkung dieser Unveränderlichkeit von Strings beim Einsatz in Ihren Programmen: Obwohl Strings als Referenztypen gelten, haben sie letzten Endes das Verhalten von Wertetypen, eben dadurch, dass sie unveränderlich sind. Wenn zwei Stringvariablen auf denselben Speicherbereich verweisen und Sie den Inhalt eines Strings verändern, dann sieht es nämlich nur so aus, als würden Sie ihn verändern. In Wirklichkeit legen Sie ja ein komplett neues String-Objekt im Speicher ab und lassen die vorhandene Variable darauf verweisen. Damit kommen Sie nie in die Situation, die Sie von Referenztypen kennen: Das Verändern eines Objektinhaltes durch die eine Objektvariable führt bei Strings nie dazu, dass eine andere Objektvariable, die auf den gleichen Speicherbereich zeigte, den Stringinhalt verändert wiedergibt, denn Strings werden ja nicht verändert. Diese Tatsache erklärt, dass Strings zwar Referenztypen sind, sich aber wie Wertetypen »anfühlen«.

Mehr Informationen dazu erhalten Sie auch im nächsten Abschnitt und den programmtechnischen Beweis dafür im ► Abschnitt »Trimmen von Strings« auf Seite 469.

## Speicheroptimierung von Strings durch das Framework

Für die Speicherung von Strings gibt es einen so genannten *internen Pool*, der dazu verwendet wird, Redundanzen bei der Zeichenkettenspeicherung zu vermeiden. Wenn Sie innerhalb Ihres Programms zwei Strings mit der gleichen Konstante definieren, erkennt der Visual Basic-Compiler das und legt den String im Speicher nur ein einziges Mal ab, lässt beide Objektvariablen aber auf denselben Speicherbereich zeigen, wie das folgende Beispiel beweist:

```

Dim locString As String
Dim locAndererString As String

locString = "Miriam" & " Sonntag"
locAndererString = "Miriam Sonntag"
Console.WriteLine(locString Is locAndererString)

```

Wenn Sie dieses Programm starten, gibt es True aus – beide Strings verweisen also auf den gleichen Speicherbereich.

Diese Tatsache gilt aber nur so lange, wie der Compiler die Gleichheit der Strings erkennen kann, und dafür müssen die Konstanten in der gleichen Zeile zusammengesetzt werden. Schon bei der Veränderung in die folgende Version kann der Compiler die Gleichheit der Strings nicht mehr erkennen, und das Ergebnis wird False:

```

Dim locString As String
Dim locAndererString As String

locString = "Miriam"
locString &= " Sonntag"
locAndererString = "Miriam Sonntag"
Console.WriteLine(locString Is locAndererString)

```

Es liegt auf der Hand, dass dieses Verhalten zur Laufzeit zu viel Zeit kosten würde, um es sinnvoll anzuwenden. Bei sehr hohem String-Aufkommen würde die BCL zu viel Zeit nach der Suche bereits vorhandener Strings verschwenden. Allerdings haben Sie die Möglichkeit, einen String, den Sie zur Laufzeit erstellen, gezielt dem Pool hinzuzufügen. Gleichen sich mehrere Strings, die Sie dem internen Pool hinzufügen, werden die Speicherbereiche wieder nicht-redundant zugewiesen – mehrere, gleich lautende Strings teilen sich dann den Speicher. Sinnvoll ist das natürlich nur dann, wenn vorzusehen ist, dass es viele gleich lautende Strings innerhalb eines Programms geben wird. Ein Beispiel zeigt, wie das Hinzufügen eines Strings mit der statischen Funktion Intern zum internen Pool explizit vorgenommen wird:

```

Dim locString As String = New String(locCharArray)
Dim locAndererString As String

locString = "Miriam"
locString &= " Sonntag"
locString = String.Intern(locString)
locAndererString = String.Intern("Miriam Sonntag")
Console.WriteLine(locString Is locAndererString)

```

Wenn Sie dieses Programm starten, lautet die Ausgabe wieder True.

## Ermitteln der String-Länge

**Visual Basic 6 kompatibler Befehl:** Len

**Visual Basic .NET:** strVar.Length

**Anmerkung:** Mit diesem Befehl ermitteln Sie die Länge eines Strings in Zeichen (nicht in Bytes!).

**Beispiel:** Das folgende Beispiel liest eine Zeichenkette von der Tastatur ein und gibt die Zeichen in umgekehrter Reihenfolge aus:

```

Dim locString As String
Console.Write("Geben Sie einen Text ein: ")
locString = Console.ReadLine()
For count As Integer = locString.Length - 1 To 0 Step -1
    Console.Write(locString.Substring(count, 1))
Next

```

Das gleiche Beispiel mit den VB6-Kompatibilitätsbefehlen finden Sie im nächsten Abschnitt.

## Ermitteln von Teilen eines Strings oder eines einzelnen Zeichens

**Visual Basic 6 kompatible(r) Befehl(e):** Left, Right, Mid

**Visual Basic .NET:** strVar.SubString

**Anmerkung:** Mit diesem Befehl können Sie einen bestimmten Teil eines Strings als String zurückgeben lassen.<sup>7</sup>

**Beispiel:** Das folgende Beispiel liest eine Zeichenkette von der Tastatur ein und gibt die Zeichen in umgekehrter Reihenfolge aus. Das gleiche Beispiel finden Sie im vorherigen Abschnitt mit den Funktionen des String-Objektes.

```

Dim locString As String
Console.Write("Geben Sie einen Text ein: ")
locString = Console.ReadLine()
For count As Integer = Len(locString) To 1 Step -1
    Console.Write(Mid(locString, count, 1))
Next

```

---

**HINWEIS:** Bitte achten Sie darauf, dass die VB6-Kompatibilitätsfunktionen die Zeichenzählung eines Strings durch Left, Right oder Mid bei 1 beginnen lassen, die Funktion SubString jedoch bei 0 beginnen lässt.

---

## Angleichen von String-Längen

**Visual Basic 6 kompatible(r) Befehl(e):** RSet, LSet

**Visual Basic .NET:** strVar.PadLeft; strVar.PadRight

**Anmerkung:** Mit diesen Befehlen können Sie die Länge eines Strings auf eine bestimmte Zeichenanzahl erweitern; der String wird dabei entweder vorne oder am Ende mit Leerzeichen aufgefüllt.

**Beispiel:** Das folgende Beispiel demonstriert den Umgang mit der PadLeft- und der PadRight-Methode.

```

Dim locString As String = "Dieser String ist so lang"
Dim locString2 As String = "Dieser nicht"
Dim locString3 As String = "Dieser"
Len(locString)

```

---

<sup>7</sup> Warum das gute alte Left bzw. Right als jeweilige Methoden der Klasse String weggelassen wurde, weiß allein der Programmierer. Vielleicht weiß es der auch nicht, sondern hat es einfach vergessen oder kennt nur C und kann sich nicht vorstellen, dass die Welt so einfach wie in Basic sein kann.

```

locString2 = locString2.PadLeft(locString.Length)
locString3 = locString3.PadRight(locString.Length)

Console.WriteLine(locString + ":")
Console.WriteLine(locString2 + ":")
Console.WriteLine(locString3 + ":")

```

Wenn Sie dieses Programm laufen lassen, generiert es die folgende Ausgabe:

```

Dieser String ist so lang:
Dieser nicht:
Dieser      :

```

## Suchen und Ersetzen

**Visual Basic 6 kompatible(r) Befehl(e):** Instr; InstrRev; Replace

**Visual Basic .NET:** strVar.IndexOf; strVar.IndexOfAny; strVar.Replace; strVar.Remove

**Anmerkung:** Mit dem VB6-kompatiblen Befehl Instr können Sie nach dem Vorkommen eines Zeichens oder einer Zeichenfolge in einem String suchen. InstrRev macht das Gleiche, beginnt die Suche aber von hinten. Replace erlaubt Ihnen, eine Zeichenfolge im String durch eine andere zu ersetzen.

Mit der String-Funktion IndexOf suchen Sie nach dem Vorkommen eines Zeichens oder einer Zeichenfolge im aktuellen String. IndexOfAny erlaubt Ihnen darüber hinaus, das Vorhandensein verschiedener Zeichen, die in einem Char-Array übergeben werden, im String zu finden. Replace ersetzt einzelne Zeichen oder Zeichenfolgen durch andere im aktuellen String und mit Remove haben Sie die Möglichkeit, eine bestimmte Zeichenfolge ganz aus dem String zu entfernen.

**Beispiel:** Das folgende Beispiel demonstriert den Umgang mit den Suchen- und Ersetzen-Methoden des String-Objektes:

---

**BEGLEITDATEIEN:** Sie finden das Beispielprojekt im Verzeichnis *.\VB 2005 - Entwicklerbuch\E - Datentypen\Kap16\Strings - Suchen und Ersetzen*.

---

```

Imports System.Globalization

Module Strings
    Sub Main()
        Dim locString As String =
            "Weisheiten:" + vbCrLf +
            "* Wenn man 8 Jahre, 7 Monate und 6 Tage schreien würde," + vbCrLf +
            " hätte man genug Energie produziert, um eine Tasse Kaffee heiß zu machen." + vbCrLf +
            "* Wenn man seinen Kopf gegen die Wand schlägt, verbraucht man 150 Kalorien." + vbCrLf +
            "* Elefanten sind die einzigen Tiere, die nicht springen können." + vbCrLf +
            "* Eine Kakerlake kann 9 Tage ohne Kopf überleben, bevor sie verhungert." + vbCrLf +
            "* Gold und andere Metalle entstehen ausschließlich in" + vbCrLf +
            " Supernovae (Sternenexplosionen)." + vbCrLf +
            "* Der Mond besteht aus den Trümmern der Kollision eines Mars großen" + vbCrLf +
            " Planeten mit der Erde." + vbCrLf +
            "* New York wird ""Big Apple"" genannt, weil ""Big Apple"" in der Sprache" + vbCrLf +
            " der Jazz-Musiker ""das große Los ziehen"" bedeutete. In New York Karriere" + vbCrLf +
            " zu machen war ihr großes Los." + vbCrLf +

```

```

    /* Der Ausdruck ""08/15"" für etwas Unoriginelles war ursprünglich " + vbNewLine +
    " die Typenbezeichnung für das Maschinengewehr LMG 08/15;" + vbNewLine +
    " er wurde Metapher für geistlosen, militärischen Drill." + vbNewLine +
    /* ""Durch die Lappen gehen"" ist ein Begriff aus der Jagd:" + vbNewLine +
    " Hirsche ließen nicht durch eine aus Lappen bestehende," + vbNewLine +
    " flatternde Umzäunung - aus Angst. Außer manchmal."



'Zahlenkombi durch Buchstaben ersetzen
locString = locString.Replace("08/15", "Null-Acht-Fünfzehn")

'Satzzeichen zählen
Dim locPosition, locCount As Integer

Do
    locPosition = locString.IndexOfAny(New Char() {"c, ,c, :"c, "?"c}, locPosition)
    If locPosition = -1 Then
        Exit Do
    Else
        locCount += 1
    End If
    locPosition += 1
Loop

Console.WriteLine("Der folgende Text...")
Console.WriteLine(New String("=c, 79))
Console.WriteLine(locString)
Console.WriteLine(New String("=c, 79))
Console.WriteLine("...verfügt über {0} Satzzeichen.", locCount)
Console.WriteLine()
Console.WriteLine("Und sieht nach dem Ersetzen von 'Big Apple' durch 'Großer Apfel' so aus:")
Console.WriteLine(New String("=c, 79))

'Noch eine Ersetzung
locString = locString.Replace("Big Apple", "Großer Apfel")
Console.WriteLine(locString)
Console.ReadLine()

End Sub

End Module

```

Dieses Beispiel gibt folgendes auf dem Bildschirm aus:

```

Der folgende Text...
=====
Weisheiten:
* Wenn man 8 Jahre, 7 Monate und 6 Tage schreien würde,
  hätte man genug Energie produziert, um eine Tasse Kaffee heiß zu machen.
* Wenn man seinen Kopf gegen die Wand schlägt, verbraucht man 150 Kalorien.
* Elefanten sind die einzigen Tiere, die nicht springen können.
* Eine Kakerlake kann 9 Tage ohne Kopf überleben, bevor sie verhungert.
* Gold und andere Metalle entstehen ausschließlich in
  Supernovae (Sternenexplosionen).
* Der Mond besteht aus den Trümmern der Kollision eines Mars großen

```

Planeten mit der Erde.

- \* New York wird "Big Apple" genannt, weil "Big Apple" in der Sprache der Jazz-Musiker "das große Los ziehen" bedeutete. In New York Karriere zu machen war ihr großes Los.
  - \* Der Ausdruck "Null-Acht-Fünfzehn" für etwas Unoriginales war ursprünglich die Typenbezeichnung für das Maschinengewehr LMG Null-Acht-Fünfzehn; er wurde Metapher für geistlosen, militärischen Drill.
  - \* "Durch die Lappen gehen" ist ein Begriff aus der Jagd: Hirsche liefen nicht durch eine aus Lappen bestehende, flatternde Umzäunung - aus Angst. Außer manchmal.
- 

...verfügt über 23 Satzzeichen.

Und sieht nach dem Ersetzen von 'Big Apple' durch 'Großer Apfel' so aus:

Weisheiten:

- \* Wenn man 8 Jahre, 7 Monate und 6 Tage schreien würde, hätte man genug Energie produziert, um eine Tasse Kaffee heiß zu machen.
- \* Wenn man seinen Kopf gegen die Wand schlägt, verbraucht man 150 Kalorien.
- \* Elefanten sind die einzigen Tiere, die nicht springen können.
- \* Eine Kakerlake kann 9 Tage ohne Kopf überleben, bevor sie verhungert.
- \* Gold und andere Metalle entstehen ausschließlich in Supernovae (Sternenexplosionen).
- \* Der Mond besteht aus den Trümmern der Kollision eines Mars großen Planeten mit der Erde.
- \* New York wird "Großer Apfel" genannt, weil "Großer Apfel" in der Sprache der Jazz-Musiker "das große Los ziehen" bedeutete. In New York Karriere zu machen war ihr großes Los.
- \* Der Ausdruck "Null-Acht-Fünfzehn" für etwas Unoriginales war ursprünglich die Typenbezeichnung für das Maschinengewehr LMG Null-Acht-Fünfzehn; er wurde Metapher für geistlosen, militärischen Drill.
- \* "Durch die Lappen gehen" ist ein Begriff aus der Jagd: Hirsche liefen nicht durch eine aus Lappen bestehende, flatternde Umzäunung - aus Angst. Außer manchmal.

---

**TIPP:** Das Beispiel zu ► »Algorithmisches Auflösen eines Strings in Teile« auf Seite 470 enthält eine weitere, selbst geschriebene Funktion, die ich ReplaceEx genannt habe, und mit der Sie nach mehreren Zeichen suchen und, falls eines von ihnen dem durchsuchten Zeichen entsprach, es durch ein angebares Zeichen ersetzen können.

---

## Trimmen von Strings

**Visual Basic 6 kompatible(r) Befehl(e):** Trim; RTrim; LTrim

**Visual Basic .NET:** strVar.Trim; strVar.TrimEnd; strVarTrimStart

**Anmerkung:** Mit diesen Befehlen können Sie überflüssige Zeichen am Anfang, am Ende oder an beiden Seiten eines Strings entfernen. Die Objektmethoden des Strings sind dabei den Kompatibilitätsfunktionen vorzuziehen, da Sie bei ersteren auch bestimmen können, welche Zeichen getrimmt werden sollen, wie das unten stehende Beispiel zeigt. Die VB6-Kompatibilitätsfunktionen beschränken ihre Trimmfähigkeit auf Leerzeichen.

**Beispiel:** Das folgende Beispiel generiert ein String-Array, dessen einzelne Elemente am Anfang und am Ende unerwünschte Zeichen (nicht nur Leerzeichen) haben, die durch die Trim-Funktion entfernt werden.

---

**HINWEIS:** Dieses Beispiel zeigt dabei ebenfalls, dass Strings, obwohl sie als Referenztypen gelten, sich durch die Tatsache, dass sie an sich unveränderlich sind, anders verhalten als herkömmliche Objekte. Wenn Sie ein Objekt zwei Objektvariablen zuweisen und den Inhalt eines Objektes über eine Variable verändern, dann spiegelt die zweite Objektvariable ebenfalls den geänderten Inhalt des Objektes wider. Obwohl Strings Referenzvariablen sind, zeigen sie dennoch dieses Verhalten nicht, da Sie den String-Inhalt nicht verändern können. Strings werden immer neu erstellt, nie verändert (lesen Sie Weiteres zu diesem Thema im ► Abschnitt »Strings sind unveränderlich« auf Seite 464).

---

```
Dim locStringArray() As String = {  
    " - Hier geht der eigentliche Text los!", _  
    "Dieser Text endet mit komischen Zeichen! .-", _  
    " - Hier sind beide Seiten problematisch - "}  
  
For Each locString As String In locStringArray  
    locString = locString.Trim(New Char() {" ", ".", "-", "-c"})  
    Console.WriteLine("Sauber und ordentlich: " + locString)  
Next  
  
'Wichtig: String ist zwar ein Referenztyp, am Array hat sich dennoch nichts verändert.  
'Das liegt daran, dass Strings nicht direkt verändert, sondern immer neu – dabei verändert – angelegt  
werden.  
For Each locString As String In locStringArray  
    Console.WriteLine("Immer noch unordentlich: " + locString)  
Next
```

Wenn Sie dieses Programm laufen lassen, generiert es die folgende Ausgabe:

Sauber und ordentlich: Hier geht der eigentliche Text los!  
Sauber und ordentlich: Dieser Text endet mit komischen Zeichen!  
Sauber und ordentlich: Hier sind beide Seiten problematisch  
Immer noch unordentlich: - Hier geht der eigentliche Text los!  
Immer noch unordentlich: Dieser Text endet mit komischen Zeichen! .-  
Immer noch unordentlich: - Hier sind beide Seiten problematisch -

## Algorithmisches Auflösen eines Strings in Teile

**Visual Basic 6 kompatible(r) Befehl(e):** Split

**Visual Basic .NET:** strVar.Split

**Anmerkung:** Die .NET-Split-Methode des String-Objektes ist der Kompatibilitätsfunktion insofern überlegen, als sie erlaubt, mehrere Separatorzeichen in einem Char-Array anzugeben. Damit werden Ihre Programme ungleich flexibler bei der Analyse und dem Neuaufbau von Texten.

**Beispiel:** Das folgende Beispiel zerlegt die einzelnen, durch verschiedene Separatorzeichen getrennten Begriffe oder Abschnitte eines Strings in Teilstrings, die anschließend als Elemente eines String-Arrays vorliegen und durch weitere Funktionen noch besser aufbereitet werden.

---

**BEGLEITDATEIEN:** Sie finden dieses Beispiel im Verzeichnis .\VB 2005 - Entwicklerbuch\E - Datentypen\Kap16\String-Split\String - Split.sln

---

```
Module Strings
    Sub Main()
        Dim locString As String =
            "Einzelne, Elemente; durch, die , verschiedensten - Zeichen , getrennt."
        Console.WriteLine("Aus der Zeile:")
        Console.WriteLine(locString)
        Console.WriteLine()
        Console.WriteLine("Wird ein String-Array mit folgenden Elementen:")
        Dim locStringArray As String()
        locStringArray = locString.Split(New Char() {"c", ";", "-", ".})
        For Each locStr As String In locStringArray
            Console.WriteLine(ReplaceEx(locStr, New Char() {"c", ";", "-", ".}, _
                Convert.ToChar(vbNullChar)).Trim)
        Next
        Console.ReadLine()
    End Sub

    Public Function ReplaceEx(ByVal str As String, ByVal SearchChars As Char(), _
        ByVal ReplaceChar As Char) As String
        Dim locPos As Integer
        Do
            locPos = str.IndexOfAny(SearchChars)
            If locPos = -1 Then Exit Do
            If AscW(ReplaceChar) = 0 Then
                str = str.Remove(locPos, 1)
            Else
                str = str.Remove(locPos, 1).Insert(locPos, ReplaceChar.ToString)
            End If
        Loop
        Return str
    End Function
End Module
```

Wenn Sie dieses Programm laufen lassen, generiert es die folgende Ausgabe:

Aus der Zeile:

Einzelne, Elemente; durch, die , verschiedensten - Zeichen , getrennt.

Wird ein String-Array mit folgenden Elementen:

Einzelne  
Elemente  
durch  
die  
verschiedensten  
Zeichen  
getrennt

## Ein String-Schmankerl zum Schluss

Ich muss gestehen, ich habe ein Faible für bestimmte Fernsehserien. Besonders angetan haben es mir *Emergency Room* und *Star Trek Enterprise*<sup>8</sup>. Mein Faible geht so weit, dass ich – natürlich nur für den eigenen, privaten Bedarf – nicht nur keine Folge verpassen will, sondern sie mir auch gerne noch mal anschau. Aus dem Grund habe ich einen DVD-Rekorder, mit dem ich die Folgen aufzeichne und anschließend auf meinem Computer ins DivX-Format umrechne – natürlich nur für mich privat – damit ich sie mir auch auf meinem Notebook anschauen kann, wenn ich auf Reisen bin. Manchmal jedoch vergesse ich, eine Folge aufzuzeichnen, und dann kommt mein guter Freund Christian ins Spiel. Auch er hat ein Faible für Enterprise und ER<sup>9</sup> und springt beim Aufnehmen manchmal für mich ein, wenn ich einmal vergessen habe, den Rekorder zu programmieren. Nur leider benennt er die für ausschließlich seinen eigenen Bedarf ins DivX-Format umgewandelten Videodateien nach einem anderen System. Er verwendet anstelle von Leerzeichen häufig den Unterstrich. Außerdem kennzeichnet er Staffelnummer und Episode nicht – so wie ich – im Format »sxee«, sondern setzt noch jeweils ein »S« und ein »E« davor. Aber er macht das auch nicht immer. So passiert es immer wieder, dass ich eine Staffel komplett auf der Platte gespeichert habe, aber die Dateinamen irgendwie unschön unregelmäßig benannt sind, etwa so:

```
F:\Video\ER\ER - 9x01 - chaos theory.mpg  
F:\Video\ER\ER - 9x02 - dead again.mpg  
F:\Video\ER\ER - 9x03 - insurrection.mpg  
F:\Video\ER\ER - 9x04 - walk like a man.mpg  
F:\Video\ER\ER - 9x05 - a hopeless wound.mpg  
F:\Video\ER\ER - _S09E11 - _A_Little_Help_from_My_Friends.von_Christian.mpg  
F:\Video\ER\ER - _S09E14 - _No_strings_Attached.von_Christian.mpg  
F:\Video\ER\ER - _S09E15 - _A_Boy_Falling_out_of_the_Sky.von_Christian.mpg  
F:\Video\ER\ER - _S09E19 - _Things_Change.von_Christian.mpg  
F:\Video\ER\ER - _S09E20 - _Foreign.Affairs.von_Christian.mpg
```

Sie sehen an der Liste, dass das manuelle Umbenennen dieser Dateien eine Ewigkeit benötigen würde, um die Dateinamen in ein einheitliches Format zu bringen. Aus diesem Grund habe ich ein kleines Programm geschrieben, das das Umbenennen enorm erleichtert. Mit ihm können Sie nicht nur die Nummerierung von Episodendateien in mein bevorzugtes<sup>10</sup> Format durchführen lassen – Sie können sogar in den Dateinamen ein Suchen und Ersetzen ausführen, um – in meinem Fall – das nervige »von\_Christian«<sup>11</sup> zu entfernen.

Sie können dieses Tool natürlich auch für andere Aufgaben einsetzen. Es hat mir bei diesem Buch beispielsweise mehrfach geholfen, die Bilder, die Kapitelnummerpräfixe trugen, auf neue Kapitelnummern anzupassen – Anwendungen für das Tool gibt es viele.

---

**BEGLEITDATEIEN:** Sie finden das Programm im Verzeichnis .\VB 2005 - Entwicklerbuch\E - Datentypen\Kap16\Dateinamenangleicher.

---

<sup>8</sup> Und ich verstehe nicht, wieso Star Trek Enterprise eingestellt werden musste...

<sup>9</sup> Abby ist schwanger von Luca Kovac – wer hätte das gedacht?

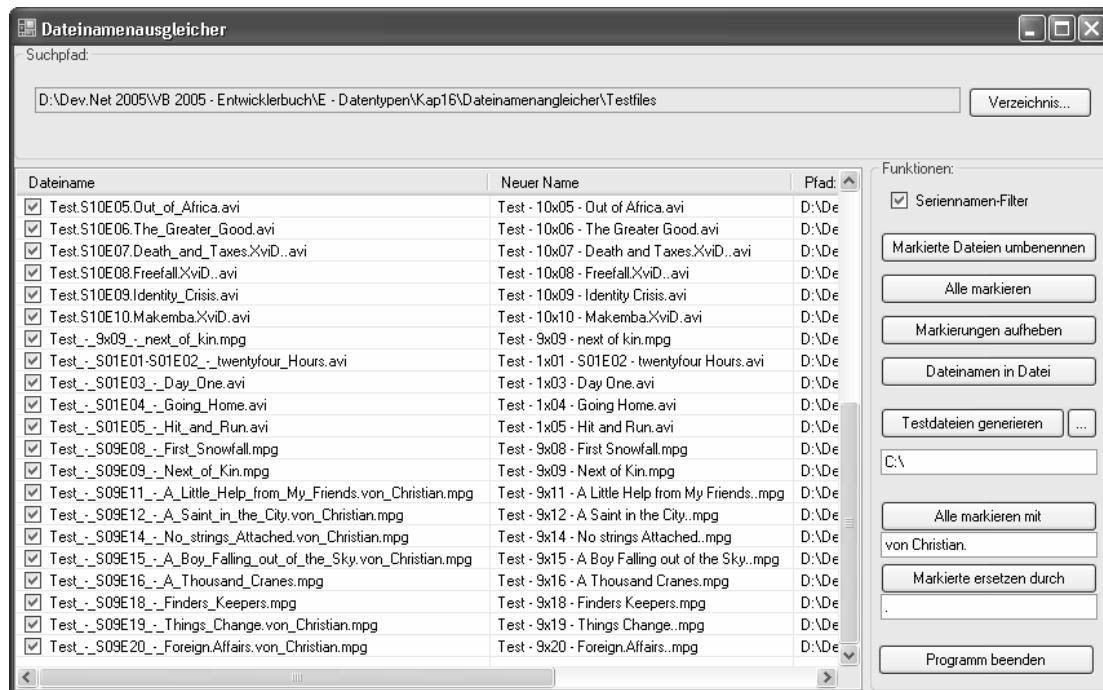
<sup>10</sup> Mit ein paar Handgriffen können Sie natürlich das Programm so ändern, dass Ihr bevorzugtes Format dargestellt wird.

<sup>11</sup> No offense, Kricke!

Wenn Sie das Programm starten, sehen Sie einen Dialog, der in seiner Größe beliebig veränderbar ist, etwa wie in Abbildung 16.2 zu sehen.

In der linken Spalte sehen Sie die Originaldateinamen. In der rechten Spalte sehen Sie die durch Ihre Eingriffe veränderten. Wichtig: Das Programm benennt die Dateinamen erst dann physisch auf der Festplatte um, wenn Sie die Schaltfläche *Markierte Dateien umbenennen* anklicken – Sie können also beruhigt mit den Dateinamen experimentieren, ohne Angst haben zu müssen, dass Sie sich Dateinamen »zerschießen«.

Sie finden in den Unterverzeichnissen *Testfiles* und *Backup Testfiles* Dateien (natürlich leere Testdateien), mit denen Sie experimentieren können. Die Liste dieser Dateien ist übrigens auch in der Abbildung zu sehen. Um das angezeigte Verzeichnis zu wechseln, klicken Sie auf die Schaltfläche *Verzeichnis*. Die Dateinamenliste wird dann eingelesen und dargestellt.



**Abbildung 16.2:** Mit dem Dateinamenausgleicher können Sie Dateinamen algorithmisch umbenennen; auch das Suchen und Ersetzen in Dateinamen ist damit möglich

Möchten Sie den Algorithmus zum Ausgleichen des Dateinamens für Seriennamen nicht anwenden, entfernen Sie einfach das Häkchen vor *Seriennamen-Filter*.

Die anderen Funktionen lassen Sie sich am besten durch das Programm erklären: Hinter jeder Schaltfläche verbirgt sich ein Tooltip, der Sie über die jeweilige Funktion aufklärt.

Erst, wenn Sie alle Veränderungen vorgenommen haben, klicken Sie auf die Schaltfläche *Markierte Dateien umbenennen*, um Ihre Änderungen zu übernehmen.

Und jetzt, nachdem Sie wissen, wie Sie das Programm anwenden, werden Sie eine ungefähre Vorstellung davon haben, dass String-Funktionen bei seiner Programmierung nicht zu kurz kamen. Allerdings werde ich mich in diesem Zusammenhang – aus Platzgründen – bei der Erklärung des Programms auf die String-relevanten Funktionen beschränken.

Nur soviel zur generellen Funktion des Programms: Es verwendet keine eigenen Klassen zur Speicherung der Daten, sondern erweitert die vorhandenen Klassen `ListView` für die Darstellung der Dateien, und `ListViewItem` für einen einzelnen darzustellenden Eintrag innerhalb der `ListView`. Durch diese Vorgehensweise wird es extrem kompakt.

Die für die String-Verarbeitung interessanten Punkte befinden sich im Seriennamen-Algorithmus, der über die Eigenschaft `NewFilename` der aus `ListViewItem` abgeleiteten Klasse `FilenameEnumeratorItem` realisiert wird:

```

Public Overridable ReadOnly Property NewFilename() As FileInfo
    Get
        Dim locFilename As String = myFilename.Name
        Dim locParts As New ArrayList
        Dim blnCharTypesChanged As Boolean
        Dim locChar, locPrevChar As Char
        Dim locCurrentPart As String
        Dim locNumberStartPart, locNumberEndPart As Integer
        Dim locProhibitFurtherCharTypeChanges As Boolean
        Dim locPräfix, locNumPart, locPostfix As String
        If Not myEpisodeNameFilter Then
            Return myFilename
        End If

        'Finden des Nummern-Parts und Ersetzen aller Unterstriche
        'durch Leerzeichen.
        For count As Integer = 0 To locFilename.Length - 1
            locChar = locFilename.Chars(count)
            If locChar = "_"c Then locChar = " "c

            'Den Nummernpart des Dateinamens suchen.
            If Not locProhibitFurtherCharTypeChanges Then
                If locChar.IsDigit(locChar) And Not blnCharTypesChanged Then
                    blnCharTypesChanged = True
                    'Falls "S" oder "s" davor stand, Buchstaben mit einbeziehen.
                    If locPrevChar = "S"c Or locPrevChar = "s"c Then
                        locNumberStartPart = count - 1
                    Else
                        locNumberStartPart = count
                    End If
                End If

                If Not locProhibitFurtherCharTypeChanges Then
                    'Wenn Nummernpart schon vorbei, und wieder ein Buchstabe...
                    If Char.IsLetter(locChar) And blnCharTypesChanged Then
                        If count < locFilename.Length - 2 Then
                            Dim locNextChar As Char = locFilename.Chars(count + 1)
                            '...aber nur wenn der Buchstabe kein Episodenkennzeichner ist...
                        End If
                    End If
                End If
            End If
        Next
    End Get

```

```

        If Not ((Char.IsLetter(locChar) And Char.IsDigit(locNextChar)) And _
            (locChar = "E"c Or locChar = "e"c Or locChar = "x" Or locChar = "X")) Then
            '...ist der Nummernpart vorbei, und es folgt wieder Text.
            locNumberEndPart = count
            locProhibitFurtherCharTypeChanges = True
        End If
    End If
End If
locCurrentPart += locChar.ToString
locPrevChar = locChar
Next

'Sonderfall: Dateiname endet mit Nummer.
If locNumberEndPart = 0 Then
    locNumberEndPart = locFilenameLength
End If

'Dateinamen auseinander bauen.
locPräfix = locCurrentPart.Substring(0, locNumberStartPart)
locNumPart = locCurrentPart.Substring(locNumberStartPart, _
    locNumberEndPart - locNumberStartPart)
locPostfix = locCurrentPart.Substring(locNumberEndPart)
'Alle denkbaren "Umbauten" im Dateinamen durchführen.
locPräfix = locPräfix.Replace(".", "c, " "c")
locPräfix = locPräfix.Trim(New Char() {" "c, "-c"})
locNumPart = locNumPart.Replace(".", "c, ")
locNumPart = locNumPart.Replace("S"c, "")
locNumPart = locNumPart.Replace("s"c, "")
locNumPart = locNumPart.Replace("E"c, "x"c)
locNumPart = locNumPart.Replace("e"c, "x"c)
locNumPart = locNumPart.Replace("X"c, "x"c)
locNumPart = locNumPart.Trim(New Char() {" "c, "-c"})
locPostfix = locPostfix.Trim(New Char() {" "c, "-c"})

'Und neuen Dateinamen zurückliefern.
Return New FileInfo(Filename.DirectoryName + "\\" + locPräfix + " - " + locNumPart + _
    " - " + locPostfix)
End Get
End Property

```

## Iterieren durch einen String

Sie sehen, dass diese Routine für das Iterieren durch die einzelnen Buchstaben eine weitere Variation verwendet: Mit der `Chars`-Eigenschaft eines `String`-Objekts greifen Sie auf ein `Char`-Array zu, dass die einzelnen Zeichen des Strings repräsentiert. Da das `String`-Objekt auch die Funktion `GetEnumerator` anbietet, gäbe es auch die folgende Möglichkeit, durch einen String zu iterieren:

```

For Each locChar As Char In "Dies ist ein String"
    'Tu irgendwas.
Next

```

Durch die Mächtigkeit des String-Objektes sind die Funktionen Suchen und Ersetzen des Programms erschreckend einfach zu realisieren, wie der folgende Codeausschnitt eindrucksvoll zeigt:

```
Private Sub btnCheckFound_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnCheckFound.Click
    For Each locFEI As FilenameEnumeratorItem In fneFiles.Items
        Dim locFilename As String = locFEI.Filename.Name
        If locFilename.IndexOf(txtSearch.Text) > -1 Then
            locFEI.Checked = True
        Else
            locFEI.Checked = False
        End If
    Next
End Sub

Private Sub btnReplaceChecked_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnReplaceChecked.Click
    For Each locFEI As FilenameEnumeratorItem In fneFiles.Items
        Dim locFilename As String = locFEI.SubItems(1).Text
        If locFEI.Checked Then
            If locFilename.IndexOf(txtSearch.Text) > -1 Then
                locFilename = locFilename.Replace(txtSearch.Text, txtReplace.Text)
                locFEI.SubItems(1).Text = locFilename
            End If
        End If
    Next
End Sub
```

## StringBuilder vs. String – wenn es auf Geschwindigkeit ankommt

Sie haben im Laufe des String-Abschnittes gesehen, dass Ihnen .NET mit dem Datentyp String ein mächtiges Werkzeug für die Bearbeitung von Zeichenketten in die Hände legt. Wenn Sie den Abschnitt über die Speicherverwaltung gelesen haben, dann werden Sie aber auch bemerkt haben, dass es um die Geschwindigkeit bei der Verarbeitung von Strings in bestimmten Szenarien nicht so gut bestellt ist. Der Grund dafür ist einfach: Strings sind unveränderlich. Wenn Sie mit Algorithmen hantieren, die Strings im Laufe ihrer Entstehungsgeschichte zeichenweise zusammensetzen, dann wird für jedes Zeichen, das zum String hinzukommt, ein komplett neuer String erstellt. Und das kostet Zeit.

Eine Alternative dazu bildet die `StringBuilder`-Klasse. Sie hat bei weitem nicht die Funktionsvielfalt eines Strings, aber sie hat einen entscheidenden Vorteil: Sie wird dynamisch verwaltet und ist damit ungleich schneller. Das heißt für Sie: Wann immer es darum geht, Strings zusammenzusetzen (indem Sie Zeichen anhängen, einfügen oder löschen), sollten Sie ein `StringBuilder`-Objekt einsetzen – gerade wenn große Datenmengen im Spiel sind.

Der Umgang mit einem `StringBuilder`-Objekt ist denkbar einfach. Um auf das Objekt zurückgreifen zu können, benötigen Sie Zugriff auf den Namensbereich `Text`, den Sie mit der Anweisung

```
Imports System.Text
```

in Ihre Klassen- oder Moduldatei einbinden.

Sie deklarieren eine Variable einfach vom Typ `StringBuilder` und definieren sie mit einer der folgenden Anweisungen:

```
'Deklaration ohne Parameter:  
Dim locSB As New StringBuilder  
'Deklaration mit Kapazitätsreservierung  
locSB = New StringBuilder(1000)  
'Deklaration aus einem vorhandenen String  
locSB = New StringBuilder("Aus einem neuen String entstanden")  
'Deklaration aus String mit der Angabe einer zu reservierenden Kapazität  
locSB = New StringBuilder("Aus String entstanden mit Kapazität für weitere", 1000)
```

Sie können, falls Sie das möchten, eine Ausgangskapazität bei der Definition eines `StringBuilder`-Objektes angeben. Damit wird der Platz, den Ihr `StringBuilder`-Objekt voraussichtlich benötigen wird, direkt reserviert – zusätzlicher Speicher muss zur Laufzeit nicht angefordert werden, und das spart zusätzlich Zeit.

Um den String um Zeichen zu erweitern, verwenden Sie die `Append`-Methode. Mit `Insert` können Sie weitere Zeichen in ein `StringBuilder`-Objekt einfügen. `Replace` erlaubt Ihnen das Ersetzen einer Zeichenkette durch eine andere. Und mit `Remove` haben Sie die Möglichkeit, eine bestimmbarer Anzahl von Zeichen ab einer bestimmten Zeichenposition zu entfernen.

Beispiel:

```
locSB.Append(" - und das wird an den String angefügt")  
locSB.Insert(20, ">>das kommt irgendwo in die Mitte<<")  
locSB.Replace("String", "StringBuilder")  
locSB.Remove(0, 4)
```

Wenn der String komplett zusammengesetzt wurde, können Sie ihn mit der `Tostring`-Funktion in einen »echten« String umwandeln:

```
'StringBuilder hat den String fertig zusammengesetzt,  
'in String umwandeln.  
Dim locString As String = locSB.ToString  
Console.WriteLine(locString)
```

Führten Sie dieses Beispiel aus, würde es folgenden Text im Konsolenfenster anzeigen:

```
StringBuilder entstande>>das kommt irgendwo in die Mitte<<n mit Kapazität für we  
itere - und das wird an den StringBuilder angefügt
```

## Performance-Vergleich: String gegen StringBuilder

---

**BEGLEITDATEIEN:** Sie finden im Verzeichnis `\VB 2005 - Entwicklerbuch\E - Datentypen\Kap16\StringVsStringBuilder\` ein Projekt, mit der die Performance-Unterschiede zwischen der Verarbeitungsgeschwindigkeit von `String`- und `StringBuilder`-Objekten deutlich werden.

---

Das Programm kreiert eine bestimmbarer Anzahl von `String`-Elementen, die jeweils aus einer ebenfalls bestimmbarer Menge aus zufälligen Zeichen bestehen. Wenn Sie das Programm starten, bestimmen Sie diese Parameter:

```
Geben Sie die String-Länge eines Elementes ein: 100  
Geben die Anzahl der zu erzeugenden Elemente ein: 100000
```

```
Erzeugen von 100000 Stringelementen mit der String-Klasse...
Dauer: 2294 Millisekunden
```

```
Erzeugen von 100000 Stringelementen mit der StringBuilder-Klasse...
Dauer: 1111 Millisekunden
```

Sie sehen, dass bei einer Elementlänge von *100* Zeichen die Verwendung der *StringBuilder*-Klasse bereits eine Verdopplung der Geschwindigkeit mit sich bringt.

Starten Sie anschließend das Programm erneut. Geben Sie für die Elementlänge *1000* ein und bestimmen Sie für die Anzahl der zu erzeugenden Elemente den Wert *10000*. Die Geschwindigkeitsausbeute ist jetzt noch beeindruckender:

```
Geben Sie die String-Länge eines Elementes ein: 1000
Geben die Anzahl der zu erzeugenden Elemente ein: 10000
```

```
Erzeugen von 10000 Stringelementen mit der String-Klasse...
Dauer: 6983 Millisekunden
```

```
Erzeugen von 10000 Stringelementen mit der StringBuilder-Klasse...
Dauer: 1091 Millisekunden
```

Mit diesen Parametern ist der *StringBuilder* ca. um den Faktor 6 schneller – im Vergleich zum normalen *String*-Objekt!

Je mehr Zeichen für einen String zur Laufzeit generiert werden müssen, desto mehr lohnt sich der Einsatz eines *StringBuilder*-Objektes.

Das Programm selbst greift für die Messungen übrigens auf die Klasse *HighSpeedTimeGauge* zurück, zu dem der anschließende graue Kasten mehr zu sagen weiß. Das folgende Listing zeigt seine Verwendungsweise:

```
Imports System.Text

Module StringsVsStringBuilder

    Sub Main()
        Dim locTimeGauge As New HighSpeedTimeGauge
        Dim locAmountElements As Integer
        Dim locAmountCharsPerElement As Integer
        Dim locVBStringElements As VBStringElements
        Dim locVBStringBuilderElements As VBStringBuilderElements

        'StringBuilderBeispiele()
        'Return

        Console.WriteLine("Geben Sie die String-Länge eines Elementes ein: ")
        locAmountCharsPerElement = Integer.Parse(Console.ReadLine)
        Console.WriteLine("Geben die Anzahl der zu erzeugenden Elemente ein: ")
        locAmountElements = Integer.Parse(Console.ReadLine)
        Console.WriteLine()
        Console.WriteLine("Erzeugen von " & locAmountElements & _
            " Stringelementen mit der String-Klasse...")
        locTimeGauge.Start()
```

```

locVBStringElements = New VBStringElements(locAmountElements, locAmountCharsPerElement)
locTimeGauge.Stop()
Console.WriteLine("Dauer: " & locTimeGauge.ToString())
locTimeGauge.Reset()
Console.WriteLine()
locTimeGauge.Reset()
Console.WriteLine("Erzeugen von " & locAmountElements &
    " Stringelementen mit der StringBuilder-Klasse...")
locTimeGauge.Start()
locVBStringBuilderElements = New VBStringBuilderElements(locAmountElements, _
    locAmountCharsPerElement)
locTimeGauge.Stop()
Console.WriteLine("Dauer: " & locTimeGauge.ToString())
locTimeGauge.Reset()
Console.WriteLine()
Console.ReadLine()
End Sub

Sub StringBuilderBeispiele()

'Deklaration ohne Parameter:
Dim locSB As New StringBuilder
'Deklaration mit Kapazitätsreservierung
locSB = New StringBuilder(1000)
'Deklaration aus einem vorhandenen String
locSB = New StringBuilder("Aus einem neuen String entstanden")
'Deklaration aus String mit der Angabe einer zu reservierenden Kapazität
locSB = New StringBuilder("Aus String entstanden mit Kapazität für weitere", 1000)

locSB.Append(" - und das wird an den String angefügt")
locSB.Insert(20, ">>das kommt irgendwo in die Mitte<<")
locSB.Replace("String", "StringBuilder")
locSB.Remove(0, 4)

'StringBuilder hat den String fertig zusammengesetzt,
'in String umwandeln
Dim locString As String = locSB.ToString
Console.WriteLine(locString)
Console.ReadLine()
End Sub
End Module

Public Class VBStringElements
Private myStrElements() As String

Sub New(ByVal AmountOfElements As Integer, ByVal AmountChars As Integer)

ReDim myStrElements(AmountOfElements - 1)
Dim locRandom As New Random(DateTime.Now.Millisecond)
Dim locString As String

For locOutCount As Integer = 0 To AmountOfElements - 1
    locString = ""

```

```

For locInCount As Integer = 0 To AmountChars - 1
    Dim locIntTemp As Integer = Convert.ToInt32(locRandom.NextDouble * 52)
    If locIntTemp > 26 Then
        locIntTemp += 97 - 26
    Else
        locIntTemp += 65
    End If
    locString += Convert.ToChar(locIntTemp).ToString
Next
myStrElements(locOutCount) = locString
Next
End Sub
End Class
Public Class VBStringBuilderElements

    Private myStrElements() As String

    Sub New(ByVal AmountOfElements As Integer, ByVal AmountChars As Integer)

        ReDim myStrElements(AmountOfElements - 1)
        Dim locRandom As New Random(DateTime.Now.Millisecond)
        Dim locStringBuilder As StringBuilder

        For locOutCount As Integer = 0 To AmountOfElements - 1
            locStringBuilder = New StringBuilder(AmountChars)
            For locInCount As Integer = 0 To AmountChars - 1
                Dim locIntTemp As Integer = Convert.ToInt32(locRandom.NextDouble * 52)
                If locIntTemp > 26 Then
                    locIntTemp += 97 - 26
                Else
                    locIntTemp += 65
                End If
                locStringBuilder.Append(Convert.ToChar(locIntTemp))
            Next
            myStrElements(locOutCount) = locStringBuilder.ToString
        Next
    End Sub
End Class

```

## Wrapper-Klassen für Betriebssystemaufrufe

Es ist eine allgemein übliche Vorgehensweise, Aufrufe an das Windows-Betriebssystem durch so genannte Wrapper-Klasse zu realisieren. Eine Wrapper-Klasse kapselt<sup>12</sup> Betriebssystemaufrufe, sodass sie auf gewohnte »Framework«-Weise aufrufbar sein. Wrapper-Klassen stellen dazu ganze Funktionsbibliotheken zur Verfügung, die nur als Schnittstelle zu den dafür benötigten Betriebssystemen fungieren, die aber die eigentliche Aufgabe übernehmen. Viele Steuerelemente von Windows sind auf diese Weise im Framework integriert.

---

<sup>12</sup> von engl. »to wrap«: einpacken (etwa in Geschenkpapier)

Schon ein simples TextBox-Steuerelement ist im Framework nicht von Grund auf neu entwickelt worden. Vielmehr bildet die TextBox-Klasse des Frameworks lediglich einen Wrapper um die Windows-TextBox und stellt entsprechende Bearbeitungsfunktionen FCL-konform zur Verfügung.

Die im Beispiel verwendeten Betriebssystemaufrufe steuern den Performance-Counter (etwa: Leistungsmesser), der durch den Windows-Kernel zur Verfügung gestellt wird. Er erlaubt eine viel genauere Zeitmessung, insbesondere von extrem kurz andauernden Operationen und liefert daher aussagekräftigere Zahlen, als Sie diese über die normalen DateTime-Funktionen ermitteln könnten. Das folgende Listing zeigt die Funktionsweise:

```
Option Explicit On
Option Strict On

Public Class HighSpeedTimeGauge

    'Die Routinen brauchen wir zum "Hochgeschwindigkeitsmessen" aus dem Kernel
    Declare Auto Function QueryPerformanceFrequency Lib "Kernel32" (ByRef lpFrequency As Long) _
        As Boolean
    Declare Auto Function QueryPerformanceCounter Lib "Kernel32" (ByRef lpPerformanceCount As Long) _
        As Boolean

    'So ginge es übrigens auch:
    '<System.Runtime.InteropServices.DllImport("KERNEL32")>
    'Private Shared Function QueryPerformanceCounter(ByRef lpPerformanceCount As Long) As Boolean
    'End Function

    '<System.Runtime.InteropServices.DllImport("KERNEL32")>
    'Private Shared Function QueryPerformanceFrequency(ByRef lpFrequency As Long) As Boolean
    'End Function

    Private myStartTime As Long = 0
    Private myEndTime As Long = 0
    Private myDuration As Long = 0
    Private myFrequency As Long = 0

    Public Sub New()

        QueryPerformanceFrequency(myFrequency)

    End Sub

    Public Sub Start()

        myStartTime = 0
        QueryPerformanceCounter(myStartTime)

    End Sub

    Public Sub [Stop]()

        myEndTime = 0
        QueryPerformanceCounter(myEndTime)

    End Sub
```



```

myDuration = myEndValue - myStartValue

End Sub

Public Sub Reset()

    myStartValue = 0
    myEndValue = 0
    myDuration = 0
End Sub
Public ReadOnly Property DurationInSeconds() As Double
    Get
        Return CDbl(myDuration) / CDbl(myFrequency)
    End Get
End Property

Public ReadOnly Property DurationInMilliseconds() As Long
    Get
        Return CLng(1000 * DurationInSeconds)
    End Get
End Property

Public ReadOnly Property Frequency() As Long
    Get
        Return myFrequency
    End Get
End Property

Public Overrides Function ToString() As String
    Return DurationInMilliseconds & " Millisekunden"
End Function
End Class

```

Der Einsatz dieser Klasse ist denkbar einfach. Mit einer Klasseninstanz haben Sie Zugriff auf die Funktionen Start, Stop und Reset, mit denen Sie den Performance-Counter starten, anhalten und zurücksetzen können. Die Funktion ToString liefert Ihnen eine gemessene Zeitperiode in Millisekunden zurück.

## Der Datentyp Boolean

Der Boolean-Datentyp speichert binäre Zustände, also eigentlich nicht viel: Sein Wert kann entweder falsch oder wahr sein – etwas anderes kann er nicht speichern. Dieser Datentyp wird am häufigsten bei der Ausführung von bedingtem Programmcode verwendet; zu diesem Thema erfahren Sie mehr im ► Abschnitt »Anweisungen, die bedingten Programmcode ausführen« auf Seite 485.

**.NET-Datentyp:** System.Boolean

**Stellt dar:** einen von zwei Zuständen – True oder False

**Typliteral:** keins vorhanden

**Speicherbedarf:** 2 Byte

**Delegation an den Prozessor:** ja, als Integer

**Anmerkungen:**

Wenn Sie eine boolesche Variable definieren wollen, verwenden Sie dazu die Schlüsselwörter True und False direkt und ohne Anführungszeichen im Programmtext, etwa wie im folgenden Beispiel:

```
Dim locBoolean As Boolean  
locBoolean = True ' Ausdruck ist 'wahr'.  
locBoolean = False ' Ausdruck ist 'falsch'.
```

## Konvertieren von und in numerische Datentypen

Sie können einen booleschen Typ in einen numerischen Datentyp umwandeln.

---

**WICHTIG:** Beachten Sie, dass Visual Basic .NET bei der internen Darstellung des primitiven Datentyps Boolean vom Framework abweicht. Wenn Sie mit Visual Basic-Befehlen einen Boolean- in beispielsweise einen Integer-Datentyp umwandeln, wird der Wert True in -1 umgewandelt. Wenn Sie mit Framework-Konvertierungen – also beispielsweise der Convert-Klasse – arbeiten, wird True zu +1 umgewandelt.

Das folgende Beispiel verdeutlicht, was gemeint ist:

---

```
Dim locInt As Integer = CInt(locBoolean)      ' locInt ist -1  
locInt = Convert.ToInt32(locBoolean)          ' locInt ist jetzt +1!!!  
Dim locLong As Long = CLng(locBoolean)        ' locLong ist -1  
locLong = Convert.ToInt64(locBoolean)          ' locLong ist +1
```

Bei der umgekehrten Konvertierung ist das Verhalten der Convert-Klasse des Frameworks und den Konvertierungsanweisungen von Visual Basic .NET einerlei. Nur der Zahlenwert 0 ergibt das boolesche Ergebnis False, alle anderen Werte ergeben True, wie das folgende Beispiel zeigt.

---

**BEGLEITDATEIEN:** Sie finden dieses Beispiel und weitere Beispiele zusammengefasst im Projekt im Verzeichnis *.\VB 2005 - Entwicklerbuch\E - Datentypen\Kap16\Primitives03*.

---

```
locBoolean = CBool(-1)                      ' locBoolean ist True.  
locBoolean = CBool(0)                        ' locBoolean ist False.  
locBoolean = CBool(1)                        ' locBoolean ist True.  
locBoolean = Convert.ToBoolean(-1)           ' locBoolean ist True.  
locBoolean = Convert.ToBoolean(+1)           ' locBoolean ist True.  
locBoolean = CBool(100)                       ' locBoolean ist True.  
locBoolean = Convert.ToBoolean(100)           ' locBoolean ist True.
```

## Konvertierung von und in Strings

Wenn Sie einen booleschen Datentyp in eine Zeichenkette konvertieren – beispielsweise um ihren Status in einer Datei abzuspeichern –, wird der jeweilige Wert in eine Zeichenkette umgewandelt, die durch die statischen Nur-Lese-Eigenschaften `TrueString` und `FalseString` der Boolean-Struktur festgehalten sind. Sie ergeben in der aktuellen Framework-Version (2.0) »True« und »False«, egal, welche Kultureinstellungen für das jeweilige Framework gelten.

Bei der Konvertierung von String in Boolean ergibt jede Zeichenkette außer »True« den Wert False.

## Vergleichsoperatoren, die boolesche Ergebnisse zurückliefern

Visual Basic kennt die folgenden so genannten Vergleichsoperatoren, die zwei Ausdrücke miteinander vergleichen und ein boolesches Ergebnis zurückliefern:

- Ausdruck1 = Ausdruck2: Prüft auf gleich; liefert True zurück, wenn beide Ausdrücke gleich sind.
- Ausdruck1 > Ausdruck2: Prüft auf größer; liefert True zurück, wenn Ausdruck1 größer als Ausdruck2 ist.
- Ausdruck1 < Ausdruck2: Prüft auf kleiner; liefert True, wenn Ausdruck1 kleiner als Ausdruck2 ist.
- Ausdruck1 >= Ausdruck2: Prüft auf größer/gleich; liefert True zurück, wenn Ausdruck1 größer oder gleich Ausdruck2 ist.
- Ausdruck1 <= Ausdruck2: Prüft auf kleiner/gleich; liefert True, wenn Ausdruck1 größer oder gleich Ausdruck2 ist.
- Ausdruck1 <> Ausdruck2: Prüft auf ungleich; liefert True, wenn Ausdruck1 nicht Ausdruck2 entspricht.
- Ausdruck1 Is [Ausdruck2|Nothing]: Prüft auf Gleichheit eines Objektverweises (nur auf Referenztypen anwendbar); liefert True zurück, wenn Ausdruck1 auf den gleichen Datenspeicherbereich wie Ausdruck2 zeigt. Wenn Ausdruck1 keinem Speicherbereich zugewiesen ist (definierte Objektvariable aber kein instanziertes Objekt), liefert der Vergleich durch Is mit Nothing den booleschen Wert True zurück.
- Ausdruck1 IsNot [Ausdruck2|Nothing]: Prüft auf Ungleichheit eines Objektverweises (nur auf Referenztypen anwendbar); liefert True zurück, wenn Ausdruck1 auf einen anderen Datenspeicherbereich wie Ausdruck2 zeigt. Wenn Ausdruck1 auf einen gültigen Speicherbereich mit Instanzdaten verweist, liefert der Vergleich durch IsNot mit Nothing den booleschen Wert True zurück.
- String1 Like String2: Prüft auf Ähnlichkeit zweier Strings; ein Mustervergleich kann den Vergleich flexibilisieren. Bei vorliegender Gleichheit der beiden Strings nach bestimmten Regeln<sup>13</sup> wird True zurückgeliefert, anderenfalls False.

Die folgenden Codezeilen demonstrieren den Einsatz der Vergleichsoperatoren:

```
Dim locString1 As String = "Uwe"
Dim locString2 As String = "Klaus"

locBoolean = (locString1 = locString2)      ' Ergibt False.
locBoolean = (locString1 > locString2)       ' Ergibt True.
locBoolean = (locString1 < locString2)       ' Ergibt False.
locBoolean = (locString1 >= locString2)      ' Ergibt True.
locBoolean = (locString1 <= locString2)      ' Ergibt False.
locBoolean = (locString1 <> locString2)      ' Ergibt True.
locBoolean = (locString1 Is locString2)       ' Ergibt False.

locString2 = "Uwe"
String.Intern(locString2)                  ' Ergibt jetzt True, da beide
locBoolean = (locString1 Is locString2)      ' Stringobjekte auf einen Bereich zeigen.
```

---

<sup>13</sup> Genauereres erfahren Sie in der MSDN unter dem Schlagwort *Like-Operator*.

```

locString1 = "Klau's, und lass Dich nicht erwischen"
locString2 = "Klau*"
locBoolean = (locString1 Like locString2) ' Ergibt True.

```

## Anweisungen, die bedingten Programmcode ausführen

Der Boolean-Datentyp wird in der Regel bei der Auswertung von Entscheidungen benötigt. Mit ihm können Sie in Abhängigkeit seines Wertes steuern, ob Programmcode ausgeführt wird. Dafür verwenden Sie die If-, Case- [Is], While- oder Until-Anweisungen.

Die IIf-Funktion steuert zwar nicht den Programmablauf, sollte aber auf Grund ihrer Ähnlichkeit zur If-Anweisung, ebenfalls in diesem Zusammenhang erwähnt werden. Sie liefert ein Funktionsergebnis auf Grund des booleschen Wertes, der ihr übergeben wird. Ist der übergebene Wert True, wird das erste mögliche Funktionsergebnis zurückgeliefert, ist er False, das zweite.

### If ... Then ... Else ... Elself ... EndIf

Die If-Anweisung haben Sie höchstwahrscheinlich schon hunderte Male angewendet und wissen deshalb aus dem Effeff, wie sie funktioniert. Der Vollständigkeit halber möchte ich sie dennoch ein wenig genauer unter die Lupe nehmen:

In der einfachsten Form wird bei der If-Anweisung der Code ausgeführt, der zwischen If und End If positioniert wird, wenn der hinter If stehende boolesche Ausdruck True wird. Obwohl Basic seit Jahren im Einsatz ist, gibt es immer noch Entwickler, die das Konzept von Vergleichen mit booleschen Ausdrücken nicht verinnerlicht und Schwierigkeiten beim Verständnis folgender Konstrukte haben:

```

locBoolean = True
If locBoolean Then
    'Wird nur ausgeführt, wenn locBoolean True ist.
End If

```

Einige Entwickler verstehen nicht, wieso hier nicht der folgende Ausdruck zum Einsatz kommen muss:

```

locBoolean = True
If locBoolean = True Then
    'Wird nur ausgeführt, wenn locBoolean True ist.
End If

```

Tatsache ist, das der Ausdruck

```
locBoolean = True
```

in diesem Ausdruck keine Besonderheit der If-Anweisung ist, sondern im Prinzip eine ganz normale Funktion. Wenn locBoolean den Wert True hat, ist der gesamte Ausdruck natürlich ebenfalls wieder True. If macht nichts weiter, als den dahinter stehenden booleschen Wert zu untersuchen und die nachstehenden Anweisungen nur dann auszuführen, wenn der Wert True war. Aus diesem Grund braucht der Wert nicht noch zusätzlich durch Eingreifen des Programmierers überprüft zu werden; das wäre redundant. Denn das Gleichheitszeichen ist hier ja der Vergleichsoperator! Wenn man locBoolean durch seinen augenblicklichen Wert ersetzt, lautet die Überprüfung

```
If True = True Then ' (Wenn wahr wahr ist, dann)
```

was man ziemlich selbstsicher durch If True Then ... oder If locBoolean Then ... ersetzen kann (wenn es wahr ist, dann).

Es ist in Basic (nicht nur in Visual Basic) aber in der Tat verwirrend, dass Zuweisungsoperator und Vergleichsoperator mit denselben Zeichen angewandt werden. Der Ausdruck:

```
Dim locBoolean = "Klaus" = "Uwe"
```

ist aber natürlich gültig. Das erste Gleichheitszeichen fungiert hier als Zuweisungsoperator, das zweite als boolescher Vergleichsoperator. In diesem Beispiel nimmt locBoolean den Wert False an, weil die Zeichenkette »Klaus« nicht »Uwe« entspricht. Der Vergleichsoperator hat vor dem Zuweisungsoperator die höhere Priorität. Andernfalls käme es bei diesem Beispiel auch zu einem Typkonvertierungsfehler.

Aber auch der andere Weg ist nicht wirklich überzeugend, C++ benutzt beispielsweise = für die Zuweisung, == für den Vergleich. Das ist aber alles andere als intuitiv. Allein 5 % – 8 % aller Fehler in C++ Programmen geht nämlich auf diese Verwechslung zurück. Und das ist viel häufiger als die Anzahl der Fehler, die durch die Falschbenutzung des = Zeichens in VB passieren.

Dem If-Codeblock kann auch ein Else-Codeblock folgen, der ausgeführt wird, wenn der boolesche Ausdruck hinter If den Wert False annahm. Darüber hinaus können Sie mit ElseIf weitere Auswertungen in das If-Konstrukt einschieben. Der Codeblock hinter dem letzten Else-Codeblock wird, falls vorhanden, nur dann ausgeführt, wenn keine der Bedingungen der einzelnen If- bzw. ElseIf-Sektionen True ergaben.

Ein Beispiel:

```
locString1 = "Klau's, und lass Dich nicht erwischen"
locString2 = "Klaus*"
locBoolean = (locString1 Like locString2)      ' ergibt False

If locBoolean Then
    'Schachteln geht natürlich auch:
    If locString2 = "Klaus" Then
        Console.WriteLine("Namen gefunden!")
    Else
        Console.WriteLine("Keinen Namen gefunden!")
    End If
ElseIf Now = #12:00:00 PM# Then
    Console.WriteLine("Mittag!")
ElseIf Now = #12:00:00 AM# Then
    Console.WriteLine("So spät noch auf?")
Else
    Console.WriteLine("Es ist irgendwann sonst oder locString1 war nicht wie locString1...")
End If
```

## Select ... Case ... End Select

Sie können, wie im vorherigen Abschnitt zu sehen war, `ElseIf` zur Optionsanalyse verwenden, wenn Sie mehrere boolesche Ausdrücke in einem Rutsch prüfen und darauf mit der Ausführung entsprechenden Programmcodes reagieren müssen. Mit einem `Select`-Konstrukt geht das allerdings sehr viel eleganter. `Select` bereitet einen Ausdruck für einen Vergleich mit booleschem Ergebnis vor; der eigentliche Vergleich findet dann durch verschiedene `Case`-Anweisungen, aber mindestens eine `Case`-Anweisung statt, hinter denen jeweils ein entsprechendes Vergleichsargument vom gleichen Typ (oder implizit konvertierbar) folgen muss. `Case Else` kann optional für die Ausführung von Anweisungen herangezogen werden, wenn keine der hinter `Case` angegebenen Bedingungen zutraf. `End Select` schließt das Konstrukt ab. Andererseits führt `Select` auch keine weiteren Überprüfungen durch, wenn einmal eine Bedingungsprüfung positiv verlief.

Bei der Bedingungsprüfung prüft `Case` ohne Zusatz auf Gleichheit. Durch Verwenden des `Is`-Schlüsselwortes können Sie auch andere Vergleichsoperatoren verwenden. Das folgende Beispiel soll den Umgang verdeutlichen:

```
Dim locString1 as String = "Miriam"

Select Case locString1
    Case "Miriam"
        Console.WriteLine("Treffer")
    Case Is > "M"
        Console.WriteLine("Name kommt nach 'M' im Alphabet")

    Case Is < "M"
        Console.WriteLine("Name kommt vor 'M' im Alphabet")

    Case Else
        Console.WriteLine("Name beginnt mir 'M'")

    'case like "Miri"
    'Das funktioniert nicht!!!
End Select
```

Allerdings werden hier Vergleichsoperation und bedingte Ausführung in einem Abwasch gemacht, sodass das folgende Konstrukt nicht funktioniert:

```
'Das funktioniert so nicht!!!
Select Case locBoolean

    case: Console.WriteLine("War wahr!")

End Select
```

Der Compiler meckert hier zu Recht.

## Schleifenabbruchbedingungen

Nur der Vollständigkeit halber sei der Einsatz von booleschen Variablen auch für Schleifenabbruchbedingungen erwähnt. Sowohl bei `While/End While` als auch bei `Do/Loop` dienen boolesche Werte als Abbruchbedingungen.

```

Dim locCount As Integer

'Raufzählen.
Do While locCount < 10
    locCount += 1
Loop

'locCount ist jetzt 10; wieder runterzählen.
Do
    locCount -= 1
Loop Until locCount = 0

'locCount ist jetzt 0; wieder raufzählen.
While locCount < 10
    locCount += 1
End While

'locCount ist wieder 10; und wieder bis 0 runterzählen.
Do Until locCount = 0
    locCount -= 1
Loop

```

## Der Datentyp Date

Mit dem Date-Datentyp speichern Sie Datumswerte. Er hilft Ihnen, Zeitdifferenzen zu berechnen, Datumswerte aus Zeichenketten einzulesen (zu parsen) und wieder formatiert in Zeichenketten zurückzuverwandeln.

**.NET-Datentyp:** System.DateTime

**Stellt dar:** Zeiten und Uhrzeiten im Bereich vom 1.1.0001 (0 Uhr) bis 31.12.9999 (23:59:59 Uhr) mit einer Auflösung von 100 Nanosekunden (diese Einheit wird als *Tick* bezeichnet)

**Typliteral:** keins vorhanden

**Speicherbedarf:** 8 Byte

**Delegation an den Prozessor:** ja, als *Long*

**Anmerkungen:** Da Date-Datentypen ebenfalls zu den primitiven Datentypen zählen, sind sie direkt durch Literale im Programmcode definierbar. Allerdings gilt hier das Gleiche wie bei numerischen Werten: Das englische bzw. amerikanische Darstellungsformat zählt. Falls Sie mit dieser Kultur – was die Zeitmessung und Schreibweise anbelangt – nicht so sehr vertraut sind, lassen Sie mich kurz die Besonderheiten erklären.

Datums-Werte werden in der Reihenfolge Monat/Tag/Jahr vorgenommen und durch Schrägstrich und nicht durch Punkt von einander getrennt. Diese Schreibweise kann leicht für Verwirrung sorgen (und zu totalen Fehlbuchungen führen), wenn Sie nicht darüber Bescheid wissen. Beim Datum

12/17/2004

ahnhen Sie noch, dass etwas nicht stimmt, da es keinen 17. Monat im Jahr gibt.

Aber es macht schon einen Unterschied, ob Sie das Datum

12/06/2004

als 12. Juni oder 6. Dezember interpretieren.

Ähnliches gilt für die Uhrzeit. Die 24-Stunden-Anzeige finden Sie in den USA vielleicht auf dem einen oder anderen Busfahrplan oder beim Militär. Ansonsten wird durch die Postfixe »AM« (für »ante meridiem«, etwa »am Vormittag«) und »PM« (»post meridiem«, etwa »am Nachmittag«) definiert, welches 3:00 Uhr beispielsweise gemeint ist. Kritisch wird es bei 12:00 (0:00 gibt es nicht!) – vielleicht haben Sie selbst schon mal die Erfahrung gemacht, ein amerikanisches Videorekordermodell zu programmieren, das nicht die gewünschte Sendung, sondern eine andere, 12 Stunden später ausgestrahlte, aufgenommen hat.

12:00 AM entspricht unseren 0:00 Uhr, also Mitternacht; 12:00 PM entspricht Mittag.

Wertzuweisungen an einen Date-Datentyp im Programmcode werden durch Einklammern in das Nummernzeichen (»#«) realisiert. Die folgenden Beispiele zeigen, wie es geht:

```
Dim Mitternacht As Date = #12:00:00 AM#
Dim Mittag As Date = #12:00:00 PM#
Dim Silvester As System.DateTime = #12/31/2004#
Dim ZeitFürSekt As System.DateTime = #12/31/2004 11:58:00 PM#
Dim ZeitFürAsperin As System.DateTime = #1/1/2005 11:58:00 AM#
```

Der halbwegs intelligente Editor hilft Ihnen dabei ein wenig, das korrekte Format zu finden. Den Ausdruck

#0:00#

wandelt er beispielsweise selbstständig in

#12:00:00 AM#

um. Er ergänzt ebenfalls fehlende Sekundeneingaben oder Nullen, wenn die Eingabe eines Bereichs nur einstellig erfolgte. Sie können Uhrzeiten auch im 24-Stunden-Format eingeben; der Editor wandelt diese Zeit dann automatisch ins 12-Stunden-Format um.

## Rechnen mit Zeiten und Datumswerten – TimeSpan

Das Besondere am Date-Datentyp ist, dass er das Errechnen von Zeitdifferenzen erlaubt. Um jedoch sinnvolle Ergebnisse, beispielsweise die Differenz zweier Datumswerte oder Zeitwerte dazustellen, ist der Date-Datentyp selbst nicht sonderlich geeignet. Aus diesem Grund gibt es einen weiteren Datentyp – TimeSpan – der Ergebnisse von Zeitberechnungen aufnimmt. Er selbst zählt allerdings nicht zu den primitiven Datentypen von .NET.

Der Umgang mit diesem Datentyp ist sehr einfach: Sie können entweder einen Datumswert von einem anderen subtrahieren, um die dazwischenliegende Zeitspanne zu ermitteln. Oder Sie erstellen einen TimeSpan-Datentyp und verwenden ihn, um einen bestimmten Zeitabschnitt auf ein Datum zu addieren oder es von ihm abzuziehen. Ein paar Beispielcodezeilen sollen das verdeutlichen.

---

**BEGLEITDATEIEN:** Sie finden dieses Beispiel im Verzeichnis |VB 2005 - Entwicklerbuch|E - Datentypen|Kap16|Date-Time. Starten Sie das Beispiel mit **Strg + F5**.

---

```

Dim locDate1 As Date = #3:15:00 PM#
Dim locDate2 As Date = #4:23:32 PM#
Dim locTimeSpan As TimeSpan = locDate2.Subtract(locDate1)
Console.WriteLine("Der Zeitunterschied zwischen {0} und {1} beträgt", _
    locDate1.ToString("HH:mm:ss"), _
    locDate2.ToString("HH:mm:ss"))
Console.WriteLine("{0} Sekunde(n) oder", locTimeSpan.TotalSeconds)
Console.WriteLine("{0} Minute(n) und {1} Sekunde(n) oder", _
    Math.Floor(locTimeSpan.TotalMinutes), _
    locTimeSpan.Seconds)
Console.WriteLine("{0} Stunde(n), {1} Minute(n) und {2} Sekunde(n) oder", _
    Math.Floor(locTimeSpan.TotalHours), _
    locTimeSpan.Minutes, locTimeSpan.Seconds)
Console.WriteLine("{0} Ticks", _
    locTimeSpan.Ticks)

```

## Bibliothek mit brauchbaren Datumsrechenfunktionen

Sie finden im gleichen Beispiel eine Klassendatei namens *DateCalcHelper.vb*, die eine statische Klasse gleichen Namens enthält. Diese Klasse stellt einige, wie ich meine, ganz brauchbare Funktionen zur Verfügung, die einerseits die Berechnung bestimmter relativer Zeitpunkte vereinfacht und andererseits das Rechnen mit Datumswerten verdeutlicht.

Die Klasse erklärt sich durch die XML-Kommentare selber – das komplette Listing dieser Klasse finden Sie im Folgenden. Wenn Sie eigene Programme entwickeln, die intensiv von Berechnungen relativer Zeitpunkte Gebrauch macht, fügen Sie diese Codedatei einfach Ihrem Projekt (oder der Assembly Ihres Projekts) hinzu.

Zur besseren Orientierung sind die Funktionserklärungen und Methodennamen der einzelnen Funktionen im folgenden Listing in Fettschrift gesetzt,

```

Public NotInheritable Class DateCalcHelper

    ''' <summary>
    ''' Errechnet das Datum, das dem 1. des Monats entspricht,
    ''' der sich aus dem angegebenen Datum ergibt.
    ''' </summary>
    ''' <param name="CurrentDate">Datum, dessen Monat für die Berechnung zugrunde gelegt wird.</param>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Public Shared Function FirstDayOfMonth(ByVal CurrentDate As Date) As Date
        Return New Date(CurrentDate.Year, CurrentDate.Month, 1)
    End Function

    ''' <summary>
    ''' Errechnet das Datum, das dem Letzen des Monats entspricht,
    ''' der sich aus dem angegebenen Datum ergibt.
    ''' </summary>
    ''' <param name="CurrentDate">Datum, dessen Monat für die Berechnung zugrunde gelegt wird.</param>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Public Shared Function LastDayOfMonth(ByVal CurrentDate As Date) As Date

```

```

        Return New Date(CurrentDate.Year, CurrentDate.Month, 1).AddMonths(1).AddDays(-1)
End Function

''' <summary>
''' Errechnet das Datum, das dem 1. des Jahres entspricht,
''' das sich aus dem angegebenen Datum ergibt.
''' </summary>
''' <param name="CurrentDate">Datum, dessen Jahr für die Berechnung zugrunde gelegt wird.</param>
''' <returns></returns>
''' <remarks></remarks>
Public Shared Function FirstOfYear(ByVal CurrentDate As Date) As Date
    Return New Date(CurrentDate.Year, 1, 1)
End Function

''' <summary>
''' Errechnet das Datum, das dem ersten Montag der ersten Woche des Monats entspricht,
''' der sich aus dem angegebenen Datum ergibt.
''' </summary>
''' <param name="CurrentDate">Datum, dessen Woche für die Berechnung zugrunde gelegt wird.</param>
''' <returns></returns>
''' <remarks></remarks>
Public Shared Function MondayOfFirstWeekOfMonth(ByVal CurrentDate As Date) As Date
    Dim locDate As Date = FirstDayOfMonth(CurrentDate)
    If Weekday(locDate) = DayOfWeek.Monday Then
        Return locDate
    End If
    Return locDate.AddDays(6 - Weekday(CurrentDate))
End Function

''' <summary>
''' Errechnet das Datum, das dem Montag der Woche entspricht,
''' die sich aus dem angegebenen Datum ergibt.
''' </summary>
''' <param name="CurrentDate">Datum, dessen Woche für die Berechnung zugrunde gelegt wird.</param>
''' <returns></returns>
''' <remarks></remarks>
Public Shared Function MondayOfWeek(ByVal CurrentDate As Date) As Date
    If Weekday(CurrentDate) = DayOfWeek.Monday Then
        Return CurrentDate
    Else
        Return CurrentDate.AddDays(-Weekday(CurrentDate) + 1)
    End If
End Function

''' <summary>
''' Errechnet das Datum, das dem ersten Montag der zweiten Woche des Monats entspricht,
''' der sich aus dem angegebenen Datum ergibt.
''' </summary>
''' <param name="CurrentDate">Datum, dessen Woche für die Berechnung zugrunde gelegt wird.</param>
''' <returns></returns>
''' <remarks></remarks>
Public Shared Function MondayOfSecondWeekOfMonth(ByVal currentDate As Date) As Date
    Return MondayOfFirstWeekOfMonth(currentDate).AddDays(7)

```

```

End Function

''' <summary>
''' Errechnet das Datum, das dem ersten Montag der letzten Woche des Monats entspricht,
''' der sich aus dem angegebenen Datum ergibt.
''' </summary>
''' <param name="CurrentDate">Datum, dessen Woche für die Berechnung zugrunde gelegt wird.</param>
''' <returns></returns>
''' <remarks></remarks>
Public Shared Function MondayOfLastWeekOfMonth(ByVal CurrentDate As Date) As Date
    Dim locDate As Date = FirstDayOfMonth(CurrentDate).AddDays(-1)
    If Weekday(locDate) = DayOfWeek.Monday Then
        Return locDate
    End If
    Return locDate.AddDays(-Weekday(CurrentDate) + 1)
End Function

''' <summary>
''' Ergibt das Datum des nächsten Arbeitstages.
''' </summary>
''' <param name="CurrentDate">Datum der Berechnungsgrundlage</param>
''' <param name="WorkOnSaturdays">True, wenn Samstag Arbeitstag ist.</param>
''' <param name="WorkOnSundays">True, wenn Sonntag Arbeitstag ist.</param>
''' <returns></returns>
''' <remarks></remarks>
Public Shared Function NextWorkday(ByVal CurrentDate As Date, ByVal WorkOnSaturdays As Boolean, _
    ByVal WorkOnSundays As Boolean) As Date
    CurrentDate = CurrentDate.AddDays(1)
    If Weekday(CurrentDate) = DayOfWeek.Saturday And Not WorkOnSaturdays Then
        CurrentDate = CurrentDate.AddDays(1)
    End If
    If Weekday(CurrentDate) = DayOfWeek.Sunday And Not WorkOnSundays Then
        CurrentDate = CurrentDate.AddDays(1)
    End If
    Return CurrentDate
End Function

''' <summary>
''' Ergibt das Datum des vorherigen Arbeitstages.
''' </summary>
''' <param name="CurrentDate">Datum der Berechnungsgrundlage</param>
''' <param name="WorkOnSaturdays">True, wenn Samstag Arbeitstag ist.</param>
''' <param name="WorkOnSundays">True, wenn Sonntag Arbeitstag ist.</param>
''' <returns></returns>
''' <remarks></remarks>
Public Shared Function PreviousWorkday(ByVal CurrentDate As Date, ByVal WorkOnSaturdays As Boolean, _
    ByVal WorkOnSundays As Boolean) As Date
    CurrentDate = CurrentDate.AddDays(-1)
    If Weekday(CurrentDate) = DayOfWeek.Sunday And Not WorkOnSundays Then
        CurrentDate = CurrentDate.AddDays(-1)
    End If
    If Weekday(CurrentDate) = DayOfWeek.Saturday And Not WorkOnSaturdays Then
        CurrentDate = CurrentDate.AddDays(-1)

```

```

End If
Return CurrentDate
End Function
End Class

```

## Zeichenketten in Datumswerte wandeln

Genau wie die numerischen primitiven Datentypen können Sie Zeichenketten auch in einen Date-Datentyp umwandeln lassen. Allerdings stellt Ihnen der Date-Datentyp zwei Funktionen zur Verfügung, die eine Zeichenkette analysieren und den eigentlichen Datumswert daraus bilden: Parse und ParseExact.

### Umwandlungen mit Parse

Wenn Sie Parse verwenden, versucht der Parser jeden Trick, den er kennt, um ein Datum, eine Zeit oder eine Kombination aus beiden in einen Zeitwert umzuwandeln, wie das folgende Beispiel zeigt:

```

Dim locToParse As Date
locToParse = Date.Parse("13.12.04") ' OK, deutsche Grundeinstellung wird verarbeitet.
locToParse = Date.Parse("6/7/04")    ' OK, aber deutsch trotz "/" wird angewendet.
locToParse = Date.Parse("13/12/04") ' OK, wie oben.
locToParse = Date.Parse("06.07")     ' OK, wird um Jahreszahl erweitert.
locToParse = Date.Parse("06,07,04") ' OK, Komma ist akzeptabel.
locToParse = Date.Parse("06,07")     ' OK, Komma ist akzeptabel; Jahreszahl wird ergänzt.
'locToParse = Date.Parse("06072004") ' --> Exception: wurde nicht als gültiges Datum erkannt!
'locToParse = Date.Parse("060705")   ' --> Exception: wurde nicht als gültiges Datum erkannt!
locToParse = Date.Parse("6,7,4")     ' OK, Komma wird akzeptiert; führende Nullen werden ergänzt.

locToParse = Date.Parse("14:00")     ' OK, 24-Stunden-Darstellung wird akzeptiert.
locToParse = Date.Parse("PM 11:00")  ' OK, PM darf vorne...
locToParse = Date.Parse("11:00 PM")  ' ...und auch hinter der Zeitangabe stehen.
'locToParse = Date.Parse("12,00 PM") ' --> Exception: wurde nicht als gültiges Datum erkannt!

'Beide Datum- Zeitkombinationen funktionieren:
locToParse = Date.Parse("6.7.04 13:12")
locToParse = Date.Parse("6,7,04 11:13 PM")

```

Sie sehen hier aber auch, dass ein in Deutschland sehr übliches EingabefORMAT nicht erkannt wird – wenn Sie nämlich die einzelnen Wertegruppen des Datums ohne irgendein Trennzeichen hintereinander schreiben. Doch auch dafür gibt es eine Lösung.

### Umwandlungen mit ParseExact

Wenn Parse – so flexibel es auch sein mag – versagt, haben Sie die Möglichkeit, ein Erkennungsmuster für die Eingabe vorzugeben, indem Sie die Methode ParseExact benutzen, um die Zeichenkettenumwandlung vorzunehmen. ParseExact sollten Sie auch dann verwenden, wenn Sie die Eingabe eben nicht so flexibel gestalten möchten, wie Parse es vorsieht.

Insbesondere wenn es Zeitwerte und Datumswerte zu unterscheiden gilt, ist ParseExact der richtige Kandidat. In einem Feld, in dem vom Anwender Ihres Programms eine Zeiteingabe verlangt wird, weiß Ihr Programm genau, dass die Eingabe

die Uhrzeit 23:12:00 meint, und nicht den 23.12.2004. Parse kann den Kontext natürlich nicht erkennen und wird in diesem Fall nicht funktionieren.

ParseExact benötigt, neben dem zu analysierenden String, mindestens zwei weitere Parameter. Zum einen ist dies eine Zeichenkette, die das spezifische Erkennungsmuster enthält, und einen so genannten *Format Provider*, der weitere Formatierungsvorschriften vorgibt.

Es gibt verschiedene Format Provider, die Sie dafür verwenden können. Wichtig ist, dass Sie auf diese nur zugreifen können, wenn Sie zuvor folgende Zeile zur Einbindung des erforderlichen Namensbereichs am Anfang Ihres Moduls oder Ihrer Klassendatei eingefügt haben:

```
Imports System.Globalization
```

Die einfachste Version, um eine Uhrzeit als Uhrzeit zu erkennen, wenn sie im oben stehenden Format eingegeben würde, sähe beispielsweise folgendermaßen aus:

```
locToParse = Date.ParseExact("12,00", "HH,mm", CultureInfo.CurrentCulture)
```

Die Zeichenfolge »HH« besagt, dass Uhrzeiten (Hour = Stunde) im 24-Stunden-Format akzeptiert werden. Gäben Sie hier zwei kleine »h« ein, akzeptierte der Parser nur das englische 12-Stunden-Format. Mit dem anschließenden Komma geben Sie das Trennzeichen vor und anschließend die Minuten mit der Zeichenfolge »mm«.

Nun passiert es in der Praxis recht selten, dass sich Anwender an bestimmte Vorgaben halten, und darum sollte ihr Programm in der Lage sein, mehrere Versionen von Zeiteingabeformaten zu erkennen. Aus diesem Grund können Sie mit der ParseExact-Funktion gleich eine ganze Reihe von möglichen Formaten vorgeben, anhand derer der Parser die Umwandlung vornehmen kann. In diesem Fall definieren Sie einfach ein String-Array mit den zugelassenen Formaten und übergeben es anschließend der ParseExact-Methode zusammen mit der zu analysierenden Zeichenfolge. Wenn Sie sich zu dieser Methode des Parsens entschließen, müssen Sie jedoch noch einen weiteren Parameter bestimmen, der die Flexibilität des Parsens regelt (beispielsweise ob Leerzeichen im zu analysierenden String enthalten sein dürfen, die aber ignoriert werden sollen). Diese Angabe wird durch einen Parameter vom Typ *DateTimeStyles* geregelt, der folgende Einstellungen zulässt:

Member-Name	Beschreibung	Wert
AdjustToUniversal	Bestimmt, dass das Datum und die Zeit in koordinierte Weltzeit bzw. Greenwich Mean Time <sup>14</sup> (GMT) (Deutschland –1 Stunde) konvertiert werden müssen.	16
AllowInnerWhite	Bestimmt, dass zusätzliche Leerzeichen innerhalb der Zeichenfolge beim Analysieren ignoriert werden, es sei denn, die <i>DateTimeFormatInfo</i> -Formatmuster enthalten Leerzeichen.	4
AllowLeadingWhite	Bestimmt, dass vorangestellte Leerzeichen während der Analyse ignoriert werden, es sei denn, die <i>DateTimeFormatInfo</i> -Formatmuster enthalten Leerzeichen.	1
AllowTrailingWhite	Bestimmt, dass nachgestellte Leerzeichen während der Analyse ignoriert werden, es sei denn, die <i>DateTimeFormatInfo</i> -Formatmuster enthalten Leerzeichen.	2 ►

---

<sup>14</sup> Zum besseren Mitreden: Die korrekte Aussprache lautet »Grännitsch Miehn Teim«. Und auch wenn es sich so ausgesprochen wie ein Irischer Dialekt anhört – auch Engländer aus Oxford sprechen es so aus.

Member-Name	Beschreibung	Wert
AllowWhiteSpaces	Bestimmt, dass zusätzliche Leerzeichen, die sich an einer beliebigen Stelle in der Zeichenfolge befinden, während der Analyse ignoriert werden müssen, es sei denn, die <i>DateTimeFormatInfo</i> -Formatmuster enthalten Leerzeichen. Dieser Wert stellt eine Kombination aus dem <i>AllowLeadingWhite</i> -Wert, dem <i>AllowTrailingWhite</i> -Wert und dem <i>AllowInnerWhite</i> -Wert dar.	7
NoCurrentDateDefault	Datum und Zeit sind untrennbar im <i>Date</i> -Datentyp vereint. Auch wenn Sie nur eine Zeit zuweisen, spiegelt ein <i>Date</i> -Wert immer auch ein gültiges Datum wider. Diese Einstellung bestimmt, dass die <i>DateTime.Parse</i> -Methode und die <i>DateTime.ParseExact</i> -Methode das Datum nach dem gregorianische Kalender mit Jahr = 1, Monat = 1 und Tag = 1 zugrunde legen, wenn die analysierte Zeichenfolge nur die Uhrzeit und nicht das Datum enthält. Wenn dieser Wert nicht verwendet wird, wird vom gerade aktuellen Datum ausgegangen.	8
None	Bestimmt, dass die Standardformatierungsoptionen verwendet werden müssen. Dies ist das Standardformat für <i>DateTime.Parse</i> und <i>DateTime.ParseExact</i> .	0

**Tabelle 16.4:** Diese erweiterten Einstellungen können Sie mit *ParseExact* anwenden

Die folgenden Codezeilen zeigen, wie Sie *ParseExact* für die Umwandlung von Zeichenketten in Datums-Werte mit geregelten Vorgaben für zugelassene Datums-Formate einsetzen können.

```
Imports System.Globalization
```

```
Module Module1
```

```
Sub Main()

    Dim locToParseExact As Date
    Dim locZeitenMuster As String() = {"H,m", "H.m", "ddMMyy", "MM\dd\yy"}

    'Funktioniert, ist im Uhrzeitenmuster.
    locToParseExact = Date.ParseExact("12,00", _
        locZeitenMuster, _
        CultureInfo.CurrentCulture, _
        DateTimeStyles.AllowWhiteSpaces)

    'Funktioniert, ist im Uhrzeitenmuster, und "Whitespaces" sind erlaubt.
    locToParseExact = Date.ParseExact(" 12 , 00 ", _
        locZeitenMuster, _
        CultureInfo.CurrentCulture, _
        DateTimeStyles.AllowWhiteSpaces)

    'Funktioniert nicht, ist zwar im Uhrzeitenmuster, es sind aber keine "Whitespaces" erlaubt.
    'locToParseExact = Date.ParseExact(" 12 , 00 ", _
    '    locZeitenMuster, _
    '    CultureInfo.CurrentCulture, _
    '    DateTimeStyles.None)

    'Funktioniert, ist im Uhrzeitenmuster.
    locToParseExact = Date.ParseExact("1,2", _
        locZeitenMuster, _
        CultureInfo.CurrentCulture, _
        DateTimeStyles.None)
```

```

'Funktioniert, ist im Uhrzeitenmuster.
'Das Datum entspricht aber dem 1.1.0001 und wird
'als Tooltip deswegen nicht mit angezeigt, im Gegensatz
'zu allen anderen hier gezeigten Beispielen.
locToParseExact = Date.ParseExact("12.2", _
    locZeitenMuster, _
    CultureInfo.CurrentCulture, _
    DateTimeStyles.NoCurrentDateDefault)

'Funktioniert, ist nicht im Uhrzeitenmuster, da hier Sekunden mit im Spiel sind
'locToParseExact = Date.ParseExact("12,2,00", _
'    locZeitenMuster, _
'    CultureInfo.CurrentCulture, _
'    DateTimeStyles.NoCurrentDateDefault)

'Funktioniert nicht, ist mit Doppelpunkt nicht im Uhrzeitenmuster.
'locToParseExact = Date.ParseExact("1:20", _
'    locZeitenMuster, _
'    CultureInfo.CurrentCulture, _
'    DateTimeStyles.None)

'Funktioniert jetzt, da im Zeitenmuster als Datum hinterlegt.
'(drittes Element im String-Array)
locToParseExact = Date.ParseExact("241205", _
    locZeitenMuster, _
    CultureInfo.CurrentCulture, _
    DateTimeStyles.AllowWhiteSpaces)

'Funktioniert mit Übernahme im englisch-amerikanischen Format,
'da durch die Schrägstriche und der vorgegebenen Reihenfolge der Gruppen definiert.
'(viertes Element im String-Array).
locToParseExact = Date.ParseExact("12/24/05", _
    locZeitenMuster, _
    CultureInfo.CurrentCulture, _
    DateTimeStyles.AllowWhiteSpaces)

End Sub

End Module

```

Beachten Sie bei der Definition von Schrägstrichen als Gruppentrennzeichen, dass Sie ihnen jeweils ein Backslash voransetzen, damit der einfache Schrägstrich nicht als Steuerzeichen behandelt wird.

Eine Liste mit den entsprechenden Steuerzeichen und Datenformatierungen finden Sie im nächsten Kapitel. Das nächste Kapitel gibt Ihnen ebenfalls einen tieferen Einblick in den Umgang mit Format Providern.

- 
- 497 **Allgemeines über Format Provider in .NET**
  - 498 **Kulturabhängige Formatierungen mit CultureInfo**
  - 501 **Formatierung durch Formatzeichenfolgen**
  - 511 **Gezielte Formatierungen mit Format Providern**
  - 513 **Kombinierte Formatierungen**
  - 517 **So helfen Ihnen benutzerdefinierte Format Provider, Ihre Programme zu internationalisieren**
- 

Beim Durcharbeiten der vergangenen Kapitel sind Sie bereits auf die eine oder andere Funktionalität von .NET gestoßen, Zahlen oder Zeitwerte aufzubereiten, um sie in bestimmten Darstellungsformaten auszugeben. Auch die Vorgehensweise zur Vorgabe von Zeichenmustern für das Parsen von Zeichenketten, um sie in entsprechende Datentypen umzuwandeln, konnten Sie schon in einigen Beispielen kennen lernen. Doch glauben Sie mir: Sie haben bislang nur die Spitze des Eisberges gesehen.

Die folgenden Abschnitte beschäftigen sich intensiver mit der textlichen Aufbereitung von Daten, und insbesondere der Umgang mit den verfügbaren Format Providern bis hin zur Realisierung eigener Format Provider wird für Sie sicherlich von großen Interesse sein.

## Allgemeines über Format Provider in .NET

Wenn Sie einen Zahlenwert in eine Zeichenkette umwandeln möchten, dann machen Sie das in der Regel mit der `ToString`-Funktion, da sie Ihnen die flexibelste Steuerung der Umwandlung bietet. Für numerische Werte stehen Ihnen folgende Möglichkeiten zur Verfügung:

- `DoubleVariable.ToString()`: Der Wert der Zeichenkette wird mit den aktuellen Kultureinstellungen ausgegeben, die von Ihren Betriebssystemeinstellungen abhängig sind.
- `DoubleVariable.ToString("formatzeichenfolge")`: Die Formatzeichenfolge bestimmt, wie der Zahlenwert in eine Zeichenkette umgewandelt werden soll.
- `DoubleVariable.ToString(IFormatProvider)`: Ein so genannter Format Provider (etwa: Formatierungsanbieter) bestimmt, wie die Zeichenfolge formatiert werden soll.

- `DoubleVariable.ToString("formatzeichenfolge", IFormatProvider)`: Eine Formatzeichenfolge bestimmt, wie der Zahlenwert in eine Zeichenkette umgewandelt werden soll; der zusätzlich angegebene Format Provider regelt nähere Konventionen.

Das Gleiche gilt bei der Ausgabe von Datums- bzw. Zeitwerten, wenn also der Datentyp Date in eine Zeichenkette umgewandelt werden soll.

Von der ersten Möglichkeit haben Sie sicherlich schon oft Gebrauch gemacht. Wichtig zu wissen: Die kulturabhängigen Besonderheiten werden bei der Umwandlung berücksichtigt. Wenn Sie auf einem deutschen .NET-System den Code

```
Module FormatDemos
```

```
Sub Main()  
  
    Dim locDouble As Double = 1234.56789  
    Dim locDate As Date = #12/24/2003 10:33:00 PM#  
  
    Console.WriteLine(locDouble.ToString())  
    Console.WriteLine(locDate.ToString())  
    Console.ReadLine()  
  
End Sub
```

```
End Module
```

ablaufen lassen, erhalten Sie die folgende Ausgabe:

```
1234,56789  
24.12.2003 22:33:00
```

Wenn Sie das Programm kompilieren und auf einem englischen oder amerikanischen Computer<sup>1</sup> laufen lassen (und das, ohne es neu zu kompilieren), sieht die Ausgabe schon ganz anders aus:

```
1234.56789  
12/24/2003 10:33:00 PM
```

Der Hintergrund: Auch wenn Sie keinen zusätzlichen Parameter bei der Verwendung von `ToString` angegeben haben, muss es irgendein Regelwerk in .NET geben, das bestimmt, wann ein Dezimalkomma wirklich ein Komma und wann ein Punkt ist, so wie auf englischen oder amerikanischen Systemen. Diese Bestimmungen werden von den Format Providern geregelt. Sie können auch auf einem deutschen System so tun, als würden Sie englisch formatieren.

## Kulturabhängige Formatierungen mit CultureInfo

Um länderspezifische Formatierungen bei der Umwandlung vorzunehmen, verwenden Sie einen Format Provider namens `CultureInfo`. Auf diese Klasse können Sie zugreifen, wenn Sie den Namensbereich `System.Globalization` in Ihr Programm eingebunden haben. Verändern Sie das vorherige Programm wie folgt (Änderungen sind fett hervorgehoben):

---

<sup>1</sup> Es gilt das zuvor Gesagte: Die Ländereinstellungen des Betriebssystems sind dafür verantwortlich.

```

Module FormatDemosCultureInfo

Sub Main()

    Dim locDate As Date = #12/24/2003 10:33:00 PM#
    Dim locDouble As Double = 1234.56789
    Dim locCultureInfo As New CultureInfo("en-US")

    Console.WriteLine(locDouble.ToString(locCultureInfo))
    Console.WriteLine(locDate.ToString(locCultureInfo))
    Console.ReadLine()

End Sub

```

End Module

Wenn Sie dieses Programm starten, werden Sie sehen, dass die Ausgabe exakt dem entspricht, was auf dem US-Computer bei der Verwendung des vorherigen Programms ausgegeben wurde.

Wenn Sie keinen Format Provider bei der `ToString`-Methode eines primitiven Datentyps angeben, verwendet .NET automatisch den, der der eingestellten Kultur entspricht. Sie können den Format Provider der aktuellen Kultureinstellung mit der statischen Funktion `CurrentCulture` der `CultureInfo`-Klasse ermitteln. Wenn Sie die beiden Zeilen

```

Console.WriteLine(locDouble.ToString(locCultureInfo))
Console.WriteLine(locDate.ToString(locCultureInfo))

```

gegen die Zeilen

```

Console.WriteLine(locDouble.ToString(CultureInfo.CurrentCulture))
Console.WriteLine(locDate.ToString(CultureInfo.CurrentCulture))

```

austauschen und wieder einmal auf einem deutschen und einmal auf einem amerikanischen System laufen lassen, bekommen Sie exakt das Ergebnis, das Sie auch beim Beispiel beobachten konnten, bei dem `ToString` gar kein Parameter übergeben wurde.

Wie Sie im vorherigen Beispiel sehen konnten, erstellen Sie eine neue `CultureInfo`-Klasseninstanz, indem Sie eine Buchstabenkombination angeben, die die entsprechende Kultur bezeichnet. Die folgende Tabelle zeigt Ihnen die wichtigsten Einstellungen. Eine vollständige Tabelle können Sie aus der Visual Studio-Online-Hilfe erhalten.

Kulturkürzel	Kulturcode	Sprache-Land/Region
""(leere Zeichenfolge)	0x007F	Invariante Kultur
NL	0x0013	Niederländisch
nl-BE	0x0813	Niederländisch – Belgien
nl-NL	0x0413	Niederländisch – Niederlande
En	0x0009	Englisch
en-AU	0x0C09	Englisch – Australien
en-CB	0x2409	Englisch – Karibik
<b>en-IE</b>	<b>0x1809</b>	<b>Englisch – Irland</b> ►

Kulturkürzel	Kulturcode	Sprache-Land/Region
en-GB	0x0809	Englisch – Großbritannien
en-US	0x0409	Englisch – USA
Fr	0x000C	Französisch
fr-BE	0x080C	Französisch – Belgien
fr-CA	0x0C0C	Französisch – Kanada
fr-FR	0x040C	Französisch – Frankreich
fr-LU	0x140C	Französisch – Luxemburg
fr-MC	0x180C	Französisch – Monaco
fr-CH	0x100C	Französisch – Schweiz
de	0x0007	Deutsch
de-AT	0x0C07	Deutsch – Österreich
de-DE	0x0407	Deutsch – Deutschland
de-LI	0x1407	Deutsch – Liechtenstein
de-LU	0x1007	Deutsch – Luxemburg
de-CH	0x0807	Deutsch – Schweiz
it	0x0010	Italienisch
it-IT	0x0410	Italienisch – Italien
it-CH	0x0810	Italienisch – Schweiz
es	0x000A	Spanisch
es-AR	0x2C0A	Spanisch – Argentinien
es-PR	0x500A	Spanisch – Puerto Rico
es-ES	0x0C0A	Spanisch – Spanien
es-VE	0x200A	Spanisch – Venezuela

**Tabelle 17.1:** Diese Einstellungen verwenden Sie für *CultureInfo*-Objekte

## Vermeiden von kulturabhängigen Programmfehlern

Auf Programmentwicklungen hat das unter Umständen Auswirkungen, nämlich dann, wenn Sie numerische Daten oder Zeitdaten in Textdateien speichern und diese kulturübergreifend ausgetauscht werden müssen. Und: Das Definieren von konstanten Werten sollte im Programm nur direkt mit Literalen und nie durch Zeichenketten und entsprechende Konvertierungsfunktionen erfolgen. Dazu ein Beispiel:

Die Codezeilen

```
Dim locDoubleTest As Double = CDbl("1.234")
Console.WriteLine(locDoubleTest)
```

ergeben auf einem deutschen System den Wert 1234; auf einem englischsprachigen System jedoch den Wert 1,234 (aber 1.234 angezeigt), da hier der Punkt nicht als Tausendergruppierung, sondern eben als Dezimalkomma (*Decimal Point*) angesehen wird. Mehr Überlegungen zu diesem Thema finden Sie in auch im vorherigen Kapitel. Das dort Gesagte gilt gleichermaßen auch für Datumswerte. Die Umwandlung aus einer Zeichenfolge wirkt dabei noch konstruiert, aber denken Sie daran, wie oft Eingaben aus einer TextBox mit TextBox.Text (eine Methode, die einen String zurückgibt) geparst werden müssen, auch und gerade für numerische Werte.

Wenn Sie Formatierungen für das Serialisieren (Speichern) vornehmen müssen, verwenden Sie dazu am besten ein so genanntes *invariantes CultureInfo-Objekt*, das Sie mit folgender Anweisung ermitteln können:

```
Dim locCultureInfo As CultureInfo = CultureInfo.InvariantCulture
```

Wenn Sie alle Konvertierungen innerhalb Ihres Programms (also sowohl das Parsen eines Strings, um ihn in einen entsprechenden Datentyp umzuwandeln, als auch das Konvertieren eines Datentyps in einen String) damit durchführen, sind Sie auf der sicheren Seite.

## Formatierung durch Formatzeichenfolgen

Sowohl die `ToString`-Methode als auch die `Parse`- bzw. die `ParseExact`-Methode unterstützen neben dem Einsatz von Format Providern auch die Formatierung durch direkte Mustervorlagen, die durch Strings oder String-Arrays übergeben werden. Sie haben schon an einigen Beispielen gesehen, dass bestimmte Buchstaben, zu Gruppen zusammengeführt, bestimmte Formatierungen eines Datenblocks (Tage, Monate oder Jahre beispielsweise) bewirken. Die folgenden Abschnitte demonstrieren den Einsatz von Formatzeichenfolgen.

### Formatierung von numerischen Ausdrücken durch Formatzeichenfolgen

---

**BEGLEITDATEIEN:** Sie finden im Verzeichnis `.\VB 2005 - Entwicklerbuch\E - Datentypen\Kap17\FormatProvider - Num` die Beispiele, die in diesem Abschnitt behandelt werden.

---

Wenn Sie die Codezeilen,

```
Dim locDouble As Double  
locDouble = Math.PI + 10000 * Math.PI  
Console.WriteLine("locDouble hat den Wert:" + locDouble.ToString())
```

ausführen, erhalten Sie die Ausgabe

```
locDouble hat den Wert: 31419,0681285515
```

In vielen Fällen ist es aber erwünscht, beispielsweise den Tausendergruppierungspunkt ebenfalls anzeigen zu lassen oder nur eine bestimmte Anzahl von Nachkommastellen auszugeben. Dann können Sie die Ausgabe durch Formatzeichenfolgen reglementieren.

Wenn Sie beispielsweise die Zeile

```
Console.WriteLine("locDouble hat den Wert:" + locDouble.ToString())
```

des obigen Beispiels durch die folgende ersetzen,

```
Console.WriteLine("locDouble hat den Wert: " + locDouble.ToString("#,###.00"))
```

sieht die Ausgabe schon sehr viel ordentlicher aus:

```
locDouble hat den Wert: 31.419,07
```

In diesem Fall ist der übergebene Parameter als Regelvorgabe für die Formatierung übergeben worden. Jedes »#«-Zeichen stellt dabei eine mögliche Ziffer dar, jede 0 eine Muss-Ziffer. Das bedeutet: Die Zahl 304 würde keine zusätzliche führende 0 bei der Ausgabe bekommen, da die Formatzeichenfolge zwar vier Zeichen (»#,###«) vorgibt, durch die Verwendung des »#«-Zeichens aber keine zwingende Verwendung aller Ziffern vorgeschrieben wird.

Anders sieht es bei dem gleichen Wert und seinen Nachkommastellen nach der Formatierung aus: Da hier »0« und nicht »#« angegeben wird, lauten die Nachkommastellen nach der Formatierung ».00«, obwohl zur Wertdarstellung eigentlich keine Nachkommastellen nötig wären.

Die folgende Tabelle zeigt, welche Formatzeichen Sie bei der Zusammenstellung von Formatzeichenfolgen für numerische Werte verwenden können:

Formatzeichen	Name	Beschreibung
0	0-Platzhalter	<p>Gibt es an der Stelle des 0-Platzhalters eine korrelierende Ziffer im aufzubereitenden Wert, dann wird die Ziffer des Wertes an der Stelle platziert, die der 0-Platzhalter durch seine Position vorgibt.</p> <p>Wenn der zu formatierende Wert die vorgegebene Ziffernposition nicht hergibt (bei »000,00« als Vorgabe gäbe es für den Wert 34,1 nicht alle vorgegebenen Position), wird eine 0 angezeigt.</p> <p>Die Positionen der »0«, die am weitesten links vor dem Dezimaltrennzeichen steht, und der »0«, die am weitesten rechts hinter dem Dezimaltrennzeichen steht, bestimmen also den Bereich der Ziffern, die immer in der Ergebniszeichenfolge enthalten sind.</p> <p>Falls im zu formatierenden Wert mehr Nachkommastellen vorhanden sind, als Nullen Positionen vorgeben, wird auf die entsprechende Anzahl auf Stellen gerundet.</p>
#	Ziffernplatzhalter	<p>Verfügt der zu formatierende Wert über eine Ziffer an der Stelle, an der »#« in der Formatzeichenfolge steht, wird diese Ziffer in der Ergebniszeichenfolge angezeigt, anderenfalls nicht.</p> <p>Beachten Sie, dass dieses Formatzeichen nie die Anzeige von 0 bewirkt, wenn es sich nicht um eine signifikante Ziffer handelt, selbst wenn 0 die einzige Ziffer in der Zeichenfolge ist. 0 wird jedoch angezeigt, wenn es sich um eine signifikante Ziffer in der angezeigten Zahl handelt (bei 0304,030 beispielsweise sind die äußeren Nullen nicht signifikant, d. h. Weglassen verändert den Wert nicht).</p>
.	Dezimaltrennzeichen	<p>Das erste ».-«-Zeichen in der Formatzeichenfolge bestimmt die Position des Dezimaltrennzeichens im formatierten Wert. Weitere ».-«-Zeichen werden ignoriert. Das für das Dezimaltrennzeichen verwendete Zeichen wird durch die <i>NumberDecimalSeparator</i>-Eigenschaft der <i>NumberFormatInfo</i> bestimmt, die die Formatierung steuert, allerdings nur für die Ausgabe, nie für die Vorgabe! <b>WICHTIG:</b> Verwechseln Sie ».,« und »,.« nicht bei der Erstellung der Formatzeichenfolge. Die amerikanisch/englische Formatvorgabe ist hier maßgeblich – beispielsweise »1,000,000.23« für Einemillionenkommazweidrei.</p>

Formatzeichen	Name	Beschreibung
,	Tausendertrennzeichen und Zahlenskalierung	Das »,«-Zeichen erfüllt zwei Aufgaben. Erstens: Wenn die Formatzeichenfolge ein »,«-Zeichen zwischen zwei Ziffernplätzen enthält (»0« oder »#«) und einer der Platzhalter links neben dem Dezimaltrennzeichen steht, werden in der Ausgabe Tausendertrennzeichen zwischen jeder Gruppe von drei Ziffern links neben dem Dezimaltrennzeichen eingefügt. Das in der Ergebniszeichenfolge als Dezimaltrennzeichen verwendete Zeichen wird durch die <i>NumberGroupSeparator</i> -Eigenschaft der aktuellen <i>NumberFormatInfo</i> bestimmt, die die Formatierung steuert. Zweitens: Wenn die Formatzeichenfolge ein oder mehr »,«-Zeichen direkt links neben dem Dezimaltrennzeichen enthält, wird die Zahl durch die Anzahl der »,«-Zeichen multipliziert mit 1000 dividiert, bevor sie formatiert wird. Die Formatzeichenfolge »0,« stellt 100 Millionen z. B. als 100 dar. Verwenden Sie das »,«-Zeichen, um anzugeben, dass die Skalierung keine Tausendertrennzeichen in der formatierten Zahl enthält. Um also eine Zahl um 1 Million zu skalieren und Tausendertrennzeichen einzufügen, verwenden Sie die Formatzeichenfolge »#,##0,«. <b>WICHTIG:</b> Verwechseln Sie »« und »..« nicht bei der Erstellung der Formatzeichenfolge. Die amerikanisch/englische Formatvorgabe ist hier maßgeblich – beispielsweise »1,000,000.23« für <i>Einemillionkommazweidrei</i> .
%	Prozentplatzhalter	Enthält eine Formatzeichenfolge ein »%«-Zeichen, wird die Zahl vor dem Formatieren mit 100 multipliziert. Das entsprechende Symbol wird in der Zahl an der Stelle eingefügt, an der »%« in der Formatzeichenfolge steht. Das verwendete Prozentzeichen ist von der aktuellen <i>NumberFormatInfo</i> -Klasse abhängig.
E0 E+0 E-0 e0 e+0 e-0	Wissenschaftliche Notation	Enthält die Formatzeichenfolge die Zeichenfolgen »E«, »E+«, »E-«, »e«, »e+« oder »e-« und folgt direkt danach mindestens ein »0«-Zeichen, wird die Zahl mit der wissenschaftlichen Notation formatiert und ein »E« bzw. »e« zwischen der Zahl und dem Exponenten eingefügt. Die Anzahl der »0«-Zeichen nach dem entsprechenden Formatzeichen für die wissenschaftliche Notation bestimmt die Mindestanzahl von Ziffern, die für den Exponenten ausgegeben werden. Das »E+«-Format und das »e+«-Format geben an, dass immer ein Vorzeichen (Plus oder Minus) vor dem Exponenten steht. Die Formate »E«, »E-«, »e« oder »e-« geben an, dass nur vor negativen Exponenten ein Vorzeichen steht.
'ABC' "ABC"	Zeichenfolgenliteral	Zeichen, die in einfachen bzw. doppelten Anführungszeichen stehen, werden direkt in die Ergebniszeichenfolge kopiert, ohne die Formatierung zu beeinflussen.
:	Abschnittstrennzeichen	Mit dem »;«-Zeichen werden Abschnitte für positive und negative Zahlen sowie Nullen in der Formatzeichenfolge voneinander getrennt. Lesen Sie dazu auch die Anmerkung am Ende der Tabelle.
Sonstige	Alle anderen Zeichen	Alle anderen Zeichen werden als Literale an der angegebenen Position in die Ergebniszeichenfolge kopiert.

**Tabelle 17.2:** Für numerische Formatzeichenfolgen verwenden Sie diese Formatzeichen

**HINWEIS:** Bitte beachten Sie, dass Sie für negative und positive Werte sowie für den Wert 0 jeweils eine individuelle Zeichenfolge bestimmen können, die durch das Semikolon getrennt werden. Das folgende Beispiel verdeutlicht ihre Anwendung:

```

Dim locDouble As Double
Dim locFormat As String = "#,###.00;-#,###.0000;+-0.00000"
locDouble = Math.PI + 10000 * Math.PI
Console.WriteLine("locDouble hat den Wert: " + locDouble.ToString(locFormat))
locDouble *= -1
Console.WriteLine("locDouble hat den Wert: " + locDouble.ToString(locFormat))
locDouble = 0
Console.WriteLine("locDouble hat den Wert: " + locDouble.ToString(locFormat))

```

Dieser Codeausschnitt würde das folgende Ergebnis auf dem Bildschirm produzieren:

```

locDouble hat den Wert: 31.419,07
locDouble hat den Wert: -31.419,0681
locDouble hat den Wert: +-0,00000

```

## Formatierung von numerischen Ausdrücken durch vereinfachte Formatzeichenfolgen

Für den einfachen Einsatz gibt es in .NET einige vereinfachte Formatzeichenfolgen, die in der Regel nur aus einem einzigen Zeichen bestehen und einen Typ von Formatierung bezeichnen. Anstatt z.B. eine Formatzeichenfolge zu erstellen, die eine Währungsformatierung vorgibt, können Sie einfach das »C«-Zeichen als Formatzeichenfolge verwenden. In Kombination mit dem entsprechenden *CultureInfo*-Format-Provider brauchen Sie sich obendrein noch nicht einmal um das Finden des entsprechenden Währungssymbols zu kümmern:

```

Dim locDouble As Double

locDouble = 12234.346
Console.WriteLine("Sie bekommen {0} aus einem Lottogewinn.", locDouble.ToString("#,###.00"))
Console.WriteLine("Sie bekommen {0} aus einem Lottogewinn.", locDouble.ToString("C", New CultureInfo("en-US")))

```

Beide `Console.WriteLine`-Anweisungen zeigen in diesem Beispiel das exakt gleiche Ergebnis im Konsolenfenster an, nämlich:<sup>2</sup>

```

Sie bekommen $12.234,35 aus einem Lottogewinn.
Sie bekommen $12,234.35 aus einem Lottogewinn.

```

Die folgende Tabelle zeigt Ihnen, welche Kurzformen für Formatzeichenfolgen .NET für die Formatierung von numerischen Werten kennt. Bitte beachten Sie, dass, wie im Beispiel gezeigt, die Resultate unter Umständen kulturabhängig sind und sich deswegen mit einem *CultureInfo*-Objekt entsprechend steuern lassen.

Formatzeichen	Beschreibung
c, C	Währungsformat.
d, D	Dezimales Format.

<sup>2</sup> Ich habe bei Währungsangaben bewusst auf die Euro-Währung verzichtet, da Konsolenanwendungen das Euro-Zeichen nicht ohne weiteres darstellen können.

Formatzeichen	Beschreibung
e, E	Wissenschaftliches Format (Exponentialformat).
f, F	Festkommaformat.
g, G	Allgemeines Format.
n, N	Zahlenformat.
r, R	Schleifenformat, das sicherstellt, dass in Zeichenfolgen konvertierte Zahlen denselben Wert haben, wenn sie wieder in Zahlen konvertiert werden.
x, X	Hexadezimales Format.

**Tabelle 17.3:** Für vereinfachte numerische Formatzeichenfolgen verwenden Sie diese Formatzeichen

## Formatierung von Datums- und Zeitausdrücken durch Formatzeichenfolgen

---

**BEGLEITDATEIEN:** Sie finden im Verzeichnis `.\VB 2005 - Entwicklerbuch\E - Datentypen\Kap17\FormatProvider - Date\` die Beispiele, die in diesem Abschnitt behandelt werden.

---

Was für numerische Datentypen gilt, ist auch auf den Datumstyp `Date` anwendbar: Es gibt eine umfangreiche Unterstützung durch bestimmte Formatzeichenfolgen, die bestimmen, wie ein Datumswert in eine Zeichenkette umgewandelt werden kann.

Wenn Sie die Codezeilen,

```
Dim locDate As Date
locDate = #12/24/2003 10:33:00 PM#
Console.WriteLine("locDate hat den Wert: " + locDate.ToString())
```

ausführen, erhalten Sie die Ausgabe

```
locDate hat den Wert: 24.12.2003 22:33:00
```

In den meisten Fällen benötigen Sie aber keine kombinierte Ausgabe des Zeitwertes von Uhrzeit und Datum sondern nur eine von beiden. Außerdem verlangen viele Anwendungen, das Datum etwas ausführlicher darzustellen – beispielsweise indem die ersten Buchstaben des Monats dargestellt werden oder der Wochentag des Datums mit ausgegeben wird.

Wenn Sie beispielsweise die Zeile

```
Console.WriteLine("locDate hat den Wert: " + locDate.ToString())
```

des obigen Beispiels durch die folgende ersetzen,

```
Console.WriteLine("locDate hat den Wert: " + locDate.ToString("ddd, dd. MMM yyyy - HH:mm"))
```

erkennen Sie, dass Sie durch den Einsatz einer Datums-Formatzeichenfolgen sehr viel mehr Einfluss auf die Art der Umwandlung des Datumswertes in eine Zeichenkette genommen haben, denn die Ausgabe sieht nun folgendermaßen aus:

```
locDate hat den Wert: Mittwoch, 24. Dez 2003 - 22:33
```

In diesem Fall haben Sie der `ToString`-Methode eine Zeichenkette mit Formatanweisungen für die Formatierung übergeben. Jedes der verwendeten Zeichen der Formatzeichenfolge spiegelt dabei eine Datengruppe innerhalb eines `Date`-Datentyps wider.

Die folgende Tabelle zeigt, welche Formatzeichen Sie bei der Zusammenstellung von Formatzeichenfolgen für numerische Werte verwenden können.

---

**WICHTIG:** Einige Formatzeichen der folgenden Tabelle können auch den Kurzformatzeichen der übernächsten Tabelle entsprechen. Achten Sie deswegen darauf, dass Sie die folgenden Zeichen im beschriebenen Kontext nicht alleine verwenden. Und: Je nach zusätzlich verwendetem `CultureInfo` produzieren die verwendeten Formatzeichenkombinationen unterschiedliche Resultate.

---

Formatzeichen	Beschreibung
d	Zeigt den aktuellen Tag des Monats als Zahl zwischen 1 und 31 an. Wenn die Nummer des Tages nur einstellig ist, wird diese nicht mit einer führenden Null versehen.
dd	Zeigt den aktuellen Tag des Monats als Zahl zwischen 1 und 31 an. Wenn die Nummer des Tages nur einstellig ist, wird ihr eine 0 vorangestellt.
ddd	Zeigt den abgekürzten Namen des Tages für den angegebenen <code>Date</code> -Wert an.
ddd ddd	Zeigt den vollständigen Namen des Tages für den angegebenen <code>Date</code> -Wert an.
f	Zeigt die Bruchteile von Sekunden als eine Ziffer an.
ff	Zeigt die Bruchteile von Sekunden als zwei Ziffern an.
fff	Zeigt die Bruchteile von Sekunden als drei Ziffern an.
ffff	Zeigt die Bruchteile von Sekunden als vier Ziffern an.
Fffff	Zeigt die Bruchteile von Sekunden als fünf Ziffern an.
ffffff	Zeigt die Bruchteile von Sekunden als sechs Ziffern an.
fffffff	Zeigt die Bruchteile von Sekunden als sieben Ziffern an.
g oder gg	Zeigt den Zeitraum für den angegebenen <code>Date</code> -Wert an. Wenn Sie mit einer deutschen Kultureinstellung (durch <code>CultureInfo</code> bestimmt) arbeiten, ist das beispielsweise der Vermerk »n. Christ.«.
H	Zeigt die Stunde des angegebenen <code>Date</code> -Wertes im Bereich 1–12 an. Die Stunde stellt die ganzen Stunden dar, die seit Mitternacht (als 12 dargestellt) oder Mittag (ebenfalls als 12 dargestellt) vergangen sind. Wenn dieses Format einzeln verwendet wird, können die jeweils gleichen Stunden vor und nach Mittag nicht unterschieden werden. Wenn die Stundenzahl nur einstellig ist, wird diese auch als einzelne Ziffer angezeigt. <b>Wichtig:</b> Für das 24-Stunden-Format wählen Sie das große »H«.
Hh	Zeigt die Stunde des angegebenen <code>Date</code> -Wertes im Bereich 1–12 an. Die Stunde stellt die ganzen Stunden dar, die seit Mitternacht (als 12 dargestellt) oder Mittag (ebenfalls als 12 dargestellt) vergangen sind. Wenn dieses Format einzeln verwendet wird, können die jeweils gleichen Stunden vor und nach Mittag nicht unterschieden werden. Wenn die Stundenzahl nur einstellig ist, wird eine führende Null vorangestellt. <b>Wichtig:</b> Für das 24-Stunden-Format wählen Sie das große »H«.
H	Zeigt die Stunde des angegebenen <code>Date</code> -Wertes im Bereich 0–23 an. Die Stunde stellt die seit Mitternacht (als 0 angezeigt) vergangenen Stunden dar. Wenn die Stundenzahl nur einstellig ist, wird sie auch nur als einzelne Ziffer angezeigt.

Formatzeichen	Beschreibung
HH	Zeigt die Stunde des angegebenen <i>Date</i> -Wertes im Bereich 0–23 an. Die Stunde stellt die seit Mitternacht (als 0 angezeigt) vergangenen Stunden dar. Wenn die Stundenzahl einstellig ist, wird dieser Ziffer eine führende 0 vorangestellt.
m	Zeigt die Minute des angegebenen <i>Date</i> -Wertes im Bereich 0–59 an. Die Minute stellt die seit der letzten Stunde vergangenen ganzen Minuten dar. Wenn die Minute nur einstellig ist, wird diese auch nur als einzelne Ziffer angezeigt.
mm	Zeigt die Minute des angegebenen <i>Date</i> -Wertes im Bereich 0–59 an. Die Minute stellt die seit der letzten Stunde vergangenen ganzen Minuten dar. Wenn die Minute nur eine einzelne Ziffer ist (0–9), wird dieser Ziffer bei der Formatierung eine 0 (01–09) vorangestellt.
M	Zeigt den Monat als Zahl zwischen 1 und 12 an. Die Ausgabe erfolgt bei einstelligen Werten ohne Führungsnull. <b>Wichtig:</b> Denken Sie daran, hier Großbuchstaben zu verwenden, da Sie sonst eine Minutenausgabe erwirken.
MM	Zeigt den Monat als Zahl zwischen 1 und 12 an. Die Ausgabe erfolgt bei einstelligen Werten mit Führungsnull. <b>Wichtig:</b> Denken Sie daran, hier Großbuchstaben zu verwenden, da Sie sonst eine Minutenausgabe erwirken.
MMM	Zeigt den abgekürzten Namen des Monats für den angegebenen <i>Date</i> -Wert an. <b>Wichtig:</b> Denken Sie daran, hier Großbuchstaben zu verwenden, da Sie sonst eine Minutenausgabe erwirken.
MMMM	Zeigt den vollständigen Namen des Monats für den angegebenen <i>Date</i> -Wert an. <b>Wichtig:</b> Denken Sie daran, hier Großbuchstaben zu verwenden, da Sie sonst eine Minutenausgabe erwirken.
s	Zeigt die Sekunden im Bereich 0–59 an. Die Sekunde stellt die seit der letzten Minute vergangenen ganzen Sekunden dar. Die Ausgabe erfolgt bei einstelligen Werten ohne Führungsnull.
ss	Zeigt die Sekunden im Bereich 0–59 an. Die Sekunde stellt die seit der letzten Minute vergangenen ganzen Sekunden dar. Die Ausgabe erfolgt bei einstelligen Werten mit Führungsnull.
t	Zeigt das erste Zeichen des A.M./P.M.-Bezeichners für den angegebenen <i>Date</i> -Wert an. <b>Wichtig:</b> Beim deutschen <i>CultureInfo</i> -Format-Provider (entspricht auf deutschen .NET-Systemen der Standardeinstellung, wenn kein Format Provider angegeben wurde), bleibt dieses Formatzeichen unberücksichtigt – wird also ignoriert.
tt	Zeigt den vollständigen A.M./P.M.-Bezeichner für den angegebenen <i>Date</i> -Wert an. <b>Wichtig:</b> Beim deutschen <i>CultureInfo</i> -Format-Provider (entspricht auf deutschen .NET-Systemen der Standardeinstellung, wenn kein Format Provider angegeben wurde), bleibt dieses Formatzeichen unberücksichtigt – wird also ignoriert.
y	Zeigt das Jahr an. Die ersten beiden Ziffern des Jahres werden ausgelassen. Die Ausgabe erfolgt bei einstelligen Werten ohne Führungsnull.
yy	Zeigt das Jahr an. Die ersten beiden Ziffern des Jahres werden ausgelassen. Die Ausgabe erfolgt bei einstelligen Werten mit Führungsnull.
yyyy	Zeigt das Jahr an. Wenn das Jahr aus weniger als vier Ziffern besteht, werden diesem Nullen vorangestellt, damit es vier Ziffern enthält.
z	Zeigt das Offset der aktuellen Zeitzone des Systems in ganzen Stunden an. Das Offset wird immer mit einem vorangestellten Plus- bzw. Minuszeichen angezeigt (Null wird als »+0« angezeigt), das die Stunden vor (+) GMT (Greenwich Mean Time) bzw. nach (-) GMT angibt. Der Wertebereich liegt zwischen -12 und +13 Stunden. Ist das Offset einstellig, erfolgt die Anzeige ebenfalls nur einstellig; es werden keine führenden Nullen vorangestellt. Die Einstellung für die Zeitzone wird als +X oder -X angegeben, wobei X der Offset zur GMT in Stunden ist. Das angezeigte Offset kann durch die Sommer-/Winterzeitumstellungen beeinflusst werden.
zz	Wie vorher; das Offset wird bei einstelligen Werten jedoch mit führender Null angezeigt. ►

Formatzeichen	Beschreibung
zzz	Wie vorher; das Offset wird aber in Stunden und Minuten angezeigt.
:	Trennzeichen für Zeitangaben.
/	Trennzeichen für Datumsangaben. <b>WICHTIG:</b> Wenn Sie für das Parsen eines Datumswertes einen Schrägstrich verlangen, müssen Sie »\« verwenden, um den Schrägstrich als Gruppentrennzeichen zu bestimmen.
"	Zeichenfolge in Anführungszeichen. Zeigt den literalen Wert einer Zeichenfolge zwischen zwei Anführungszeichen an, denen ein Escapezeichen (/) vorangestellt ist.
'	Zeichenfolge in Anführungszeichen. Zeigt den literalen Wert einer Zeichenfolge zwischen zwei »'«-Zeichen an.
Jedes andere Zeichen	Andere Zeichen werden als Literale direkt in die Ergebniszeichenfolge geschrieben.

**Tabelle 17.4:** Für Formatzeichenfolgen zur Aufbereitung von Datums- und Zeitwerten verwenden Sie diese Formatzeichen

Die folgenden Beispiele sollen den Umgang mit den Formatzeichenfolgen verdeutlichen:

```
Module FormatDemosFormatzeichenfolgen
Sub Main()
    Dim locDate As Date = #12/24/2005 10:32:22 PM#
    Dim locFormat As String

    'Normale Datumsausgabe; große Ms ergeben den Monat.
    locFormat = "dd.MM.yyyy"
    Console.WriteLine("'{0}' : {1}", _
                      locFormat, _
                      locDate.ToString(locFormat, CultureInfo.CurrentCulture))

    'Falsche Ausgabe; anstelle des Monats werden Minuten ausgegeben.
    locFormat = "dd.mm.yyyy"
    Console.WriteLine("'{0}' : {1}", _
                      locFormat, _
                      locDate.ToString(locFormat, CultureInfo.CurrentCulture))

    'Komplette Ausgabe, ausführlicher geht's nicht.
    locFormat = "dddd, dd.MMMM.yyyy - HH:mm:ss:fffffff ""(Offset:\"" zzz)""
    Console.WriteLine("'{0}' : {1}", _
                      locFormat, _
                      locDate.ToString(locFormat, CultureInfo.CurrentCulture))

    'Falsche Ausgabe; der Text "Uhrzeit" steht nicht in Anführungszeichen.
    locFormat = "Uhrzeit: HH:mm:ss"
    Console.WriteLine("'{0}' : {1}", _
                      locFormat, _
                      locDate.ToString(locFormat, CultureInfo.CurrentCulture))

    'So geht es richtig:
    locFormat = """Uhrzeit:"" HH:mm:ss"
    Console.WriteLine("'{0}' : {1}", _
                      locFormat, _
                      locDate.ToString(locFormat, CultureInfo.CurrentCulture))
```

```

'PM-Anzeige funktioniert nicht bei deutscher...
locFormat = """Uhrzeit:"" hh:mm:ss tt"
Console.WriteLine("'{0}' : {1}",
                  locFormat,
                  locDate.ToString(locFormat, New CultureInfo("de-DE")))

'....aber beispielsweise bei amerikanischer Kultureinstellung
locFormat = """Uhrzeit:"" hh:mm:ss tt"
Console.WriteLine("'{0}' : {1}",
                  locFormat,
                  locDate.ToString(locFormat, New CultureInfo("en-US")))

'Englisches Datumsformat trotz deutscher Kultureinstellung
locFormat = """Date:"" MM\dd\yyyy"
Console.WriteLine("'{0}' : {1}",
                  locFormat,
                  locDate.ToString(locFormat, New CultureInfo("de-DE")))

'Backslash davor nicht vergessen, sonst:
locFormat = """Date:"" MM/dd/yyyy"
Console.WriteLine("'{0}' : {1}",
                  locFormat,
                  locDate.ToString(locFormat, New CultureInfo("de-DE")))

'Aber nur so mit englischen Texten:
locFormat = """Date:"" MMMM, ""the"" dd. yyyy"
Console.WriteLine("'{0}' : {1}",
                  locFormat,
                  locDate.ToString(locFormat, New CultureInfo("en-US")))

Console.ReadLine()
End Sub
End Module

```

Lassen Sie dieses Programm laufen, ergibt sich folgende Ausgabe im Konsolenfenster:

```

'dd.MM.yyyy' : 24.12.2005
'dd.mm.yyyy' : 24.32.2005
'dddd, dd.MMMM.yyyy - HH:mm:ss:fffffff "(Offset: " zzz)"' : Samstag, 24.Dezember.2005 - 22:32:22:0000000
(Offset: +01:00)
'Uhrzeit: HH:mm:ss' : U10r+lei: 22:32:22
'"Uhrzeit:" HH:mm:ss' : Uhrzeit: 22:32:22
'"Uhrzeit:" hh:mm:ss tt' : Uhrzeit: 10:32:22
'"Uhrzeit:" hh:mm:ss tt' : Uhrzeit: 10:32:22 PM
'"Date:" MM\dd\yyyy' : Date: 12/24/2005
'"Date:" MM/dd/yyyy' : Date: 12.24.2005
'"Date:" MMMM, "the" dd. yyyy' : Date: December, the 24. 2005

```

# Formatierung von Zeitausdrücken durch vereinfachte Formatzeichenfolgen

Für die bequeme Realisierung von Zeit- und Datumsformatierungen gibt es in .NET einige vereinfachte Formatzeichenfolgen, die in der Regel nur aus einem einzigen Zeichen bestehen und einen Formatierungsstil bezeichnen. Anstatt beispielsweise gezielt die Formatzeichenkombinationen zusammenzustellen, die ein Datum in Langform formatieren, reicht das Zurückgreifen auf ein bestimmtes vereinfachtes Formatzeichen. In Kombination mit dem entsprechenden `CultureInfo`-Format-Provider brauchen Sie sich obendrein noch nicht einmal um die in der Kultur übliche Darstellung zu kümmern:

```
Sub main()
    Dim locDate As Date = #12/24/2005 10:32:22 PM#
    Dim locFormat As String

    locFormat = "dddd, dd. MMMM yyyy"
    Console.WriteLine("Datumsformatierung mit Formatzeichen:" + locDate.ToString(locFormat))
    Console.WriteLine("...und mit vereinfachten Formatzeichen:" + locDate.ToString("D"))
    Console.ReadLine()
End Sub
```

Die folgende Tabelle zeigt, welche vereinfachten Formatzeichenfolgen Sie verwenden können, um Datums- und Zeitformatierungen durchzuführen. Viele der vereinfachten Formatzeichen haben äquivalente Formatzeichenfolgen, die mit entsprechenden (in der Tabelle angegebenen) Eigenschaften mithilfe eines `DateTimeFormatInfo`-Objektes ermittelt werden können.

Formatzeichen	Zugeordnete Eigenschaft/Beschreibung
d	Kurzform des Datums. Entspricht der Formatzeichenfolge, die die Eigenschaft <code>ShortDatePattern</code> zurückliefert.
D	Langform des Datums. Entspricht der Formatzeichenfolge, die die Eigenschaft <code>LongDatePattern</code> zurückliefert.
f	Vollständiges Datum und Uhrzeit (Langform des Datums und 24-Stunden-Zeitformat).
F	Langform des Datums und der Uhrzeit. Entspricht der Formatzeichenfolge, die die Eigenschaft <code>FullDateTimePattern</code> zurückliefert.
g	Allgemein (Kurzform des Datums sowie 24-Stunden-Zeitformat).
G	Allgemein (Kurzform des Datums und Langform der Zeit).
m, M	Ergibt den Tag und den ausgeschriebenen Monatsnamen (beispielsweise 24. Dezember). Entspricht der Formatzeichenfolge, die die Eigenschaft <code>MonthDayPattern</code> zurückliefert.
r, R	Ergibt Datum und Zeit in der so genannten RFC1123-Norm. Dabei hat die Zeiteingabe das Format beispielsweise von » Sat, 24 Dec 2005 22:32:22 GMT«. Entspricht der Formatzeichenfolge, die die Eigenschaft <code>RFC1123Pattern</code> zurückliefert. Diese Form wird bei der Kommunikation über das Internet verwendet, beispielsweise bei der <code>Header</code> -Dokumentierung von Mail-Dateien.
s	Ergibt ein auf der Grundlage von ISO 8601 unter Verwendung der Ortszeit formatiertes sortierbares Datum (beispielsweise » 2005-12-24T22:32:22 «). Entspricht der Formatzeichenfolge, die die Eigenschaft <code>SortableDateTimePattern</code> zurückliefert.
t	Kurzform der Zeit. Entspricht der Formatzeichenfolge, die die Eigenschaft <code>ShortTimePattern</code> zurückliefert.
T	Langform der Zeit. Entspricht der Formatzeichenfolge, die die Eigenschaft <code>LongTimePattern</code> zurückliefert. ►

Formatzeichen	Zugeordnete Eigenschaft/Beschreibung
u	Ergibt ein unter Verwendung des Formats zur Anzeige der koordinierten Weltzeit formatiertes sortierbares Datum (beispielsweise »2005-12-24 22:32:22Z«). Entspricht der Formatzeichenfolge, die die Eigenschaft <i>UniversalSortableDateTimePattern</i> zurückliefert.
U	Vollständiges Datum und Uhrzeit (langes Datumsformat und langes Zeitformat) unter Verwendung der koordinierten Weltzeit.
y, Y	Monat und Jahreszahl (etwa »Dezember 2005«). Entspricht der Formatzeichenfolge, die die Eigenschaft <i>YearMonthPattern</i> zurückliefert.

**Tabelle 17.5:** Für vereinfachte Formatzeichen zur Aufbereitung von Datums- und Zeitwerten verwenden Sie diese Tabelle

## Gezielte Formatierungen mit Format Providern

Das Ziel bei der Implementierung von Format Providern in .NET war es, dem Entwickler ein möglichst universelles Werkzeug zum Aufbereiten von Daten zum Zweck der übersichtlichen Ausgabe auf Bildschirmen oder in Dateien zu bieten. Gleichzeitig sollten kulturabhängige Formatierungseigenschaften berücksichtigt werden.

Letzteres haben Sie bereits in Form des *CultureInfo*-Objektes kennen gelernt. Wenn Sie der *ToString*-Funktion eines primitiven Datentyps ein *CultureInfo*-Objekt übergeben, das unter Angabe eines bestimmten Kulturkennzeichens instanziert wurde, passt sich die Ausgabe an die in der Kultur übliche Datendarstellungsweise an.

Neben dem *CultureInfo*-Objekt kennt das Framework zwei weitere Format Provider, die die Formatierung von numerischen Werten (*NumberFormatInfo*) und Datums- und Zeitwerten (*DateTimeFormatInfo*) genauer spezifizieren. Diese Format Provider arbeiten in Zusammenarbeit mit den vereinfachten Formatzeichen.

## Gezielte Formatierungen von Zahlenwerten mit NumberFormatInfo

Angenommen, Sie möchten eine Reihe von Zahlen als Währung formatiert untereinander ausgeben. Sie möchten dabei, dass die Ziffern der einzelnen Zahlen zwar als Tausender gruppiert werden (also 1.000.000 und nicht 1000000), aber Sie möchten nicht den Punkt, sondern das Leerzeichen als Gruppierungstrennzeichen verwenden. In diesem Fall verwenden Sie eine *NumberFormatInfo*-Instanz, um die Formatierungsregel genauer zu spezifizieren:

```
Sub main()
    'Den zu verwendenden Wert definieren.
    Dim locDouble As Double = 1234567.23
    'Kultureinstellungen sind englisch/britisches.
    Dim locCultureInfo As New CultureInfo("en-GB")
    'Die Einstellungen, die CultureInfo auf Grund der Kultur schon
    'geleistet hat, übernehmen wir in ein NumberFormatInfo...
    Dim locNumFormatInfo As NumberFormatInfo = locCultureInfo.NumberFormat

    '...dessen anzuwendende Regeln wir nun genauer spezifizieren können:
    locNumFormatInfo.CurrencyGroupSeparator = " "
    Console.WriteLine("Als Währung formatiert: " + locDouble.ToString("C", locNumFormatInfo))
```

```

'Auf die normale Fließkommadarstellung hat diese Einstellung keinen Einfluss:
Console.WriteLine("Als Fließkommazahl formatiert: " + locDouble.ToString("n", locNumFormatInfo))

'Jetzt schon!
locNumFormatInfo.NumberGroupSeparator = " "
Console.WriteLine("Als Fließkommazahl formatiert: " + locDouble.ToString("n", locNumFormatInfo))
End Sub

```

Wenn Sie diese kleine Prozedur laufen lassen, produziert sie folgende Ausgabe:

```

Als Währung formatiert: £1 234 567.23
Als Fließkommazahl formatiert: 1,234,567.23
Als Fließkommazahl formatiert: 1 234 567.23

```

`NumberFormatInfo` stellt Ihnen für die Spezialisierung von Formatierungen im hier gezeigten Stil eine ganze Reihe von Eigenschaften zur Verfügung, die Sie nach Belieben vor dem Einsatz in `ToString` einstellen können. Welche Eigenschaften welche Änderung bewirken, entnehmen Sie bitte der Online-Hilfe von Visual Studio.

## Gezielte Formatierungen von Zeitwerten mit `DateTimeFormatInfo`

Was für die Spezialisierung von numerischen Formatierungen gilt, ist auch für die Datums- und Zeitenformatierung gültig. Sie verwenden die `DateTimeFormatInfo`-Klasse, um Formatierungen dieses Datentyps zu spezialisieren.

Auch hier soll ein Beispiel dem besseren Verständnis dienen. Normalerweise gibt es keine AM/PM-Designatoren (Tagesbereichsbezeichner beim englisch/amerikanischem Datumsformat) für die deutschen Kultureinstellungen. Das folgende kleine Beispielprogramm richtet die Designatoren mit einem `DateTimeFormatInfo`-Objekt gezielt mit deutschen Bezeichnungen ein, und sie werden anschließend durch die Angabe der entsprechenden Formatzeichen in der Formatzeichenkette mit `ToString` bei der Umwandlung des Datums ausgegeben.

```

Sub main()
    'Zu verwendenden Wert definieren.
    Dim locDate As Date = #12/24/2005 1:12:23 PM#
    'Kultureinstellungen sind deutsch.
    Dim locCultureInfo As New CultureInfo("de-DE")
    'Die Einstellungen, die CultureInfo auf Grund der Kultur schon
    'geleistet hat, übernehmen wir in ein DateTimeFormatInfo...
    Dim locDateTimeFormatInfo As DateTimeFormatInfo = locCultureInfo.DateTimeFormat

    '...dessen anzuwendende Regeln wir nun genauer spezifizieren können:
    locDateTimeFormatInfo.AMDesignator = "Vormittag"
    locDateTimeFormatInfo.PMDesignator = "Nachmittag"
    Console.WriteLine("Mit deutschen AM/PM-Designatoren: "
        + locDate.ToString("dd.MM.yyyy hh:mm:ss - tt", locDateTimeFormatInfo))
    '12 Stunden dazu addieren:
    locDate = locDate.AddHours(12)
    Console.WriteLine("Mit deutschen AM/PM-Designatoren: "
        + locDate.ToString("dd.MM.yyyy hh:mm:ss - tt", locDateTimeFormatInfo))
End Sub

```

# Kombinierte Formatierungen

Kombinierte Formatierungen sind ein spezielles Feature von .NET, das die Einbindung von zu formatierendem bzw. umzuwandelndem Text gezielt an bestimmte Stellen innerhalb einer Zeichenkette ermöglicht. Dazu ein Beispiel:

Das folgende kleine Programm definiert einen bestimmten Ausgangszeitpunkt. Es addiert anschließend 15 Mal in Folge eine zufällige Zeitspanne zwischen 0 Minuten und 23 Stunden und 59 Minuten zum Ausgangsdatum und zeigt das entsprechende Ergebnis an. Das Programm dafür sieht folgendermaßen aus:

---

**BEGLEITDATEIEN:** Im Projekt im Verzeichnis `|VB 2005 - Entwicklerbuch|E - Datentypen\Kap17\CompositeFormatting|` finden Sie die anschließenden Beispiele.

---

```
Sub Main()
    Dim locBasisDatum As Date = #12/30/2005 1:12:32 PM#
    Dim locOffset As TimeSpan
    Dim locRandom As New Random(Now.Millisecond)
    'Backslash vor ', damit es gedruckt und nicht als Steuerzeichen interpretiert wird!
    Console.WriteLine("Es ist " +
        locBasisDatum.ToString("ddd, ""der"" d. MMMM \'yy, HH:mm") + _
        "...")
    '15 Wiederholungen
    For count As Integer = 1 To 15
        locOffset = New TimeSpan(locRandom.Next(23), locRandom.Next(59), 0)
        locBasisDatum = locBasisDatum.Add(locOffset)
        Console.WriteLine("...und " + Math.Floor(locOffset.TotalHours).ToString() + _
            " Std. und " + locOffset.Minutes.ToString() + _
            " Min. später ist " +
            locBasisDatum.ToString("ddd, ""der"" d. MMMM \'yy, HH:mm"))
    Next
End Sub
```

Es produziert etwa folgende Ausgabe:

```
Es ist Freitag, der 30. Dezember '05, 13:12...
...und 10 Std. und 21 Min. später ist Freitag, der 30. Dezember '05, 23:33
...und 2 Std. und 29 Min. später ist Samstag, der 31. Dezember '05, 02:02
...und 15 Std. und 16 Min. später ist Samstag, der 31. Dezember '05, 17:18
...und 20 Std. und 10 Min. später ist Sonntag, der 1. Januar '06, 13:28
...und 21 Std. und 43 Min. später ist Montag, der 2. Januar '06, 11:11
...und 2 Std. und 39 Min. später ist Montag, der 2. Januar '06, 13:50
...und 7 Std. und 53 Min. später ist Montag, der 2. Januar '06, 21:43
...und 10 Std. und 34 Min. später ist Dienstag, der 3. Januar '06, 08:17
...und 18 Std. und 52 Min. später ist Mittwoch, der 4. Januar '06, 03:09
...und 13 Std. und 27 Min. später ist Mittwoch, der 4. Januar '06, 16:36
...und 1 Std. und 24 Min. später ist Mittwoch, der 4. Januar '06, 18:00
...und 11 Std. und 40 Min. später ist Donnerstag, der 5. Januar '06, 05:40
...und 20 Std. und 46 Min. später ist Freitag, der 6. Januar '06, 02:26
...und 13 Std. und 31 Min. später ist Freitag, der 6. Januar '06, 15:57
...und 13 Std. und 50 Min. später ist Samstag, der 7. Januar '06, 05:47
```

Zwei Dinge fallen auf: Zum einen ist der Programmtext ein einziges Chaos. Man muss sich Buchstabe für Buchstabe durch das Listing hangeln, um erst nach geraumer Zeit festzustellen, was die Zeilen überhaupt bewirken. Das zweite Problem: Auch die Ausgabe ist nicht wirklich sauber formatiert.

Schauen Sie sich jetzt die folgende Version des Programms an, das die exakt gleiche Ausgabe produziert:

```
Sub ZweiteVersion()

    Dim locBasisDatum As Date = #12/30/2005 1:12:32 PM#
    Dim locOffset As TimeSpan
    Dim locRandom As New Random(Now.Millisecond)

    With locBasisDatum
        Console.WriteLine("Es ist {0}, der {1}...", _
            .ToString("ddd"), _
            .ToString("d. MMMM \'yy, HH:mm"))

        '15 Wiederholungen
        For count As Integer = 1 To 15
            locOffset = New TimeSpan(locRandom.Next(23), locRandom.Next(59), 0)
            locBasisDatum = locBasisDatum.Add(locOffset)
            Console.WriteLine("...und {0} Std. und {1} Min. später ist {2}", _
                Math.Floor(locOffset.TotalHours).ToString(), _
                locOffset.Minutes.ToString(), _
                locBasisDatum.ToString("ddd, ""der"" d. MMMM \'yy, HH:mm"))
        Next
    End With
End Sub
```

Was ist passiert? Sie haben durch Platzhalter in geschweiften Klammern bestimmt, an welchen Positionen innerhalb des angegebenen Textes, der die Platzhalter enthält, die folgenden Parameter eingesetzt werden sollen. Das Ergebnis kann sich sehen lassen: Das Listing ist leicht lesbar, und man versteht fast auf Anhieb, welchem Zweck es dient.

Sie können kombinierte Formatierungen nicht nur in der `Console.WriteLine`-Anweisung einsetzen. Alle Ableitungen der `TextWriter`-Klasse verstehen mit ihrer `WriteLine`-Anweisung ebenfalls kombinierte Formatierungen. Und zu guter Letzt haben Sie mit der statischen `String.Format`-Methode die Möglichkeit, einen neuen String zu produzieren, der durch die Möglichkeiten der kombinierten Formatierung aufbereitet werden kann.

## Ausrichtungen in kombinierten Formatierungen

Die Funktionalität von kombinierten Formatierungen geht noch weiter. Sie können in den so genannten Indexkomponenten – so nennen sich die in geschweiften Klammern eingefügten Platzhalter – angeben, wie der Text mit führenden Leerzeichen ausgerichtet werden soll. Dazu geben Sie mit Komma getrennt die Anzahl der Zeichen ein, auf die der Text durch das Einfügen von führenden Leerzeichen verlängert werden soll, sodass die einzufügenden Zeichen auf einer bestimmten Position enden. Angewendet auf das schon vorhandene Beispiel, ergibt sich folgende Version des Programms:

```

Sub DritteVersion()

    Dim locBasisDatum As Date = #12/30/2005 1:12:32 PM#
    Dim locOffset As TimeSpan
    Dim locRandom As New Random(Now.Millisecond)

    With locBasisDatum
        Console.WriteLine("Es ist {0}, der {1}...", _
            .ToString("dddd"), _
            .ToString("d. MMMM \\'yy, HH:mm"))
        '15 Wiederholungen
        For count As Integer = 1 To 15
            locOffset = New TimeSpan(locRandom.Next(23), locRandom.Next(59), 0)
            locBasisDatum = locBasisDatum.Add(locOffset)
            Console.WriteLine("...und {0,2} Std. und {1,2} Min. später ist {2,11}, der {3}", _
                Math.Floor(locOffset.TotalHours).ToString(), _
                locOffset.Minutes.ToString(), _
                .ToString("dddd"), _
                .ToString("dd. MMMM \\'yy, HH:mm") _
            )
        Next
    End With
End Sub

```

Wenn Sie diese Prozedur laufen lassen, ergibt sich die folgende Ausgabe auf dem Konsolenfenster:

Es ist Freitag, der 30. Dezember '05, 13:12...	
...und 4 Std. und 14 Min. später ist	Freitag, der 30. Dezember '05, 17:26
...und 13 Std. und 43 Min. später ist	Samstag, der 31. Dezember '05, 07:09
...und 11 Std. und 11 Min. später ist	Samstag, der 31. Dezember '05, 18:20
...und 2 Std. und 26 Min. später ist	Samstag, der 31. Dezember '05, 20:46
...und 4 Std. und 49 Min. später ist	Sonntag, der 01. Januar '06, 01:35
...und 16 Std. und 4 Min. später ist	Sonntag, der 01. Januar '06, 17:39
...und 3 Std. und 1 Min. später ist	Sonntag, der 01. Januar '06, 20:40
...und 15 Std. und 28 Min. später ist	Montag, der 02. Januar '06, 12:08
...und 9 Std. und 24 Min. später ist	Montag, der 02. Januar '06, 21:32
...und 18 Std. und 18 Min. später ist	Dienstag, der 03. Januar '06, 15:50
...und 4 Std. und 26 Min. später ist	Dienstag, der 03. Januar '06, 20:16
...und 13 Std. und 17 Min. später ist	Mittwoch, der 04. Januar '06, 09:33
...und 0 Std. und 35 Min. später ist	Mittwoch, der 04. Januar '06, 10:08
...und 22 Std. und 44 Min. später ist	Donnerstag, der 05. Januar '06, 08:52
...und 15 Std. und 38 Min. später ist	Freitag, der 06. Januar '06, 00:30

Zugegeben: Schöner ist nur die Formatierung der ersten beiden Parameter geworden. Ob das Einrücken der Wochentage wirklich zur Lesbarkeit beiträgt, ist strittig. Zur Demonstration der Funktion ist es allemal ein brauchbares Ergebnis, denn: Sie können leicht erkennen, wie sich die Erweiterung der Indexkomponenten um die Angabe der Gesamtlänge der Buchstaben (die entsprechende Zeile im Listing ist fett markiert) auf das Ergebnis auswirken.

## Angeben von Formatzeichenfolgen in den Indexkomponenten

Zu guter Letzt gibt es eine weitere Möglichkeit, die Indexkomponenten zu parametrisieren. Sie können die Formatzeichenfolge, die sich in den bisherigen Versionen bei den zu formatierenden Parametern selbst befand, ebenfalls in jeder Indexkomponente angeben. Dazu trennen Sie die Formatzeichen per Doppelpunkt von den weiteren Parametern. Das folgende Listing zeigt die letzte Version der Prozedur, bei der sie von dieser Möglichkeit Gebrauch macht:

```
Sub VierteVersion()

    Dim locBasisDatum As Date = #12/30/2005 1:12:32 PM#
    Dim locOffset As TimeSpan
    Dim locRandom As New Random(Now.Millisecond)

    Console.WriteLine("Es ist {0:dddd}, der {1:d. MMMM \'yy, HH:mm}...", _
        locBasisDatum, _
        locBasisDatum)

    '15 Wiederholungen
    For count As Integer = 1 To 15
        locOffset = New TimeSpan(locRandom.Next(23), locRandom.Next(59), 0)
        locBasisDatum = locBasisDatum.Add(locOffset)
        Console.WriteLine("...und {0,2} Std. und {1,2} Min. später ist {2,11:dddd}, der {3:dd. MMMM \'yy, HH:mm}", _
            Math.Floor(locOffset.TotalHours).ToString(), _
            locOffset.Minutes.ToString(), _
            locBasisDatum, _
            locBasisDatum)
    Next
End Sub
```

---

**WICHTIG:** Achten Sie bei dieser Version auf zwei wesentliche Änderungen. Da zum einen die Formatierungsanweisungen in die Indexkomponenten verschoben wurden, dürfen Sie jetzt nicht mehr die `ToString`-Funktion der einzelnen zu formatierenden Daten verwenden, da diese zuvor für die korrekte Formatierung zuständig war. Würden Sie die `ToString`-Funktion beibehalten, so würde die Formatierungsfunktion, die durch `WriteLine` ins Leben gerufen wird (der so genannte *Formatter*), versuchen, die Formatzeichen auf eine Zeichenkette und nicht auf den Date-Typ anzuwenden. Er würde natürlich ins Leere laufen, da Zeichenketten selbst nicht mit den Formatzeichen für den Date-Typ zu formatieren sind.

---

Beachten Sie auch, dass das bündige Ausrichten durch Leerzeichen nur mit nicht-proportionalen Zeichensätzen möglich ist. Bei proportionalen Zeichensätzen, bei denen Buchstaben nicht gleich groß sind (»www« ist viel länger als »iii«, mit anderen Worten: Ich schreibe gerade in einer proportionalen Schrift) schlägt das Formatieren mit Leerzeichen natürlich fehl.

# So helfen Ihnen benutzerdefinierte Format Provider, Ihre Programme zu internationalisieren

.NET erlaubt das Erstellen von benutzerdefinierten Format Providern. Um zu verstehen, wie Sie Format Provider in .NET entwickeln, lassen Sie mich das Pferd anhand des Ergebnisses eines Beispiels von hinten aufzäumen: Stellen Sie sich vor, Sie müssten ein Programm entwickeln, dass nicht nur die Konvertierung von Maßeinheiten vornehmen kann, sondern den Entwickler seiner Klasse auch bei der Aufbereitung der Werte unterstützt.

Ein Programm soll folgendes Problem lösen: Es erlaubt seinem Anwender, einen Wert in Meter einzugeben; das Programm wird anschließend die Werte in die in deutsch- und englischsprachigen Kulturen üblichen Maßeinheiten umrechnen, etwa wie im folgenden Beispiel zu sehen:

Geben Sie einen Wert in Metern zur Umrechnung ein: 550

mm	cm	m	km
550.000,00000	55.000,00000	550,00000	0,55000
lines	inches	yards	miles
259.820,00000	21.653,50000	601,70000	0,34177

Soweit ist das Programm absolut nichts Besonderes – jeder Basic-Newbie programmierte so etwas schon vor Jahrzehnten nach ein paar Stunden.

Allerdings ist der Weg dorthin schon bemerkenswerter. Das folgende Programm zeigt, wie die einzelnen Zeilen zustande gekommen sind.

---

**BEGLEITDATEIEN:** Sie finden dieses Programm im Verzeichnis `.\VB 2005 - Entwicklerbuch\E - Datentypen\Kap17\CustomFormatProvider01\`:

---

```
Module CustomFormatProvider

Sub Main()

    Dim locEngKultur As New CultureInfo("en-US")

    Console.WriteLine("Geben Sie einen Wert in Metern zur Umrechnung ein: ")
    Dim locLaenge As New Length(Decimal.Parse(Console.ReadLine()))

    'Umgerechneten Wert anzeigen:
    Console.WriteLine("      mm      |      cm      |      m      |      km      |")
    Console.WriteLine("{0,17} |{1,17} |{2,17} |{3,17} |", _
        locLaenge.ToString("s-d;#,##0.00000"), _
        locLaenge.ToString("k-d;#,##0.00000"), _
        locLaenge.ToString("m-d;#,##0.00000"), _
        locLaenge.ToString("g-d;#,##0.00000"))

    Console.WriteLine()
    'Umgerechneten Wert anzeigen:
    Console.WriteLine("      lines      |      inches      |      yards      |      miles      |")
    Console.WriteLine("{0,17} |{1,17} |{2,17} |{3,17} |", _
        locLaenge.ToString("s;#,##0.00000", locEngKultur), _
```

```

    locLaenge.ToString("k-e;#,##0.00000"), _
    locLaenge.ToString("m;#,##0.00000", _
        New LaengenFormatInfo(LaengenKultur.EnglischAmerikanisch, LaengenAufloesung.Mittel)), _
    locLaenge.ToString("g;#,##0.00000", locEngKultur))
Console.ReadLine()

End Sub

End Module

```

Sie können leicht erkennen, dass Sie in diesem Programm quasi keine einzige Berechnung finden können. Es werden auch keine Eigenschaften oder Funktionen einer speziellen Klasse aufgerufen – alle Konvertierungen finden ausschließlich über die Steuerung entweder von Formatzeichenfolgen oder – was auf den ersten Blick noch nicht offensichtlich ist – über mehr oder weniger spezielle Format Provider statt.

Die Konvertierungen werden aber nichtsdestotrotz durch eine spezielle Klasse realisiert – sie nennt sich für dieses Beispiel schlicht und einfach Laengen. Dies ist eine von mir selbst geschriebene Klasse, suchen Sie sie also nicht im .NET Framework, dazu später mehr. Sie stellt auf der einen Seite die wirklich simplen Konvertierungsfunktionen zur Verfügung (mit Methoden wie ToMile, ToInch, etc.). Auf der anderen Seite erweitert sie die ToString-Funktion der Basisklasse (sie ist direkt von Object abgeleitet). ToString nimmt wahlweise einen oder zwei Parameter entgegen, und zwar in dem Stil, den Sie in den vergangenen Abschnitten schon kennen gelernt haben. Sie verarbeitet Formatzeichenfolgen – wahlweise in Kombination mit einem Format Provider.

Die Formatzeichenfolgen akzeptieren Formatzeichenfolgen, die eigentlich Kombinationen aus zweien sind – oder zumindest sein können. Der erste Teil einer Formatzeichenfolge steuert, welche Maßeinheit bei der Ausgabe verwendet werden soll; der zweite Teil – und jetzt kommt der internationalisierende Part – bestimmt, welche Kulturvorgaben dabei verwendet werden sollen. Aus diesem Grund bestimmen Sie mit der Angabe von den Formatzeichenfolgen auch nicht direkt »Zentimeter« oder »Inch« (die englische Einheit für Zoll), sondern geben vielmehr eine Skalierungsbezeichnung an, wahlweise in Kombination mit einem Kulturbuchstaben. Die Skalierung habe ich der Einfachheit halber in *sehr klein, klein, mittel* und *groß* festgelegt. Diese Version der Klasse unterscheidet ferner die beiden Kulturen *deutsch* und *amerikanisch-englisch*.

Damit haben Sie, ohne direkte Bezeichnungen definieren zu müssen, kulturabhängig die Möglichkeit, verschiedene Skalierungen zu verwenden. Mit den folgenden Anweisungen lassen Sie beispielsweise eine Zeile ausgeben, die die *sehr kleine* Skalierung mit einem Wert für die Klasse Laenge von 1 verwendet:<sup>3</sup>

```

Sub Spielchen()

    'Definiert eine Laengen-Instanz mit 1 (einem Meter).
    Dim locLaenge As New Laengen(1)
    'Gibt auf einem deutschen System den Klasseninstanzwert in Millimeter aus.
    Console.WriteLine(locLaenge.ToString("s"))

```

---

<sup>3</sup> Die einzelnen Testroutinen befinden sich alle innerhalb des Moduls. Wenn Sie sie selber ausprobieren wollen, brauchen Sie am Anfang des Moduls lediglich die Kommentare der ersten beiden Zeilen zu entfernen und den Aufruf der jeweils vorgestellten Prozedur dort einzusetzen.

```

'Auf einem englischen oder amerikanischen System würde die vorherige Zeile
'die gleiche Ausgabe, wie die folgende bewirken:
Console.WriteLine(locLaenge.ToString("s", New CultureInfo("en-US")))
Console.ReadLine()
End Sub

```

Dieses Programm gibt die folgenden Zeilen aus:

```

1000
472,4

```

*Sehr klein* im Deutschen bedeutet bei diesem Beispiel *Millimeter*. Da die Klasse in der Einheit *Meter* definiert wird, druckt die entsprechende Programmzeile korrekt 1000 (für 1000 Millimeter) aus und in der englischen Version, in der *sehr klein* die Einheit *Lines* bedeutet, korrekt 472,4.

Welche Kultur Sie bei der Ausgabe berücksichtigen, lässt sich bei der Laengen-Klasse nicht nur mit dem *CultureInfo*-Objekt steuern, wie im letzten Beispiel gesehen. Sie haben nämlich auch die Möglichkeit, die Kultur in einer Erweiterung der Formatzeichenfolge zu bestimmen, etwa wie im folgenden Beispiel, das kein *CultureInfo*-Objekt verwendet:

```

Sub Spielchen2()
    'Definiert eine Laengen-Instanz mit 1 (einem Meter).
    Dim locLaenge As New Laengen(1)
    'Gibt auf jedem System den Klasseninstanzwert in Millimeter aus.
    Console.WriteLine(locLaenge.ToString("s-d"))
    'Gibt auf jedem System den Klasseninstanzwert in Lines aus.
    Console.WriteLine(locLaenge.ToString("s-e"))

    Console.ReadLine()
End Sub

```

Die Ausgabe ist dieselbe wie im vorherigen Beispiel.

Die Steuerung mit Formatzeichen erlaubt aber noch mehr. Mit Semikolon getrennt können Sie eine Formatierung für die Werte an sich bestimmen, wie Sie es im ► Abschnitt »Formatierung von numerischen Ausdrücken durch Formatzeichenfolgen« ab Seite 501 schon kennen gelernt haben. Ein weiteres Beispiel zeigt diese Verwendung der Formatzeichen:

```

Sub Spielchen3()
    'Definiert eine Laengen-Instanz mit 1 (einem Meter).
    Dim locLaenge As New Laengen(1)
    'Gibt auf jedem System den Klasseninstanzwert in Millimeter aus.
    Console.WriteLine(locLaenge.ToString("s-d;#,##0.00"))
    'Gibt auf jedem System den Klasseninstanzwert in Lines aus.
    Console.WriteLine(locLaenge.ToString("s-e;#,##0.00"))

    Console.ReadLine()
End Sub

```

Die Ausgabe lautet jetzt:

```

1.000,00
472,40

```

die Zahlen wurden den Formatzeichen entsprechend formatiert.

Die folgende Tabelle zeigt Ihnen, welche Zeichenkombinationen die `ToString`-Funktion meiner `Laengen`-Klasse auswerten kann:

Formatzeichen	Bedeutung	deutsche Maßeinheit (-d)	englisch-amerikanische Maßeinheit (-e)
s	sehr klein	Millimeter	Lines
k	klein	Zentimeter	Inches
m	mittel	Meter	Yard
g	groß	Kilometer	Miles

**Tabelle 17.6:** Die `Laengen`-Klasse versteht diese Formatzeichen

Bislang haben Sie von eigentlichen benutzerdefinierten Format Providern noch nichts gesehen. Die Formatsteuerung mit Formatzeichen in Kombination mit dem `CultureInfo`-Objekt schien zwar schon ganz gut zu funktionieren, aber die Ausgabeform eines `Laengen`-Klassenwertes konnten Sie nur auf Kulturseite beeinflussen.

Zusätzlich zur `Laengen`-Klasse gibt es im Beispielprogramm aber auch einen richtigen Format Provider – er nennt sich passenderweise `LaengenFormatInfo`. Seine Funktionsweise demonstriert das folgende Beispiel:

```
Sub Spielchen4()
    'Definiert eine Laengen-Instanz mit 1 (einem Meter).
    Dim locLaenge As New Laengen(1)
    Dim locLaengenFormatInfo As New LaengenFormatInfo

    locLaengenFormatInfo.Aufloesung = LaengenAufloesung.SehrKlein
    locLaengenFormatInfo.Kultur = LaengenKultur.Deutsch

    'Gibt auf jedem System den Klasseninstanzwert in Millimeter aus.
    Console.WriteLine(locLaenge.ToString(locLaengenFormatInfo))
    'Gibt auf jedem System den Klasseninstanzwert in Lines aus.
    locLaengenFormatInfo.Kultur = LaengenKultur.EnglischAmerikanisch
    Console.WriteLine(locLaenge.ToString(locLaengenFormatInfo))

    Console.ReadLine()

End Sub
```

Die Verwendungsweise lehnt sich an die bekannten Format Provider `NumberFormatInfo` und `DateTimeFormatInfo` an. Sie instanzieren die Klasse, setzen bestimmte Eigenschaften (im Beispieldiagramm fett gekennzeichnet), und wenn Sie die Klasseninstanz der `ToString`-Funktion der `Laengen`-Klasse übergeben, passt sie die Formatierung des Ausgabetextes entsprechend an.

Das Ergebnis dieses Beispiels ist wieder das des vorherigen.

Nachdem Sie die `Laengen`-Klasse nun umfassend anzuwenden gelernt haben, will ich Ihnen die genaue Funktionsweise nicht vorenthalten.

Die Klasse besteht zunächst einmal aus dem Konstruktor, und einer ganzen Menge einfacher Umrechnungsfunktionen, die folgendermaßen implementiert sind:

```
Public Class Laengen
```

```
    'Speichert die Länge in Meter.
```

```
    Private myLaenge As Decimal
```

```
    Sub New(ByVal Meter As Decimal)
```

```
        myLaenge = Meter
```

```
    End Sub
```

```
    Public Shared Function FromMile(ByVal Mile As Decimal) As Laengen
```

```
        Return New Laengen(Mile * 1609D)
```

```
    End Function
```

```
    Public Shared Function FromYard(ByVal Yard As Decimal) As Laengen
```

```
        Return New Laengen(Yard * 0.9144D)
```

```
    End Function
```

```
    Public Shared Function FromInch(ByVal Inch As Decimal) As Laengen
```

```
        Return New Laengen(Inch * 0.0254D)
```

```
    End Function
```

```
    Public Shared Function FromLine(ByVal Line As Decimal) As Laengen
```

```
        Return New Laengen(Line * 0.002117D)
```

```
    End Function
```

```
    Public Shared Function FromKilometer(ByVal Kilometer As Decimal) As Laengen
```

```
        Return New Laengen(Kilometer * 1000D)
```

```
    End Function
```

```
    Public Shared Function FromCentimeter(ByVal Centimeter As Decimal) As Laengen
```

```
        Return New Laengen(Centimeter * 0.01D)
```

```
    End Function
```

```
    Public Shared Function FromMillimeter(ByVal Millimeter As Decimal) As Laengen
```

```
        Return New Laengen(Millimeter * 0.001D)
```

```
    End Function
```

```
    Public Function ToMeter() As Decimal
```

```
        Return myLaenge
```

```
    End Function
```

```
    Public Function ToKilometer() As Decimal
```

```
        Return myLaenge * 0.001D
```

```
    End Function
```

```
    Public Function ToCentimeter() As Decimal
```

```
        Return myLaenge * 100D
```

```
    End Function
```

```
    Public Function ToMillimeter() As Decimal
```

```
        Return myLaenge * 1000D
```

```
    End Function
```

```

Public Function ToMile() As Decimal
    Return myLaenge * 0.0006214D
End Function

Public Function ToYard() As Decimal
    Return myLaenge * 1.094D
End Function

Public Function ToInch() As Decimal
    Return myLaenge * 39.37D
End Function

Public Function ToLine() As Decimal
    Return myLaenge * 472.4D
End Function

```

Das Interessante sind anschließend die verschiedenen Überladungen der `ToString`-Funktionen, mit denen der Inhalt der Klasseninstanz in Zeichenketten umgewandelt werden kann:

```

Public Overloads Function ToString(ByVal format As String) As String
    Return ToString(format, Nothing)
End Function

Public Overloads Function ToString(ByVal formatProvider As System.IFormatProvider) As String
    Return ToString(Nothing, formatProvider)
End Function

Public Overloads Function ToString(ByVal formatChars As String,
    ByVal formatProvider As System.IFormatProvider) As String

    Trace.WriteLine("ToString (Formattable-Signatur) wurde aufgerufen!")

    If (TypeOf formatProvider Is CultureInfo) Or formatProvider Is Nothing Then
        formatProvider = LaengenFormatInfo.FromFormatProvider(formatProvider)
    ElseIf Not (TypeOf formatProvider Is LaengenFormatInfo) Then
        Dim up As New _
            FormatException("Der Format Provider wird für die Klasse Laengen nicht unterstützt!")
        Throw up
    End If

    'LaengenFormatInfo-Provider enthält die Format-Aufbereitungsroutine
    Return DirectCast(formatProvider, LaengenFormatInfo).Format(formatChars, Me, formatProvider)
End Function

```

End Class

Erwähnenswert an dieser Stelle ist die Vorgehensweise zum Erkennen des Typs des übergebenen Format Providers am Anfang des Listings. Hier wird nämlich kein fester Typ als Parameter übernommen, sondern eine Schnittstelle. Eine Schnittstelle deswegen, damit wahlweise ein `CultureInfo`-Objekt, ein `LaengenFormatInfo`-Objekt oder `Nothing` übergeben werden kann. Mit `Type Of` überprüft `ToString` dabei, um welchen Typ es sich bei der Schnittstelle genau handelt (die relevanten Stellen sind im Listing wieder fett markiert). Sollte `Nothing` oder eine `CultureInfo` übergeben worden sein,

dann kümmert sich die statische Methode `FromFormatProvider` der `LaengenFormatInfo`-Klasse darum, dass im Anschluss nur mit eben diesem Format Provider gearbeitet wird. Und eigentlich macht genau diese Funktionsweise die Internationalisierung des Programms aus: Wenn kein Format Provider übergeben wurde, dann legt – wie wir später noch sehen werden – die statische Funktion nicht etwa direkt ein `LaengenFormatInfo`-Objekt an, sondern zunächst ein `CultureInfo`-Objekt. Dieses `CultureInfo`-Objekt ist aber nicht irgendeins, sondern es spiegelt die voreingestellte Kultur des aktuellen Threads wider – und damit die Grundeinstellung des Rechners. Erst jetzt passiert die Konvertierung in ein `LaengenFormatInfo`-Objekt – mit dem Ergebnis, dass auf einem englischen System automatisch englische und auf einem deutschen System automatisch deutsche Maßeinheiten verwendet werden.

Die Formatierungsroutine selbst befindet sich ebenfalls in unserem `LaengenFormatInfo`-Objekt. Da das zu diesem Zeitpunkt, zu dem wir es zum Aufruf von `Format` benötigen, aber unbedingt vorhanden ist (alle anderen möglicherweise artfremden Format Provider sind zu diesem Zeitpunkt entweder konvertiert worden oder haben eine Ausnahme ausgelöst), können wir die Schnittstellenvariable `formatProvider`, ohne eine Ausnahme zu riskieren, in ein `LaengenFormatInfo`-Objekt casten, um uns den Zugang zu dessen `Format`-Methode zu erschließen.

Die komplette Aufbereitungsfunktionalität an sich findet anschließend in der `LaengenFormatInfo`-Klasse in eben dieser Funktion statt. Diese Klasse ist im folgenden Listing zu sehen.

```
Public Enum LaengenAufloesung
    SehrKlein      ' Millimeter oder Line
    Klein          ' Zentimeter oder Inch
    Mittel         ' Meter oder Yard
    Gross          ' Kilometer oder Mile
End Enum

Public Enum LaengenKultur
    EnglischAmerikanisch
    Deutsch
End Enum
```

Diese beiden Enums (mehr zu Enums im nächsten Kapitel) dienen lediglich dazu, dem Entwickler den Umgang mit der im Anschluss besprochenen `LaengenFormatInfo`-Klasse zu erleichtern; er muss sich dann keine Nummern für Parametereinstellungen merken, sondern kann per Namen darauf zugreifen. Die eigentliche `LaengenFormatInfo`-Klasse verwendet die Enums an verschiedenen Stellen.

```
Public Class LaengenFormatInfo
    Implements IFormatProvider
    Private myKultur As LaengenKultur
    Private myAufloesung As LaengenAufloesung

    Sub New()
        myKultur = LaengenKultur.Deutsch
        myAufloesung = LaengenAufloesung.Mittel
    End Sub

    Sub New(ByVal Kultur As LaengenKultur)
        myKultur = Kultur
        myAufloesung = LaengenAufloesung.Mittel
    End Sub
```

```

Sub New(ByVal Kultur As LaengenKultur, ByVal Aufloesung As LaengenAufloesung)
    myKultur = Kultur
    myAufloesung = Aufloesung
End Sub
Public Shared Function FromFormatProvider(ByVal formatProvider As IFormatProvider) As LaengenFormatInfo

    Dim retLaengenFormatInfo As LaengenFormatInfo

    If formatProvider Is Nothing Then
        formatProvider = CultureInfo.CurrentCulture
    End If

    If DirectCast(formatProvider, CultureInfo).ThreeLetterISOLanguageName = "deu" Then
        retLaengenFormatInfo =
            New LaengenFormatInfo(LaengenKultur.Deutsch, LaengenAufloesung.Mittel)
    Else
        retLaengenFormatInfo =
            New LaengenFormatInfo(LaengenKultur.EnglischAmerikanisch, LaengenAufloesung.Mittel)
    End If
    Return retLaengenFormatInfo
End Function

```

Erste Station: die schon angesprochene statische Funktion `FromFormatProvider`, die dafür sorgt, dass jeder ankommende Format Provider automatisch in einen `LaengenFormatInfo`-Format-Provider umgewandelt wird. Sie sorgt dafür – wie schon gesagt –, dass die Klasse `Laengen` sich ohne weiteres Eingreifen durch den Entwickler automatisch internationalisiert.

```

Public Function GetFormat(ByVal formatType As System.Type) As Object _
    Implements System.IFormatProvider.GetFormat

    Trace.WriteLine("Ausgabe von GetFormat:" + formatType.Name)

End Function

```

Da durch die `Implements`-Anweisung am Anfang der Klasse die `IFormatProvider`-Schnittstelle eingebunden wird, muss diese Funktion `GetFormat` ebenfalls vorhanden sein. Momentan enthält sie nur eine einzelne Anweisung, die – sollte sie von wem oder was auch immer aufgerufen werden – uns darüber im Ausgabefenster (nicht Konsolenfenster!) informieren wird. Dadurch, dass wir die `IFormatProvider`-Schnittstelle einbinden, ermöglichen wir, sie auch als Parameter für `ToString` der `Laengen`-Klasse zuzulassen. Auf diese Weise schaffen wir die Möglichkeit, die Schnittstelle zu übergeben, ohne uns bei der Parameterübergabe ausschließlich auf ein `LaengenInfoFormat`-Objekt festlegen zu müssen.

---

**TIPP:** Wenn Sie das Kapitel über Schnittstellen (noch) nicht gelesen haben, dann beachten Sie Folgendes, um die benötigten Routinenrümpfe (Stubs) für die Schnittstelle `IFormatProvider` einzufügen, falls Sie ähnlichen Code selber erstellen: Tippen Sie **Implements IFormatProvider** und gehen Sie in die nächste Zeile mit **Eingabe** (nicht mit den Cursortasten!). Fertig.

---

```

Public Function Format(ByVal formatChars As String, _
    ByVal arg As Object,
    ByVal formatProvider As System.IFormatProvider) As String _

```

```

Dim locLaengen As Laengen

Trace.WriteLine("Format (CustomFormatter-Signatur) wurde aufgerufen!")
'Dafür sorgen, dass das zu formatierende Element und der Format Provider übereinstimmen.
If Not TypeOf arg Is Laengen Then
    Return String.Format(formatProvider, formatChars, arg)
End If

locLaengen = DirectCast(arg, Laengen)

'Dafür sorgen, dass die Formatzeichenfolge nie "nichts" ist.
If formatChars Is Nothing Then
    formatChars = ""
End If

'Mit Semikolon können Formatzeichen für die Formatierung des eigentlichen Wertes folgen.
Dim locSemikolonPos As Integer = formatChars.IndexOf(";";c)

'Standardzeichen für die Formatzeichen zur Werteformatierung vorgeben.
Dim locNumFormat As String = "G"

'Doppelpunkt gefunden.
If locSemikolonPos > -1 Then
    'Das ist die Formatzeichenfolge für die Werteformatierung
    locNumFormat = formatChars.Substring(locSemikolonPos + 1)

    'Das für die Wahl der Längeneinheit
    formatChars = formatChars.Substring(0, locSemikolonPos)

    'Leerstring kommt nicht in Frage.
    If locNumFormat = "" Then
        locNumFormat = "G"
    End If
End If

'Nur noch kein, ein oder drei Zeichen kommen in Frage.
If formatChars.Length <> 0 And formatChars.Length <> 1 And formatChars.Length <> 3 Then
    Dim up As New FormatException("Ungültige(s) Formatzeichen für die Kulturbestimmung!")
    Throw up
End If

'Wenn drei Zeichen, dann wird die Einstellung des FormatProviders ignoriert;
'das Formatzeichen ist der Bestimmen!
If formatChars.Length = 3 Then
    If formatChars.ToUpper.EndsWith("-D") Then
        formatProvider = New LaengenFormatInfo(LaengenKultur.Deutsch)
        formatChars = formatChars.Substring(0, 1)
    ElseIf formatChars.ToUpper.EndsWith("-E") Then
        formatProvider = New LaengenFormatInfo(LaengenKultur.EnglischAmerikanisch)
        formatChars = formatChars.Substring(0, 1)
    Else
        Dim up As New FormatException("Ungültiges Formatzeichen für die Kulturbestimmung!")
        Throw up
    End If
End If

```

```

        End If
    End If

    'Zu diesem Zeitpunkt ist formatProvider unbedingt eine LaengenformatInfo,
    'Das folgende Casting kann also nicht schiefgehen:
    Dim locLaengenFormatInfo As LaengenFormatInfo = DirectCast(formatProvider, LaengenFormatInfo)

    'formatChars besteht aus (jetzt noch) nur einem Zeichen.

    If formatChars.Length = 1 Then
        'S' für 'Sehr klein'
        If formatChars.ToUpper.StartsWith("S") Then
            locLaengenFormatInfo.Aufloesung = LaengenAufloesung.SehrKlein
        End If

        'K' für 'klein'
        If formatChars.ToUpper.StartsWith("K") Then
            locLaengenFormatInfo.Aufloesung = LaengenAufloesung.Klein
        End If

        'M' für 'Mittel'
        If formatChars.ToUpper.StartsWith("M") Then
            locLaengenFormatInfo.Aufloesung = LaengenAufloesung.Mittel
        End If

        'G' für 'groß'
        If formatChars.ToUpper.StartsWith("G") Then
            locLaengenFormatInfo.Aufloesung = LaengenAufloesung.Groß
        End If
    End If

    With locLaengenFormatInfo
        'Und alle Stringausgaben anhand des Providers durchführen
        If .Kultur = LaengenKultur.Deutsch Then
            If .Aufloesung = LaengenAufloesung.SehrKlein Then
                Return locLaengen.ToMillimeter.ToString(locNumFormat)
            ElseIf .Aufloesung = LaengenAufloesung.Klein Then
                Return locLaengen.ToCentimeter.ToString(locNumFormat)
            ElseIf .Aufloesung = LaengenAufloesung.Mittel Then
                Return locLaengen.ToMeter.ToString(locNumFormat)
            ElseIf .Aufloesung = LaengenAufloesung.Groß Then
                Return locLaengen.ToKilometer.ToString(locNumFormat)
            End If
        Else
            If .Aufloesung = LaengenAufloesung.SehrKlein Then
                Return locLaengen.ToLine.ToString(locNumFormat)
            ElseIf .Aufloesung = LaengenAufloesung.Klein Then
                Return locLaengen.ToInch.ToString(locNumFormat)
            ElseIf .Aufloesung = LaengenAufloesung.Mittel Then
                Return locLaengen.ToYard.ToString(locNumFormat)
            ElseIf .Aufloesung = LaengenAufloesung.Groß Then
                Return locLaengen.ToMile.ToString(locNumFormat)
            End If
        End If
    End With

```

```

    End If
End With
End Function

```

Und hier findet sie nun statt, die Aufbereitung der Zeichenkette für die formatierte Ausgabe. Sie macht unseren Format Provider erst wirklich zu einem *Format* Provider. Doch im Grunde genommen sind die knapp 100 Zeilen, in denen die Aufbereitung des Wertes und die Auswertung der Formatzeichenfolgen stattfindet, nichts Besonderes. Die eine oder andere Zeichenkettenanalyse, ein paar Bedingungsauswertungen – das war es schon.

```

Public Property Kultur() As LaengenKultur
Get
    Return myKultur
End Get
Set(ByVal Value As LaengenKultur)
    myKultur = Value
End Set
End Property
Public Property Aufloesung() As LaengenAufloesung
Get
    Return myAufloesung
End Get
Set(ByVal Value As LaengenAufloesung)
    myAufloesung = Value
End Set
End Property
End Class

```

Und damit ist das Geheimnis der Funktionsweise unserer Laengen-Klasse und ihres eigenen Format Providers auch schon gelüftet. Eine Sache fehlt allerdings noch, und die hat es in sich:

## Benutzerdefinierte Format Provider durch **IFormatProvider** und **ICustomFormatter**

Was bislang kein Beispielprogramm gezeigt hat, ist ein Feature, auf das Sie bei der Formatierung von Datumswerten und numerischen Daten zurückgreifen können: die direkte Einbindung von Formatzeichenfolgen beispielsweise in `WriteLine` oder `String.Format`. Da wir im jetzigen Stand einen Format Provider implementiert haben, schauen wir, was passiert, wenn wir diesen Format Provider zusammen mit unserem Datentyp in einer solchen Kombination einsetzen:

---

**BEGLEITDATEIEN:** Sie finden dieses Projekt im Verzeichnis `.\VB 2005 - Entwicklerbuch\E - Datentypen\Kap17\CustomFormatProvider02`.

---

```

Sub FormatterTest()
    'Definiert eine Laengen-Instanz mit 1 (einem Meter).
    Dim locLaenge As New Laengen(1)
    Dim locLaengenFormatInfo As New LaengenFormatInfo

    locLaengenFormatInfo.Aufloesung = LaengenAufloesung.SehrKlein
    locLaengenFormatInfo.Kultur = LaengenKultur.EnglischAmerikanisch

```

```

'Testen der Format-Funktion.
Dim locStr As String = String.Format(locLaengenFormatInfo, _
    "Testausgabe {0:; ########.00} eines Laengen-Objektes", _
    locLaenge)
Console.WriteLine(locStr)

'Testen des Formatters.
Console.WriteLine("Testausgabe {0:g-d; ########.00} eines Laengen-Objektes", locLaenge)

Console.ReadLine()

End Sub

```

Doch leider passiert nach dem Programmstart nicht das, was wir eigentlich erwarten. Die Ausgabe lautet schlicht:

```

Testausgabe CustomFormatProvider.Laengen eines Laengen-Objektes
Testausgabe CustomFormatProvider.Laengen eines Laengen-Objektes

```

Anstatt die formatierten Werte auszugeben, haben beide Zeile lediglich den Namen des Objektes in das Konsolenfenster geschrieben. Allerdings: Im Ausgabefenster (nicht im Konsolenfenster!) erscheint ein Hinweis, der uns bei der Lösung dieses Problems einen Schritt weiter bringt.

Dort ist nämlich

```

GetFormat (IFormatProvider-Signatur) wurde aufgerufen:ICustomFormatter
zu lesen, und genau diese Zeile drucken wir durch die Anweisung
Trace.WriteLine("GetFormat (IFormatProvider-Signatur) wurde aufgerufen:" + formatType.Name)
in der GetFormat-Methode der LaengenFormatInfo-Klasse aus. Die Frage, die sich jetzt stellt: Was hat
diese Funktion aufgerufen und warum?

```

Um genau zu sein: Die AppendFormat-Methode des StringBuilder-Objektes, die .NET-intern für die Aufbereitung einer zu formatierenden Zeichenfolge mithilfe eines Format Providers zuständig ist, war für den Aufruf von GetFormat verantwortlich. Sie selbst wurde über den Umweg String.Format aufgerufen und fragt uns, welcher benutzerdefinierte Format Provider die eigentliche Formatierung vornehmen soll. Da unsere Routine GetFormat z.Z. noch Nothing zurückgibt (genau genommen gibt sie gar nichts zurück – aber das entspricht ja buchstäblich *Nothing* ...), interpretiert sie das als Antwort »keiner«. Sie greift deswegen auf ein Notprogramm zurück und ruft die parameterlose ToString-Funktion des aufzubereitenden Objektes (Instanz von Laengen) auf. Wir haben diese aber nicht übergeschrieben, und deswegen liefert sie – da von Object abgeleitet und somit unverändert übernommen – nur den Namen der Klasse zurück, und genau das haben wir als Ausgabe im Konsolenfenster sehen können.

Nur wenn wir innerhalb von GetFormat einen für unseren Datentyp gültigen Formatter zurückliefern, wird AppendFormat diesen verwenden und anschließend dessen Format-Routine aufrufen. Unsere Aufgabe ist es also lediglich, einen Formatter zu implementieren und eine Instanz als Funktionsergebnis von GetFormat an AppendFormat zurückzuliefern. AppendFormat weiß dann, wen es zur Formatierung heranziehen soll. Die einzige sinnvolle Instanz, die GetFormat zu dieser Zeit kennt, ist aber die eigene.

Das bedeutet, dass die LaengenFormatInfo-Klasse auch die ICustomFormatter-Schnittstelle implementieren muss, damit sie zum Formatter wird und ein erlaubtes Funktionsergebnis zurückliefern kann.

Einige Handgriffe reichen, um zum Ziel zu gelangen: Am Anfang des Codes der LaengenFormatInfo-Klasse muss die Implements-Anweisung um die IFormatProvider-Schnittstelle ergänzt werden:

```
Public Class Laengen
    Implements IFormatProvider

    'Speichert die Länge in Meter.
    Private myLaenge As Decimal
```

Die IFormatProvider-Schnittstelle verlangt, dass eine Format-Funktion in der Klasse existiert, die die Formatierung durchführt. Sie muss als Signatur einen String mit Formatzeichen und das zu formatierende Objekt sowie ein weiteres Objekt entgegennehmen, das die IFormatProvider-Schnittstelle implementiert. Zufälligerweise<sup>4</sup> haben wir genau so eine Version von Format schon im Programm. Es reicht, diese Funktion mit der Implements-Anweisung zu versehen, damit das Implementieren der IFormattable-Schnittstelle abgeschlossen ist:

```
Public Function Format(ByVal formatChars As String, _
                      ByVal arg As Object,
                      ByVal formatProvider As System.IFormatProvider) As String
    Implements ICustomFormatter.Format

    Trace.WriteLine("Format (CustomFormatter-Signatur) wurde aufgerufen!")
    'Dafür sorgen, dass das zu formatierende Element und der FormatProvider übereinstimmen
    If Not TypeOf arg Is Laengen Then
        Return String.Format(formatProvider, formatChars, arg)
    End If
    .
    .
    .
```

Und zu guter Letzt teilen wir der AppendFormat-Methode, die GetFormat der LaengenFormatInfo aufruft, noch mit, dass sie unsere LaengenFormatInfo-Klasse als Formatter verwenden kann. Das machen wir folgendermaßen:

```
Public Function GetFormat(ByVal formatType As System.Type) As Object
    Implements System.IFormatProvider.GetFormat
    Trace.WriteLine("Auszabe von GetFormat:" + formatType.Name)
    'Wird mein Typ verlangt?
    If formatType.Name = "ICustomFormatter" Then
        'Ja, diese Instanz ist erlaubt zu handeln!
        Return Me
    Else
        'Falscher Typ, diese Instanz darf nichts machen, denn
        'wenn sie als Provider einem nicht kompatiblen Typ
        'übergeben wird, geht's in die Hose.
        Return Nothing
    End If
```

---

<sup>4</sup> ;-) – vielleicht nicht ganz so zufällig...

```
    End If  
End Function
```

Wenn wir das Programm nun abermals starten, sieht das Ergebnis viel besser aus:

```
Testausgabe 472,40000 eines Laengen-Objektes  
Testausgabe CustomFormatProvider.Laengen eines Laengen-Objektes
```

String.Format liefert jetzt brauchbare Ergebnisse – WriteLine direkt allerdings noch nicht.

Wir haben im vergangenen Abschnitt gesehen, dass AppendFormat sich richtig Mühe gibt, korrekte Formatierungen auch mit Objekten durchzuführen, die dem Framework fremd sind. Und diese Bemühungen gehen noch einen Schritt weiter. Denn bevor alle Stricke reißen und AppendFormat auf die parameterlose ToString-Funktion eines Objektes zurückgreift, sucht es automatisch nach einer weiteren Schnittstelle, die das Objekt implementieren könnte. Ihr Name: IFormattable

## Automatisch formatierbare Objekte durch Einbinden von IFormattable

Die IFormattable-Schnittstelle ist in ihrer Handhabung eigentlich sogar noch einfacher als die Format-Provider-Methode – leider auch nicht ganz so flexibel, denn sie muss auf eine korrelierende Format-Provider-Klasse beim Aufbereiten des Strings verzichten und sich ganz auf Formatzeichenfolgen verlassen.

Für unser Beispiel ist der Aufwand umso geringer, als wir eine Formatierungs-Engine, die Formatzeichen auswerten kann, von vornherein implementiert haben.

Damit AppendFormat, das in letzter Instanz auch von Console.WriteLine für das Zusammenbauen einer Parameterzeichenfolge (»text {0} text {1} ...«) verwendet wird, auf diese Formatierungs-Engine unserer Beispielklasse zurückgreift, brauchen wir lediglich die IFormattable-Schnittstelle in unsere Laengen-Klasse einzubinden:

```
Public Class Laengen  
    Implements IFormattable  
  
    'Speichert die Länge in Meter.  
    Private myLaenge As Decimal  
  
    Sub New(ByVal Meter As Decimal)  
        myLaenge = Meter  
    End Sub  
  
    .  
    .  
    .
```

IFormattable verlangt, dass es eine ToString-Funktion mit entsprechender Signatur in der sie einbindenden Klasse gibt. Die Signatur verlangt, dass eine Formatzeichenfolge und ein IFormatProvider übergeben werden können. Aber auch eine solche ToString-Version gibt es in unserer Klasse bereits:

```

Public Overloads Function ToString(ByVal formatChars As String, _
    ByVal formatProvider As System.IFormatProvider) As String Implements IFormattable.ToString

    Trace.WriteLine("ToString (Formattable-Signatur) wurde aufgerufen!")

    If (TypeOf formatProvider Is CultureInfo) Or formatProvider Is Nothing Then
        .
        .
        .

```

Wir waren gut vorbereitet, was? Denn das war es auch schon. Wenn Sie die Formatierungsprozedur

```

Sub FormatterTest()

    'Definiert eine Laengen-Instanz mit 1 (einem Meter).
    Dim locLaenge As New Laengen(1)
    Dim locLaengenFormatInfo As New LaengenFormatInfo

    locLaengenFormatInfo.Aufloesung = LaengenAufloesung.SehrKlein
    locLaengenFormatInfo.Kultur = LaengenKultur.EnglischAmerikanisch

    'Testen der Format-Funktion.
    Dim locStr As String = String.Format(locLaengenFormatInfo, _
        "Testausgabe {0:; ########.0.00} eines Laengen-Objektes", _
        locLaenge)
    Console.WriteLine(locStr)
    'Testen des Formatters.
    Console.WriteLine("Testausgabe {0:g-d; ########.0.00} eines Laengen-Objektes", locLaenge)

    Console.ReadLine()
End Sub

```

anschließend abermals starten, sehen Sie folgendes Ergebnis auf dem Bildschirm:

```

Testausgabe 472,40000 eines Laengen-Objektes
Testausgabe ,00100 eines Laengen-Objektes

```

Sie sehen: Sowohl `String.Format` mit Unterstützung eines eigenen Format Providers als auch `WriteLine` funktionieren jetzt perfekt!



# 18 Enums (Aufzählungen)

---

- 
- 534 Bestimmung der Werte der Aufzählungselemente**
  - 535 Bestimmung der Typen der Aufzählungselemente**
  - 536 Konvertieren von Enums**
  - 537 Flags-Enum (Flags-Aufzählungen)**
- 

Enums (etwa: *Aufzählungen*, nicht zu verwechseln mit *Auflistungen*, den *Collections*) dienen in erster Linie dazu, Programmierern das Leben zu erleichtern. Programmierer müssen sich gerade in den Zeiten von .NET schon eine ganze Menge merken und sind für jede Unterstützung in dieser Richtung dankbar.

Mit Enums rufen Sie konstante Werte, die thematisch zu einer Gruppe gehören, per Namen ab. Gleichzeitig haben Sie die Möglichkeit, die Werte auf diese Namen zu beschränken. Angenommen, Sie haben eine kleine Kontaktverwaltung auf die Beine gestellt, und diese Datenbankanwendung erlaubt es Ihnen, Ihre Kontakte zu kategorisieren. Sie können also einstellen, ob ein Kontakt ein Kunde, ein Bekannter, ein Geschäftskollege ist oder sonst einer Gruppierung angehört. Für jede dieser Kategorien haben Sie eine bestimmte Nummer vergeben. Und in Abhängigkeit bestimmter Kategorien (Nummern) können Sie nun spezielle Funktionen in Ihrer Datenbankanwendung aufrufen oder auch nicht (Ansprechpartner-Kontakte lassen sich beispielsweise Firmenkontakten zuordnen, Lieferanten aber nicht, da diese Kontakte je selbst eine Firma darstellen – nur als Beispiel). Ohne Enums könnte man den Code so schreiben, dass man Ihre Funktionen auch mit der Nummer (z.B. mit einem Integer) aufruft, aber dies ist zum einen schwer zu merken und zum anderen könnten Sie kaum verhindern, dass man die Funktionen mit 42 aufruft; was als Integer vollkommen in Ordnung wäre – nur haben Sie eben keine 42. Kategorie, sondern gerade mal 10 o. Ä. So leisten Enums sowohl die bessere Merkbarkeit als auch die Einschränkung möglicher Werte.

Wie gesagt: Sie können sich eine eigene Liste erstellen und die Zuordnung Kontaktkategorie/Nummer sozusagen im Kopf durchführen. Oder Sie legen praktischerweise eine solche Enum an, die Ihnen die Arbeit erleichtert.

---

**BEGLEITDATEIEN:** Die in einem Projekt zusammengefassten Beispiele für dieses Kapitel finden Sie im Verzeichnis `.|VB 2005 - Entwicklerbuch|E - Datentypen\Kap18\Enums`.

---

```

Public Enum KontaktKategorie
    Familie
    Freund
    Bekannter
    Kollege
    Geschäftspartner
    Kunde
    Lieferant
    ZuMeiden
    Firma
    AnsprechpartnerBeiFirmenKontakt
End Enum

```

Wie setzen Sie diese Enum-Elemente nun ein? Ganz einfache Geschichte: Wenn nichts anderes gesagt wird, vergibt .NET den Enums Werte, und zwar von oben nach unten bei 0 angefangen in aufsteigender Reihenfolge – was Sie nun aber nicht mehr wissen müssen. Sie können Variablen vom Typ der Aufzählung definieren und anstatt eine Nummer zu verwenden, einfach den Enum-Typ angeben:

```

Sub main()
    Dim locKontakte As KontaktKategorie

    locKontakte = KontaktKategorie.Geschäftspartner
    Console.WriteLine(locKontakte)
    Console.ReadLine()

```

End Sub

Wenn Sie diesen Code ausführen, gibt es Ihnen im Konsolenfenster den Wert 4 aus.

## Bestimmung der Werte der Aufzählungselemente

Falls Sie mit der vorgegebenen Durchnummerierung nicht einverstanden sind, legen Sie bei der Aufzählungsdefinition eben selbst Hand an, wie im folgenden Beispiel:

```

Public Enum KontaktKategorie
    Familie = 10
    Freund
    Bekannter
    Kollege
    Geschäftspartner
    Kunde = 20
    Lieferant
    ZuMeiden = 30
    Firma
    AnsprechpartnerBeiFirmenKontakt
End Enum

```

Das gleiche Programm, abermals ausgeführt, liefert anschließend den Wert 14.

## Dubletten sind erlaubt!

Dubletten sind bei den Aufzählungselementen übrigens erlaubt. So haben in der Aufzählungsdefinition

```
Public Enum Kontaktkategorie
    Familie = 10
    Freund
    Bekannter
    Kollege
    Geschäftspartner = 20
    Kunde
    Lieferant = 19
    ZuMeiden
    Firma
    AnsprechpartnerBeiFirmenKontakt
End Enum
```

sowohl Geschäftspartner als auch ZuMeiden den Wert 20.

## Bestimmung der Typen der Aufzählungselemente

Solange nichts anderes gesagt wird, werden die Elemente einer Aufzählung intern als Integer angelegt. Sie können allerdings bestimmen, ob die Aufzählungselemente darüber hinaus als Byte, Short oder Long deklariert werden sollen. Eine entsprechende Typen-Angabe bei der Enum-Definition reicht aus:

```
Public Enum KontaktKategorie As Short
    Familie
    Freund
    Bekannter
    ...
End Enum
```

## Ermitteln des Elementtyps zur Laufzeit

Wenn Sie zur Laufzeit ermitteln müssen, welcher primitive Datentyp sich hinter einem Aufzählungselement verbirgt, machen Sie das einfach mit der GetUnderlyingType-Eigenschaft:

```
'Ermittelt den Namen des zu Grunde liegenden primitiven Datentyps einer Aufzählung.
Console.WriteLine([Enum].GetUnderlyingType(GetType(KontaktKategorie)).Name)
```

```
'Ermittelt den Typnamen anhand einer Aufzählungsvariablen.
Console.WriteLine([Enum].GetUnderlyingType(locKontakte.GetType).Name)
```

Auf unser bisheriges Beispiel angewendet, würden diese beiden Zeilen folgende Ausgabe im Konsole-Fenster erscheinen lassen:

```
Int16
Int16
```

---

**WICHTIG:** Wenn Sie auf den Enum-Typbezeichner im Quelltext zugreifen, müssen Sie, wie hier im Beispiel, das Schlüsselwort in eckige Klammern setzen, damit es vom Visual Basic-Editor korrekt als Ausdruck verarbeitet werden kann.

---

## Konvertieren von Enums

In vielen Fällen kann es sinnvoll sein, ein Aufzählungselement in seinen zu Grunde liegenden Typ umzuwandeln – beispielsweise um den Wert einer Enum-Variablen in eine Datenbank zu schreiben. Einige Fälle machen es auch nötig, einen Aufzählungswert auf Grund des als Zeichenkette vorliegenden Namens eines Aufzählungselementes entstehen zu lassen oder ein Aufzählungselement in seinen Elementnamen (also in einen String) umzuwandeln.

### In Zahlenwerte umwandeln und aus Werten definieren

Um ein Aufzählungselement in seinen Wert umzuwandeln (und im Bedarfsfall auch zurück), verfahren Sie folgendermaßen (das folgende Beispiel geht immer noch von einer Aufzählungstyp-Definition als Short aus):

```
Dim locKontakte As KontaktKategorie  
Dim locShort As Short  
  
locKontakte = KontaktKategorie.Geschäftspartner  
  
'Typumwandlung von Aufzählung zu zu Grunde liegenden Datentyp...  
locShort = locKontakte  
locShort = KontaktKategorie.Firma  
  
'....und umgekehrt, auch nicht schwieriger:  
locKontakte = CType(locShort, KontaktKategorie)
```

### In Strings umwandeln und aus Strings definieren

Falls Sie wissen wollen, welcher Elementname sich hinter dem Wert einer Aufzählungsvariablen verbirgt, verfahren Sie folgendermaßen:

```
Dim locKontakte As KontaktKategorie = KontaktKategorie.Firma  
Console.WriteLine(locKontakte.ToString())
```

Die Ausgabe lautet:

Firma

Beim umgekehrten Verfahren wird es wieder ein wenig aufwändiger:

```
'Umwandlung zurück in ein Enum-Element aus einem String.  
Dim locString As String = "Geschäftspartner"  
locKontakte = DirectCast([Enum].Parse(GetType(KontaktKategorie), locString), KontaktKategorie)
```

Hier gilt: Die statische Funktion Parse der Enum-Klasse erlaubt das Generieren eines Aufzählungselementes zur Laufzeit. Parse erwartet dabei den Typ der Aufzählung, den Sie zunächst mit GetType ermitteln müssen. Da Parse ein Objekt erzeugt, das ein geboxtes Aufzählungselement enthält, müssen Sie dieses zu guter Letzt mit DirectCast aus dem Object wieder »entboxen«.

## Flags-Enum (Flags-Aufzählungen)

Flags-Aufzählungen sind eine ideale Einrichtung, wenn Sie Aufzählungen mit Elementen verwenden müssen, die untereinander kombinierbar sind. So kann es durchaus sein, dass – um bei unserem Beispiel zu bleiben – ein Kontakt in Ihrer Datenbank sowohl ein *Freund* als auch ein *Geschäftskollege* ist. Das Framework unterstützt solche Szenarien auf ideale Weise.

Bei der Definition einer Flags-Aufzählung müssen Sie drei Dinge beachten: Sie sollten eine Aufzählungsbenennung für keine der Kombinationen definieren (beispielsweise in Form von Keine oder None). Dieser Eintrag hat den Wert 0. Zweitens: Sie müssen Werte vergeben, die sich bitweise kombinieren lassen – und dazu zählen Sie die einzelnen Werte als Zweierpotenzen hoch. Und drittens: Sie statteten die Aufzählung mit dem Flags-Attribut aus.

Das folgende Beispiel zeigt, wie es geht:

```
<Flags()>
Public Enum KontaktKategorien
    Keine = 0
    Familie = 1
    Freund = 2
    Bekannter = 4
    Kollege = 8
    Geschäftspartner = 16
    Kunde = 32
    Lieferant = 64
    ZuMeiden = 128
    Firma = 256
    AnsprechpartnerBeiFirmenKontakt = 512
End Enum
```

Wenn Sie anschließend kombinierte Zuweisungen an eine Variable vom Typ KontaktKategorien vornehmen wollen, nehmen Sie den logischen Or-Operator zu Hilfe, wie das folgende Beispiel zeigt:

```
Sub EnumFlags()
    Dim lockkontakte As KontaktKategorien
    lockkontakte = KontaktKategorien.Freund Or KontaktKategorien.Geschäftspartner
    Console.WriteLine(lockkontakte)
    Console.WriteLine(lockkontakte.ToString())
    Console.ReadLine()
End Sub
```

Dieses Beispiel erzeugt die folgende Ausgabe:

```
18
Freund, Geschäftspartner
```

## Abfrage von Flags-Aufzählungen

Bei der Abfrage von *Flags*-Aufzählungen müssen Sie ein wenig aufpassen, da Kombinationen Werte ergeben, die keinem bestimmten Wert eines Elementes entsprechen. Erst ein so genanntes Ausmaskieren (Ermitteln eines einzelnen Bitwertes) mit dem And-Operator ergibt einen richtigen Wert. Auch hier demonstriert ein Beispiel, wie es geht:

```
Dim locKontakte As KontaktKategorien  
locKontakte = KontaktKategorien.Freund Or KontaktKategorien.Geschäftspartner  
  
'Achtung bei Flags! Bits müssen ausmaskiert werden!  
'Diese Abfrage liefert das falsche Ergebnis!  
If locKontakte = KontaktKategorien.Geschäftspartner Then  
    Console.WriteLine("Du bist ein Geschäftspartner")  
Else  
    Console.WriteLine("Du bist kein Geschäftspartner")  
End If  
  
'So ist's richtig; diese Abfrage liefert das richtige Ergebnis:  
If (locKontakte And KontaktKategorien.Freund) = KontaktKategorien.Freund Then  
    Console.WriteLine("Du bist ein Freund")  
Else  
    Console.WriteLine("Du bist kein Freund")  
End If  
  
'Und so funktionieren Kombiabfragen:  
If locKontakte = (KontaktKategorien.Freund Or KontaktKategorien.Geschäftspartner) Then  
    Console.WriteLine("Du bist ein Freund und ein Geschäftspartner")  
End If
```

Wenn Sie dieses Beispiel laufen lassen, erhalten Sie das folgende Ergebnis:

```
Du bist kein Geschäftspartner  
Du bist ein Freund  
Du bist ein Freund und ein Geschäftspartner
```

Die erste Zeile wird falsch ausgegeben, da der Wert der *Enum*-Variablen *locKontakte* nicht ausmaskiert und dann verglichen, sondern direkt verglichen wird.

# 19 Arrays und Auflistungen (Collections)

---

- 540 **Grundsätzliches zu Arrays**
  - 558 **Enumerator**
  - 562 **Grundsätzliches zu Auflistungen**
  - 565 **Die wichtigen Auflistungen der Base Class Library**
  - 571 **Hashtables – für das Nachschlagen von Objekten**
  - 587 **Queue – Warteschlangen im FIFO-Prinzip**
  - 589 **Stack – Stapelverarbeitung im LIFO-Prinzip**
  - 590 **SortedList – Elemente ständig sortiert halten**
- 

Arrays kennt fast jedes Basic-Derivat seit Jahrzehnten – und natürlich können Sie auch in Visual Basic .NET auf diese Datenfelder (so der deutsche Ausdruck) zurückgreifen. Doch .NET wäre nicht .NET, wenn nicht auch Arrays viel mehr Möglichkeiten bieten würden, als nur auf Daten indiziert zuzugreifen.

Arrays basieren in .NET letzten Endes wie alle anderen .NET-Klassen auf dem grundlegenden aller Datentypen, auf `Object`, und man kann sie deswegen auch als ein solches behandeln. Arrays sind also nicht nur ein simples Sprachelement in Visual Basic selbst, sondern sie gehören zur Basis des Frameworks – zur Base Class Library.

Das bedeutet, dass die Leistung von Arrays weit über das reine Zur-Verfügung-Stellen von Containern für die Speicherung verschiedener Elemente eines Datentyps hinausreicht. Da Arrays auf `Object` basieren und damit eine eigene Klasse darstellen (`System.Array` nämlich), bietet das Framework über Array-Objekte weitreichende Funktionen zur Verwaltung ihrer Elemente an.

So können Arrays beispielsweise quasi auf Knopfdruck sortiert werden. Liegen sie in sortierter Form vor, können Sie auch binär nach ihren Elementen suchen und viele weitere Dinge mit ihnen anstellen, ohne selbst den entsprechenden Code dafür entwickeln zu müssen.

Zu guter Letzt bildet der Typ `System.Array` aber auch die Basis für weitere Datentypen – zum Beispiel `ArrayList` – die Datenelemente in einer bestimmten Form verwalten können.

Dieses Kapitel zeigt Ihnen, was Sie mit Arrays und den von ihnen abgeleiteten Klassen alles anstellen können.

# Grundsätzliches zu Arrays

Arrays im ursprünglichen Basic-Sinne dienen dazu, *mehrere* Elemente desselben Datentyps unter einem bestimmten Namen verfügbar zu machen. Um die einzelnen Elemente zu unterscheiden, bedient man sich eines Indexes (ganz einfach ausgedrückt: einer Nummerierung der Elemente), damit man auf die verschiedenen Array-Elemente zugreifen kann.

---

**BEGLEITDATEIEN:** Viele der größeren nun folgenden Beispiele sind im Projekt *Arrays* in verschiedenen *Subs* zusammengefasst. Das Projekt befindet sich im Verzeichnis *.\VB 2005 - Entwicklerbuch\E - Datentypen\Kap19\Arrays*. Sie können dieses Projekt verwenden, um die Beispiele an Ihrem eigenen Rechner nachzuvollziehen oder um eigene Experimente mit Arrays durchzuführen.

---

Ein Beispiel:

```
Sub Beispield1()

    'Array mit 10 Elementen fest deklarieren.
    'Wichtig: Anders als in C# oder C++ wird der Index
    'des letzten Elementes, nicht die Anzahl der Elemente
    'festgelegt! Elementzählung beginnt bei 0.
    'Die folgende Anweisung definiert also 10 Elemente:
    Dim locIntArray(9) As Integer
    'Zufallsgenerator initialisieren,
    Dim locRandom As New Random(Now.Millisecond)

    For count As Integer = 0 To 9
        locIntArray(count) = locRandom.Next
    Next

    For count As Integer = 0 To 9
        Console.WriteLine("Element Nr. {0} hat den Wert {1}", count, locIntArray(count))
    Next

    Console.ReadLine()

End Sub
```

Wenn Sie dieses Beispiel ausführen, erscheint im Konsolenfenster eine Liste mit Werten, etwa wie im nachstehenden Bildschirmauszug zu sehen (die Werte sollten sich natürlich von Ihren unterscheiden, da es zufällige sind).<sup>1</sup>

```
Element Nr. 0 hat den Wert 1074554181
Element Nr. 1 hat den Wert 632329388
Element Nr. 2 hat den Wert 1312197477
Element Nr. 3 hat den Wert 458430355
```

---

<sup>1</sup> Kleine Anmerkung am Rande: Ganz so zufällig sind die Werte nicht – *Random* stellt nur sicher, dass generierte Zahlenfolgen zufällig *verteilt* sind. Bei gleicher Ausgangsbasis (definiert durch den Parameter *Seed*, den Sie der *Random*-Klasse beim Instanziieren übergeben) produziert *Random* auch gleiche Zahlenfolgen. Da wir hier die Millisekunde als Basis übergeben, und eine Sekunde aus 1000 Millisekunden besteht, gibt es eine Wahrscheinlichkeit von 1:1000, dass Sie dieselbe wie die hier abgedruckte Zahlenfolge generieren lassen.

```

Element Nr. 4 hat den Wert 1970029554
Element Nr. 5 hat den Wert 503465071
Element Nr. 6 hat den Wert 112607304
Element Nr. 7 hat den Wert 1507772275
Element Nr. 8 hat den Wert 1111627006
Element Nr. 9 hat den Wert 213729371

```

Dieses Beispiel demonstriert die grundsätzliche Anwendung von Arrays. Natürlich sind Sie bei der Definition des Elementtyps nicht auf Integer festgelegt. Es gilt der Grundsatz: Jedes Objekt in .NET kann Element eines Arrays sein.

## Arrays als Parameter und Funktionsergebnis

Da Arrays in .NET ganz normale Objekte sind, können sie natürlich auch als Parameter in Funktionsaufrufen und als Funktionsergebnis verwendet werden. Das folgende Beispiel definiert ein Array mit String-Objekten und ruft eine Funktion auf, die das Array mit zufällig generierten Strings füllt. Eine weitere Funktion gibt das Ergebnis im Konsolenfenster aus:

```

Sub Beispiel2()
    Dim locAnzahlStrings As Integer = 15
    Dim locStringArray(locAnzahlStrings) As String
    locStringArray = GeneriereStrings(locAnzahlStrings, 30)
    DruckeStrings(locStringArray)
    Console.ReadLine()
End Sub

Function GeneriereStrings(ByVal AnzahlStrings As Integer, ByVal LaengeStrings As Integer) As String()
    Dim locRandom As New Random(Now.Millisecond)
    Dim locChars(LaengeStrings - 1) As Char
    Dim locStrings(AnzahlStrings - 1) As String

    For locOutCount As Integer = 0 To AnzahlStrings - 1
        For locInCount As Integer = 0 To LaengeStrings - 1
            Dim locIntTemp As Integer = Convert.ToInt32(locRandom.NextDouble * 52)
            If locIntTemp > 26 Then
                locIntTemp += 97 - 26
            Else
                locIntTemp += 65
            End If
            locChars(locInCount) = Convert.ToChar(locIntTemp)
        Next
        locStrings(locOutCount) = New String(locChars)
    Next

    Return locStrings
End Function

Sub DruckeStrings(ByVal locStringArray As String())
    For count As Integer = 0 To locStringArray.Length - 1
        If Not (locStringArray(count) Is Nothing) Then
            Console.WriteLine(locStringArray(count))
        Else

```

```

        Console.WriteLine("--- NOTHING ---")
    End If
    Next
End Sub

```

Wenn Sie dieses Beispiel laufen lassen, sehen Sie im Konsolenfenster eine Reihe von Zeichenketten, die in etwa den folgenden entsprechen:

```

SfwMyCKJJckzKp0sPJXHPxZfRwXqxB
[DKZJpJGIuLRiAKLhhmfThqBUGRvC
fTMIBhp1g[jKBdwYQZVGQqFSYHWUXp
jrlGPsF[zdVvBMwUSEuEfhpmpPuju
LVOrzzw0Esq[r1UosWFqS[TkCvWUb
dLrETAiLVFWqqLPHFESAXYffTvvH
xdcnbizWZ1neXRkUckIVvqSC[GnM{[
FnEgIsuDPeQ1gTfX{Hv1RLnmvHL[NV
vnWzg{[rJsZYVeFJJzxEHgROJuAT1B
hUqujcPingxJyCMtpJg1yMDJOPWIpm
{HFIMFicvdubMrHyhPCRFsnADURhgU
GHsBqgEqRHKONDrxXBMQCiHFZhIUFFr
oBwcXnjKLURyWgRejJgUEfPmzCUUY
Wym{TJSmgEorBmWrjKSrelwYXkXhqY
1PqnvwFYdxZsDFtdDttvQtBeukCOdj

```

Sie sehen an diesem Beispiel, dass Sie Arrays als Parameter wie ganz normale andere Objekte in .NET verwenden können – doch dazu mehr in einem der nächsten Abschnitte.

Sie sehen an diesem Beispiel auch, dass sich Arrays dynamisch dimensionieren lassen – die Größe eines Arrays muss Ihnen also nicht schon zur Entwurfszeit Ihres Programms bekannt sein. Sie können – und das macht den Einsatz von Arrays unter .NET so flexibel – Arrays mit variablen Werten (im wahrsten Sinne des Wortes) zur Laufzeit anlegen.

## Änderung der Array-Dimensionen zur Laufzeit

Das Definieren der Array-Größe mit variablen Werten macht Arrays – wie im Beispiel des letzten Abschnittes gezeigt – zu einem sehr flexiblen Werkzeug bei der Verarbeitung von großen Datenmengen. Doch Arrays sind noch flexibler: Mit der ReDim-Anweisung gibt Ihnen Visual Basic die Möglichkeit, ein Array noch nach seiner ersten Deklaration neu zu dimensionieren. Damit werden selbst einfache Arrays zu dynamischen Datencontainern. Auch hier soll ein Beispiel den Umgang demonstrieren und verdeutlichen:

```

Sub Beispiel3()
    Dim locAnzahlStrings As Integer = 15
    Dim locStringArray As String()
    locStringArray = GeneriereStrings(locAnzahlStrings, 30)
    Console.WriteLine("Ausgangsgröße: {0} Elemente. Es folgt der Inhalt:", locStringArray.Length)
    Console.WriteLine(New String("c", 40))
    DruckeStrings(locStringArray)

```

```

'Wir brauchen 10 weitere, die alten sollen aber erhalten bleiben!
ReDim Preserve locStringArray(locStringArray.Length + 9)
'Bleiben die alten wirklich erhalten?
Console.WriteLine()
Console.WriteLine("Inhaltsüberprüfung:", locStringArray.Length)
Console.WriteLine(New String("="c, 40))
DruckeStrings(locStringArray)

'10 weitere Elemente generieren.
Dim locTempStrings(9) As String
'10 Zeichen mehr pro Element, sodass wir die neuen leicht erkennen können.
locTempStrings = GeneriereStrings(10, 40)
'In das "alte" Array kopieren, aber ab Index 15,
locTempStrings.CopyTo(locStringArray, 15)

'und nachschauen, was nun wirklich drinsteht!
Console.WriteLine()
Console.WriteLine("Inhaltsüberprüfung:", locStringArray.Length)
Console.WriteLine(New String("="c, 40))
DruckeStrings(locStringArray)

Console.ReadLine()
End Sub

```

Dieses Beispiel macht sich die Möglichkeit zunutze (direkt in der ersten Codezeile), die Dimensionierung und Deklaration eines Arrays zeitlich voneinander trennen zu können. Das Array `locStringArray` wird zunächst nur als Array deklariert – wie groß es sein soll, wird zu diesem Zeitpunkt noch nicht bestimmt. Dabei spielt es in Visual Basic übrigens keine Rolle, ob Sie eine Variable in diesem

`Dim locStringArray As String()`

oder diesem

`Dim locStringArray() As String`

Stil als Array deklarieren.

Die Größe des Arrays wird das erste Mal von der Prozedur `GeneriereString` festgelegt. Hier erfolgt zwar die Dimensionierung eines zunächst völlig anderen Arrays (`locStrings`); da dieses Array aber als Rückgabewert der aufrufenden Instanz überlassen wird, lebt der hier erstellte Array-Inhalt unter anderem Namen (`locStringArray`) weiter. Das durch beide Objektvariablen referenzierte Array ist dasselbe (im ursprünglichen Sinne des Wortes).

Übrigens: Diese Vorgehensweise entspricht eigentlich schon einem typsichereren Neudimensionieren eines Arrays. Wie Sie beim ersten Array-Beispiel gesehen haben, spielt es natürlich keine Rolle, ob Sie eine Array-Variable zur Aufnahme eines Array-Rückgabeparameters verwenden, die zuvor mit einer festen Anzahl an Elementen oder ohne die Angabe der Array-Größe dimensioniert wurde. Allerdings: Sie verlieren bei dieser Vorgehensweise den Inhalt des ursprünglichen Arrays, denn die Unterroutine erstellt ein neues Array, und mit der Zuweisung an die Objektvariable `locStringArray` wird intern nur ein Zeiger umgebogen. Der Speicherbereich der alten Elemente ist nicht mehr referenzierbar.

---

**WICHTIG:** Diese Tatsache hat Konsequenzen, denn: Anders, als es bei Visual Basic 6.0 noch der Fall war, werden Arrays bei einer Zuweisung an eine andere Objektvariable *nicht* automatisch kopiert. Array-Variablen verhalten sich so wie jeder andere Referenztyp auch unter .NET: Ein Zuweisen einer Array-Variablen an eine andere biegt nur ihren Zeiger auf den Speicherbereich der eigentlichen Daten im Managed Heap um. Die Elemente bleiben an ihrem Platz im Managed Heap, und es wird kein neuer Speicherbereich mit einer Kopie der Array-Elemente für die neue Objektvariable erzeugt!

---

Das Umdimensionieren kann nicht nur über Zuweisungen, sondern – wie im Beispielcode zu sehen – auch mit der `ReDim`-Anweisung erfolgen. Mit dem zusätzlichen Schlüsselwort `Preserve` haben Sie darüber hinaus auch die Möglichkeit zu bestimmen, dass die alten Elemente dabei erhalten bleiben. Man möchte meinen, dass diese Verhaltensweise die Regel sein sollte, doch mit dem Wissen im Hinterkopf, was beim Neudimensionieren mit `ReDim` genau passiert, sieht die Sache schon anders aus:

- Wird `ReDim` für eine Objektvariable aufgerufen, wird eine komplett neue Klasseninstanz eines Arrays erstellt. Ein entsprechender Speicherbereich wird dafür reserviert.
- Der Zeiger für die Objektvariable auf den Bereich für die zuvor zugeordneten Array-Elemente wird auf den neuen Speicherbereich umgebogen.
- Der Speicherbereich, der die alten Array-Elemente enthielt, fällt dem nächsten Garbage-Collector-Durchlauf zum Opfer.
- Wenn `Preserve` hinter der `ReDim`-Anweisung platziert wird, bleiben die alten Array-Elemente erhalten. Doch das entspricht nicht der vollständigen Erklärung des Vorgangs. In Wirklichkeit wird auch hier ein neuer Speicherbereich erstellt, der den Platz für die neu angegebene Anzahl an Array-Elementen bereithält. Auch bei `Preserve` wird der Zeiger auf den Speicherbereich mit den alten Elementen für die betroffene Objektvariable auf den neuen Speicherbereich umgebogen. Doch bevor der alte Speicherbereich freigegeben wird und sich der Garbage Collector über die alten Elemente hermachen kann, werden die vorhandenen Elemente (soweit wie möglich, falls das neue Array über weniger Elemente verfügt) in den neuen Bereich kopiert.

Aus diesem Grund können Sie mit `Preserve` nur Elemente retten, die in eindimensionalen Arrays gespeichert sind oder die durch die letzte Dimension eines mehrdimensionalen Arrays angesprochen werden.

### Arrays sind .NET-Objekte – das hat den einen oder anderen Vorteil!

Jedes typsichere<sup>2</sup> Array in .NET ist von `System.Array` abgeleitet. Eine Objektvariable vom Typ `System.Array` kann also ebenfalls als Zeiger auf ein Array dienen. Auf den ersten Blick bringt das nicht viel, denn ein `System.Array` hat keinen *Indexer* (keine parametrisierte *Default*-Eigenschaft), sodass es zunächst so scheint, dass Sie an die Elemente nicht mehr herankommen. Aber: `System.Array` verfügt über einen *Enumerator* (eine Funktionalität, die es gestattet, mit *For/Each* auf die Elemente zuzugreifen). Und: `GetValue` und `SetValue` erlauben das Zugreifen auf die Array-Elemente. Gerade wenn Sie den Einsatz von generischen Klassen vermeiden möchten, können Sie mit ein paar Handgriffen und

---

<sup>2</sup> Typdefiniert in diesem Zusammenhang bedeutet: Jedes Element eines Arrays ist vom selben Typ, und dieser Typ wird bei der Definition des Arrays festgelegt. Das Gegenteil dazu wären Arrays, bei denen jedes Element eines Arrays ein anderes sein kann und in einem Objekt geboxed ist.

Tricks sich diesem Konzept der generischen Programmierung wenigstens bis auf ein paar Meter nähern, und eine Klasse typsicher vererbbar aber eben nicht generisch formulieren.

Mal angenommen, Sie verarbeiten alle möglichen numerischen Daten in einem Anwendungsprogramm – lassen wir dies beispielsweise eine Statistik-Anwendung sein – und Sie benötigen für die verschiedensten Datentypen eine Routine, die das Maximum aus einer Reihe von Zahlen ermittelt. Sie möchten das Rad aber nicht ständig neu erfinden und für jeden Datentyp, den Sie benötigen, eine Max-Funktion entwerfen. Idealerweise sollten Sie über eine Funktion verfügen, die folgenden Beispielcode erlaubt:

```
Sub Beispiel4()
    Dim locDoubles(100) As Double
    Dim locIntegers(100) As Integer
    Dim locDecimals(100) As Decimal
    Dim locRandom As New Random(Now.Millisecond)

    'Jedes Array mit 101 Zufallszahlen füllen.
    '(Nicht vergessen: 100 ist das höchste Element, nicht die Länge ;-)
    For count As Integer = 0 To 100
        locDoubles(count) = locRandom.NextDouble * locRandom.Next
        locDecimals(count) = Convert.ToDecimal(locRandom.NextDouble * locRandom.Next)
        locIntegers(count) = locRandom.Next
    Next

    Console.WriteLine("Das größte Element des Double-Arrays war {0}", CDbL(Max(locDoubles)))
    Console.WriteLine("Das größte Element des Integer-Arrays war {0}", CInt(Max(locIntegers)))
    Console.WriteLine("Das größte Element des Decimal-Arrays war {0}", CDec(Max(locDecimals)))
    Console.ReadLine()
End Sub
```

Sie sehen: Die Max-Funktion gibt es nur einmal, und sie nimmt beliebige Arrays entgegen, ganz gleich, welchen spezielleren, typdefinierten Array-Datentyp sie besitzen.

Möglich wird das, weil wir uns in der aufgerufenen Funktion nicht auf ein Array eines näher spezifizierten Typs fixiert haben, sondern uns ganz allgemein der Basisklassen aller typdefinierten Arrays bedienen, nämlich `System.Array`<sup>3</sup>:

```
Enum Vergleich
    Kleiner = -1
    Gleich = 0
    Größer = 1
End Enum

Function Max(ByVal Array As System.Array) As IComparable

    'Leeres Array-Objekt, dann beenden.
    If Array.Length = 0 Then
        Return Nothing
    End If
```

---

<sup>3</sup> Ich füge hier bewusst den *Namespace*-Namen *System* in die Bezeichnung ein, um den Typ *System.Array* vom Begriff *Array* zu unterscheiden.

```

' Kann nur vergleichbare Elemente vergleichen,
' aber alle primitiven Datentypen sind glücklicherweise
' vergleichbar, und sie implementieren IComparable.
Dim locICElement As IComparable

' Erstes Element als Basis holen.
locICElement = DirectCast(Array.GetValue(0), IComparable)
If locICElement Is Nothing Then
    ' Implementiert nicht IComparable, dann Abbrechen.
    Return Nothing
End If

For Each locICSchleifenElement As IComparable In Array
    If locICSchleifenElement Is Nothing Then
        Return Nothing
    End If

    If locICSchleifenElement.CompareTo(locICElement) = Vergleich.Größer Then
        locICElement = locICSchleifenElement
    End If
Next
Return locICElement

End Function

```

Die `Enum` am Anfang der Prozedur hilft lediglich, den Code leichter zu lesen, und sie dient als Parameter für die später folgende `CompareTo`-Methode.

Interessant wird es erst in der Funktion `Max`, die das eigentliche Maximum des Arrays ermittelt. Sie nimmt – wie schon gesagt – ein `System.Array` entgegen, und damit wird der Weg für alle beliebigen Arrays frei. Zurück – und das ist das Interessante bei dieser Funktion – liefert sie einen Wert vom Typ `IComparable`. Und da alle primitiven Datentypen diese Schnittstelle implementieren, kann die Funktion indirekt auch jeden Datentyp zurückliefern. Die aufrufende Instanz muss den `IComparable`-Rückgabewert lediglich mit einer Umwandlungsanweisung (`DirectCast` für Objekte oder  `CType` bzw. `Cxxx` für Wertetypen) wieder in den ihr bekannten Typ zurückwandeln.

Um den größten Wert in einem Array aus Elementen zu finden, muss die Prozedur die verschiedenen Werte miteinander vergleichen können. Idealerweise ist die Vergleichsfunktion genau die Funktion, die `IComparable` den sie einbindenden Klassen vorschreibt. Die `Max`-Prozedur kann sich also blind darauf verlassen, dass ein gültiges `IComparable`-Objekt auch die Vergleichsmethode `CompareTo` anbietet. Genau diese Tatsache macht sie sich zunutze, wenn sie mit `For/Each` durch die Schleife iteriert. Es interessiert sie gar nicht, welche Objekttypen sie vergleicht. Sind es `Decimals`, dann wird eben `CompareTo` eines `Decimal`-Datentyps aufgerufen, bei `Integer` die `CompareTo`-Methode des `Integer`-Datentyps. Schnittstellen machen es einmal mehr möglich!

Kleiner Hinweis: Sollte das Array-Element, das gerade verarbeitet wird, die `IComparable`-Schnittstelle nicht einbinden, dann wird die Objektvariable, die das Element hält, zu `Nothing`. In diesem Fall bricht die `Max`-Funktion die Verarbeitung ab und liefert auch `Nothing` als Funktionsergebnis zurück. Die aufrufende Instanz des Beispielprogramms prüft aber nicht auf `Nothing` als Rückgabergebnis, weil sie ja weiß, dass alle übergebenden Arrays `IComparable`-Elemente aufweisen und der `Nothing`-Fall niemals eintreten kann.

---

**HINWEIS:** Das soll hier im Beispiel reichen. Im wirklichen Programmiererleben sollte man gerade zu große Arrays, zu kleine Arrays oder komplett fehlende Rückgaben (*Nothing*) wie hier prüfen und diese adäquat behandeln.

---

## Wertevorbelegung von Array-Elementen im Code

Alle Arrays, die in den vorangegangenen Beispielen verwendet wurden, sind zur Laufzeit mit Daten gefüllt worden. In vielen Fällen möchten Sie aber Arrays erstellen, die Sie quasi zu Fuß mit Daten vorbelegen, die das Programm fest vorgibt.

Im Gegensatz zu Visual Basic 6.0, bei der diese Prozedur eine endlose Quälerei des Codehackens war, indem Sie jedem einzelnen Element mit dem Zuweisungsoperator den entsprechenden Wert zuweisen mussten, geht es in Visual Basic .NET viel eleganter und schneller, wie das folgende Beispiel zeigt:

```
Sub Beispiel5()

    'Deklaration und Definition von Elementen mit Double-Werten
    Dim locDoubleArray As Double() = {123.45F, 5435.45F, 3.14159274F}

    'Deklaration und spätere Definition von Elementen mit Integer-Werten
    Dim locIntegerArray As Integer()
    locIntegerArray = New Integer() {1I, 2I, 3I, 4I}
    'Deklaration und spätere Definition von Elementen mit Date-Werten
    Dim locDateArray As Date()
    locDateArray = New Date() {#12/24/2005#, #12/31/2005#, #3/31/2006#}

    'Deklaration und Definition von Elementen im Char-Array:
    Dim locCharArray As Char() = {"V"c, "B"c, ".c, "N"c, "E"c, "T"c, " "c, _
        "r"c, "u"c, "l"c, "e"c, "s"c, !"c}

    'Zweidimensionales Array
    Dim locZweiDimensional As Integer(,)
    locZweiDimensional = New Integer(,) {{10, 10}, {20, 20}, {30, 30}}

    'Oder verschachtelt (das ist nicht zwei-Dimensional)!
    Dim locVerschachtelt As Date(){}
    locVerschachtelt = New Date()() {New Date() {#12/24/2004#, #12/31/2004#}, -
        New Date() {#12/24/2005#, #12/31/2005#}}
End Sub
```

Häufigste Fehlerquelle bei dieser Definitionsvergehensweise: Der Zuweisungsoperator wird falsch gesetzt. Bei der kombinierten Deklaration/Definition wird er benötigt; definieren Sie nur neu, lassen Sie ihn weg. Beachten Sie auch den Unterschied zwischen mehrdimensionalen und verschachtelten Arrays, auf den ich im nächsten Abschnitt genauer eingehen möchte.

## Mehrdimensionale und verschachtelte Arrays

Bei der Definition von Arrays sind Sie nicht auf eine Dimension beschränkt – das ist ein Feature, das schon bei Jahrzehnttausenden Basic-Dialekten zu finden ist. Sie können ein mehrdimensionales Array erstellen, indem Sie bei der Deklaration des Arrays die Anzahl der zu verwaltenden Elemente jeder Dimension durch Komma getrennt angeben:

```
Dim DreiDimensional(5, 10, 3) As Integer
```

Möchten Sie die Anzahl der zu verwaltenden Elemente pro Dimension bei der Deklaration des Arrays nicht festlegen, verwenden Sie die folgende Deklarationsanweisung:

```
Dim AuchDreiDimensional As Integer(,,)
```

Mit ReDim oder dem Aufruf von Funktionen, die ein entsprechend dimensioniertes Array zurückliefern, können Sie anschließend das Array definieren.

## Verschachtelte Arrays

Verschachtelte Arrays sind etwas anders konzipiert als mehrdimensionale Arrays. Bei verschachtelten Arrays ist ein Array-Element selbst ein Array (welches auch wieder Arrays beinhalten kann usw.). Anders als bei mehrdimensionalen Arrays können die einzelnen Elemente unterschiedlich dimensionierte Arrays halten, und diese Zuordnung lässt sich auch im Nachhinein noch ändern.

Verschachtelte Arrays definieren Sie, indem Sie die Klammerpaare mit der entsprechenden Array-Dimension hintereinander schreiben – anders als bei mehrdimensionalen Arrays, bei denen die Dimensionen in einem Klammerpaar mit Komma getrennt angegeben werden.

Die folgenden Beispiel-Codezeilen (aus der Sub Beispiel6) zeigen, wie Sie verschachtelte Arrays definieren, deklarieren und ihre einzelnen Elemente abrufen können:

```
'Einfach verschachtelt; Tiefe wird nicht definiert.  
Dim EinfachVerschachtelt(10)() As Integer  
  
'Erstes Element hält ein Integer-Array mit drei Elementen.  
EinfachVerschachtelt(0) = New Integer() {10, 20, 30}  
  
'Zweites Element hält ein Integer-Array mit acht Elementen.  
EinfachVerschachtelt(1) = New Integer() {10, 20, 30, 40, 50, 60, 70, 80}  
  
'Drückt das dritte Element des zweiten Elementes (30) des Arrays.  
Console.WriteLine(EinfachVerschachtelt(1)(2))  
  
'In einem Rutsch alles neu zuweisen.  
EinfachVerschachtelt = New Integer()() {New Integer() {30, 20, 10},  
                                         New Integer() {80, 70, 60, 50, 40, 30, 20, 10}}  
  
'Drückt das dritte Element des zweiten Elementes (jetzt 60) des Arrays.  
Console.WriteLine(EinfachVerschachtelt(1)(2))  
Console.ReadLine()
```

## Die wichtigsten Eigenschaften und Methoden von Arrays

In den vorangegangenen Beispielprogrammen ließ es sich nicht vermeiden, die eine oder andere Eigenschaft oder Methode des Array-Objektes bereits anzuwenden. Dieser Abschnitt soll sich ein wenig genauer mit den zusätzlichen Möglichkeiten dieses Objektes beschäftigen – denn sie sind mächtig und können Ihnen, richtig angewendet, eine Menge Entwicklungszeit ersparen.

## Anzahl der Elemente eines Arrays ermitteln mit Length

Wenn Sie wissen wollen, wie viele Elemente ein Array beherbergt, bedienen Sie sich seiner Length-Eigenschaft. Bei zwei- und mehrdimensionalen Arrays ermittelt Length ebenfalls die Anzahl aller Elemente.

Aufgepasst bei verschachtelten Arrays: Hier ermittelt Length nämlich nur die Elementanzahl des umgebenden Arrays. Sie können die Array-Länge eines Elementes des umgebenden Arrays etwa so ermitteln:

```
'Verschachtelte Arrays
Dim locVerschachtelt As Date()()
locVerschachtelt = New Date()() {New Date() {#12/24/2004#, #12/31/2004#}, _
                           New Date() {#12/24/2005#, #12/31/2005#}}

Console.WriteLine("Äußeres Array hat {0} Elemente.", locVerschachtelt.Length)
Console.WriteLine("Array des 1. Elements hat {0} Elemente.", locVerschachtelt(0).Length)
```

## Sortieren von Arrays mit Array.Sort

Arrays lassen sich durch eine ganz simple Methode sortieren, und das folgende Beispiel soll demonstrieren, wie es geht:

```
Sub Beispiel7()

    'Array-Erstellen:
    Dim locNamen As String() = {"Jürgen", "Martina", "Hanna", "Gaby", "Michaela", "Miriam", "Ute", _
                                "Leonie-Gundula", "Melanie", "Uwe", "Andrea", "Klaus", "Anja", _
                                "Myriam", "Daja", "Thomas", "José", "Kricke", "Flori", "Katrín", "Momo",
                                -
                                "Gareth", "Anne"}
    System.Array.Sort(locNamen)
    DruckeStrings(locNamen)
    Console.ReadLine()
End Sub
```

Wenn Sie dieses Beispiel starten, produziert es folgendes Ergebnis im Konsolenfenster:

```
Andrea
Anja
Anne
Daja
Flori
Gaby
Gareth
Hanna
José
Jürgen
Katrín
Klaus
Kricke
Leonie-Gundula
Martina
Melanie
```

```
Michaela  
Miriam  
Momo  
Myriam  
Thomas  
Ute  
Uwe
```

Die Sort-Methode ist eine statische Methode von `System.Array`, und sie kann noch eine ganze Menge mehr. So haben Sie beispielsweise die Möglichkeit, einen korrelierenden Index mitsortieren zu lassen, oder Sie können bestimmen, zwischen welchen Indizes eines Arrays die Sortierung stattfinden soll. Die Online-Hilfe zum Framework verrät Ihnen, welche Überladungen die Sort-Methode genau anbietet.

### **Umdrehen der Array-Anordnung mit `Array.Reverse`**

Wichtig in diesem Zusammenhang ist eine weitere statische Methode von `System.Array` namens `Reverse`, die die Reihenfolge der einzelnen Array-Elemente umkehrt. Im Zusammenhang mit der Sort-Methode erreichen Sie durch den anschließenden Einsatz von `Reverse` die Sortierung eines Arrays in absteigender Reihenfolge. Wenn Sie das vorherige Beispiel um diese Zeilen

```
Console.WriteLine()  
Console.WriteLine("Absteigend sortiert:")  
Array.Reverse(locNamen)  
DruckeStrings(locNamen)  
Console.ReadLine()
```

ergänzen, sehen Sie schließlich die Namen in umgekehrter Reihenfolge im Konsolenfenster, etwa:

```
Absteigend sortiert:  
Uwe  
Ute  
Thomas  
. . .  
Anja  
Andrea
```

### **Durchsuchen eines sortierten Arrays mit `Array.BinarySearch`**

Auch bei der Suche nach bestimmten Elementen eines Arrays ist Ihnen das Framework behilflich. Dazu stellt die `Array`-Klasse die statische Funktion `BinarySearch` zur Verfügung.

---

**WICHTIG:** Damit eine binäre Suche in einem Array durchgeführt werden kann, muss das Array in sortierter Form vorliegen – ansonsten kann die Funktion falsche Ergebnisse zurückliefern. Wirklich brauchbar ist die Funktion überdies nur dann, wenn Sie sicherstellen, dass es keine Dubletten in den Elementen gibt. Da eine binäre Suche erfolgt, ist nicht gewährleistet, ob die Funktion das erste zutreffende Objekt findet oder ein beliebiges, das dem Gesuchten entsprach.

---

Beispiel:

```
Sub Beispiel8()

    'Array-Erstellen:
    Dim locNamen As String() = {"Jürgen", "Martina", "Hanna", "Gaby", "Michaela", "Miriam", "Ute", _
        "Leonie-Gundula", "Melanie", "Uwe", "Andrea", "Klaus", "Anja", _
        "Myriam", "Daja", "Thomas", "José", "Kricke", "Flori", "Katrín", "Momo",
        "Gareth", "Anne", "Jürgen", "Gaby"}

    System.Array.Sort(locNamen)
    Console.WriteLine("Jürgen wurde gefunden an Position {0}", _
        System.Array.BinarySearch(locNamen, "Jürgen"))
    Console.ReadLine()
End Sub
```

Wie Sort ist auch BinarySearch eine überladene Funktion und bietet weitere Optionen, die die Suche beispielsweise auf bestimmte Indexbereiche beschränkt. IntelliSense und die Online-Hilfe geben hier genauere Hinweise für die Verwendung.

## Implementierung von Sort und BinarySearch für eigene Klassen

Das Framework erlaubt es, Arrays von beliebigen Typen zu erstellen, auch von solchen, die Sie selbst erstellt haben. Bei der Erstellung einer Klasse, die als Element eines Arrays fungieren soll, brauchen Sie dabei nichts Besonderes zu beachten.

Anders wird das allerdings, wenn Sie eine Funktion auf das Array anwenden wollen, die einen Elementvergleich erfordert, oder wenn Sie sogar steuern wollen, nach welchen Kriterien Ihr Array beispielsweise sortiert werden soll oder auch nach welchem Kriterium Sie es mit BinarySearch durchsuchen lassen möchten. Dazu ein Beispiel:

Sie haben eine Klasse entwickelt, die die Adressen einer Kontaktdatenbank speichert. Der nachfolgend gezeigte Code demonstriert, wie sie funktioniert und wie man sie anwendet:

---

**BEGLEITDATEIEN:** Im Folgenden sehen Sie zunächst den Klassencode, der einen Adresseneintrag verwaltet – dieses Projekt finden Sie im Verzeichnis *VB 2005 - Entwicklerbuch\E - Datentypen\Kap19\Comparer01*:

---

```
Public Class Adresse
```

```
    Protected myName As String
    Protected myVorname As String
    Protected myPLZ As String
    Protected myOrt As String

    Sub New(ByVal Name As String, ByVal Vorname As String, ByVal Plz As String, ByVal Ort As String)
        myName = Name
        myVorname = Vorname
        myPLZ = Plz
        myOrt = Ort
    End Sub
```

```

Public Property Name() As String
    Get
        Return myName
    End Get
    Set(ByVal Value As String)
        myName = Value
    End Set
End Property

Public Property Vorname() As String
    Get
        Return myVorname
    End Get
    Set(ByVal Value As String)
        myVorname = Value
    End Set
End Property

Public Property PLZ() As String
    Get
        Return myPLZ
    End Get
    Set(ByVal Value As String)
        myPLZ = Value
    End Set
End Property

Public Property Ort() As String
    Get
        Return myOrt
    End Get
    Set(ByVal Value As String)
        myOrt = Value
    End Set
End Property

Public Overrides Function ToString() As String
    Return Name + ", " + Vorname + ", " + PLZ + " " + Ort
End Function
End Class

```

Sie sehen selbst: einfachstes Basic. Die Klasse stellt ein paar Eigenschaften für die von ihr verwalteten Daten zur Verfügung, und sie überschreibt die `ToString`-Funktion der Basis-Klasse, damit sie eine Adresse als kompletten String zurückliefern kann.

Das folgende kleine Beispielprogramm definiert ein Array aus dieser Klasse, richtet ein paar Adressen zum Experimentieren ein und druckt diese anschließend in einer eigenen Unterroutine aus:

```

Module ComparerBeispiel

Sub Main()
    Dim locAdressen(5) As Adresse

```

```

locAdressen(0) = New Adresse("Löffelmann", "Klaus", "11111", "Soest")
locAdressen(1) = New Adresse("Heckhuis", "Jürgen", "99999", "Gut Uhlenbusch")
locAdressen(2) = New Adresse("Sonntag", "Miriam", "22222", "Dortmund")
locAdressen(3) = New Adresse("Sonntag", "Christian", "33333", "Wuppertal")
locAdressen(4) = New Adresse("Ademmer", "Uta", "55555", "Bad Waldholz")
locAdressen(5) = New Adresse("Kaiser", "Wilhelm", "12121", "Ostenwesten")

Console.WriteLine("Adressenliste:")
Console.WriteLine(New String("=",c, 40))
DruckeAdressen(locAdressen)
'Array.Sort(locAdressen)
Console.ReadLine()
End Sub

Sub DruckeAdressen(ByVal Adressen As Adresse())
    For Each locString As Adresse In Adressen
        Console.WriteLine(locString)
    Next
End Sub

End Module

```

Auch hier werden Sie erkennen: Es passiert nichts wirklich Aufregendes. Wenn Sie das Programm starten, produziert es, wie zu erwarten, folgende Ausgabe im Konsolenfenster:

Adressenliste:

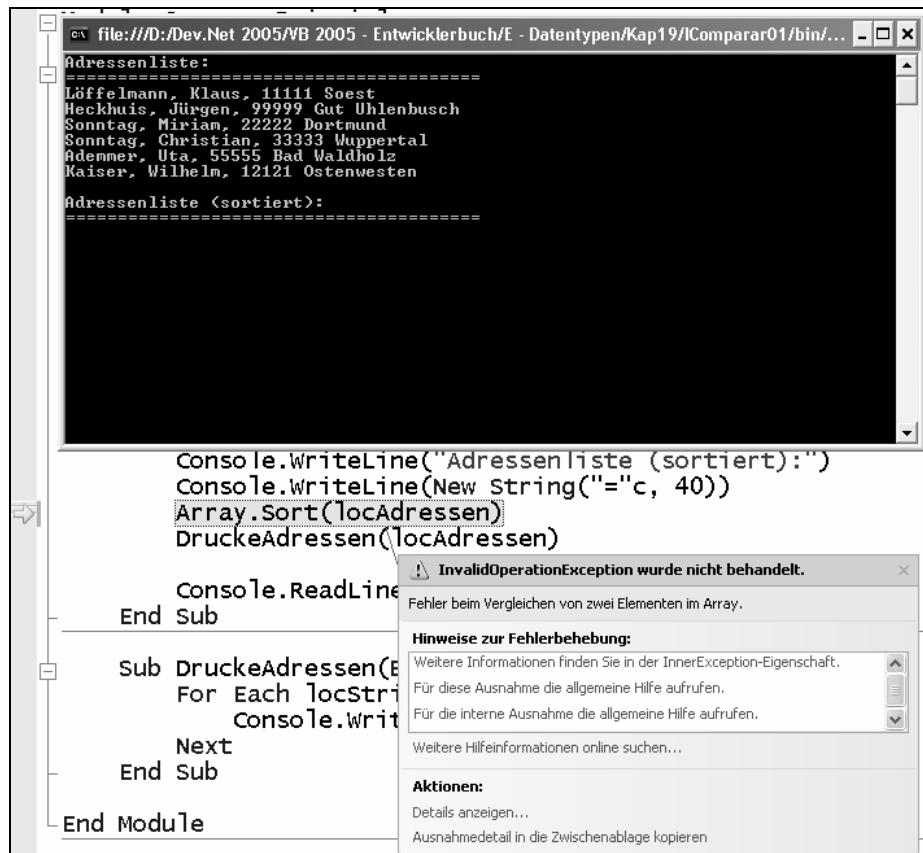
```
=====
Löffelmann, Klaus, 11111 Soest
Heckhuis, Jürgen, 99999 Gut Uhlenbusch
Sonntag, Miriam, 22222 Dortmund
Sonntag, Christian, 33333 Wuppertal
Ademmer, Uta, 55555 Bad Waldholz
Kaiser, Wilhelm, 12121 Ostenwesten
```

Die Liste, wie wir Sie anzeigen lassen, ist allerdings ein ziemliches Durcheinander. Beobachten Sie, was passiert, wenn wir das Programm um eine Sort-Anweisung ergänzen und wie die Liste anschließend ausschaut. Dazu nehmen Sie die Auskommentierung der Sort-Methode im Listing einfach zurück.

Nach dem Start des Programms warten Sie vergeblich auf die zweite, sortierte Ausgabe der Liste. Stattdessen löst das Framework eine Ausnahme aus, etwa wie in Abbildung 19.1 zu sehen.

Der Grund dafür: Die Sort-Methode der Array-Klasse versucht, die einzelnen Elemente des Arrays miteinander zu vergleichen. Dazu benötigt es einen so genannten Comparer (etwa: *Vergleicher*). Da wir nicht explizit angegeben, dass wir einen speziellen Comparer verwenden wollen (Welchen auch? – Wir haben noch keinen!), erzeugt es einen Standard-*Comparer*, der aber wiederum die Einbindung einer bestimmten Schnittstelle in der Klasse verlangt, deren Elemente er miteinander vergleichen soll. Diese Schnittstelle nennt sich *IComparable*. Leider haben wir auch diese Schnittstelle nicht implementiert, und die Ausnahme ist die Folge.

Wir haben nun drei Möglichkeiten. Wir binden die *IComparable*-Schnittstelle ein, dann können Elemente unserer Klasse auch ohne die Nennung eines expliziten *Comparers* verglichen und im Endeffekt sortiert werden.



**Abbildung 19.1:** Wenn Sie das Programm starten, löst es eine Ausnahme aus, sobald die *Sort*-Methode der *Array*-Klasse erreicht ist

Oder wir stellen der Klasse einen expliziten *Comparer* zur Verfügung; in diesem Fall müssen wir ihn beim Einsatz von *Sort* (oder auch *BinarySearch*) benennen. Dieser *Comparer* hätte den Vorteil, dass sich durch ihn steuern ließe, nach welchem Kriterium die Klasse durchsucht bzw. sortiert werden soll.

Die dritte Möglichkeit: Wir machen beides. Damit wird unsere Klasse universell nutzbar und läuft nicht Gefahr, eine Ausnahme auslösen zu können. Und genau das werden wir in den nächsten beiden Abschnitten in die Tat umsetzen.

### Implementieren der Vergleichsfähigkeit einer Klasse durch **IComparable**

Die Implementierung der *IComparable*-Schnittstelle, damit Instanzen der Klasse miteinander verglichen werden können, ist vergleichsweise simpel. Das Einbinden erfordert zusätzlich lediglich das Vorhandensein einer *CompareTo*-Methode, die die aktuelle Instanz der Klasse mit einer weiteren vergleicht. Da durch den Einsatz von *IComparable* keine Möglichkeit besteht festzulegen, welches Datenfeld das Vergleichskriterium sein soll, wird unser Kriterium eine Zeichenkette sein, die aus Name, Vorname, Postleitzahl und Ort besteht – genau die Zeichenkette also, die *Tostring* in der jetzigen Version bereits zurückliefert. Deswegen brauchen wir den eigentlichen Vergleich noch nicht

einmal selbst durchzuführen, sondern können ihn an die CompareTo-Funktion des String-Objektes, das wir von ToString zurückhalten, weiterreichen. Die Modifizierungen an der Klasse sind also denkbar gering (aus Platzgründen nur gekürzter Code, der die Änderungen widerspiegelt):

```
Public Class Adresse
    Implements IComparable
    .
    .
    .
    Public Function CompareTo(ByVal obj As Object) As Integer Implements System.IComparable.CompareTo
        Dim locAdresse As Adresse
        Try
            locAdresse = DirectCast(obj, Adresse)
        Catch ex As InvalidCastException
            Dim up As New InvalidCastException("''CompareTo' der Klasse 'Adresse' kann keine Vergleiche " + _
                "mit Objekten anderen Typs durchführen!")
            Throw up
        End Try
        Return ToString.CompareTo(locAdresse.ToString)
    End Function
End Class
```

An dieser Stelle vielleicht erwähnenswert ist der fett gekennzeichnete mittlere Bereich im Programm. Auf den ersten Blick mag es unsinnig erscheinen, einen möglichen Fehler abzufangen und ihn anschließend mehr oder weniger unverändert wieder auszulösen. Aber: Der Fehlertext ist entscheidend, und Sie tun sich selbst einen Gefallen, wenn Sie den Fehlertext einer Ausnahme, die Sie generieren, so formulieren, dass Sie eindeutig wissen, wer oder was sie ausgelöst hat. Die Fehlermeldung, die wir in der Ursprungsversion unseres Beispiels in Abbildung 19.1 gesehen habe, leistete das beispielweise nicht. Die Fehlermeldung verwirrt mehr, als sie den Zusammenhang aufklärt. Wenn Sie das Programm anschließend starten, liefert es das gewünschte Ergebnis, wie im Folgenden zu sehen:

Adressenliste:

```
=====
Löffelmann, Klaus, 11111 Soest
Heckhuis, Jürgen, 99999 Gut Uhlenbusch
Sonntag, Miriam, 22222 Dortmund
Sonntag, Christian, 33333 Wuppertal
Ademmer, Uta, 55555 Bad Waldholz
Kaiser, Wilhelm, 12121 Ostenwesten
```

Adressenliste (sortiert):

```
=====
Ademmer, Uta, 55555 Bad Waldholz
Heckhuis, Jürgen, 99999 Gut Uhlenbusch
Kaiser, Wilhelm, 12121 Ostenwesten
Löffelmann, Klaus, 11111 Soest
Sonntag, Christian, 33333 Wuppertal
Sonntag, Miriam, 22222 Dortmund
```

## Implementieren einer gesteuerten Vergleichsfähigkeit durch IComparer

So weit, so gut – unser Beispielprogramm läuft immerhin schon, und die Elemente des Arrays lassen sich sortieren. Aber: Das Programm sortiert stumpf nach dem Nachnamen – und diese Verhaltensweise können wir ihm ohne weitere Maßnahmen auch nicht abgewöhnen.

Was die Adresse-Klasse braucht, ist eine Art Steuerungseinheit, die durch Setzen bestimmter Eigenschaften bestimmt, wie Vergleiche stattfinden sollen. Wenn Sie sich zum Beispiel die Sort-Funktion von System.Array ein wenig genauer anschauen, werden Sie feststellen, dass sie als optionalen Parameter eine Variable vom Typ IComparer entgegennimmt.

Eine Klasse, die IComparer einbindet, macht genau das Verlangte. Sie kann den Vergleichsvorgang beeinflussen. Wenn Sie IComparer in einer Klasse implementieren, müssen Sie auch die Funktion Compare in dieser Klasse zur Verfügung stellen. Dieser Funktion werden zwei Objekte übergeben, die die Funktion miteinander vergleichen soll. Ist eines der Objekte größer (was auch immer das heißt, denn es ist bis dahin ein abstraktes Attribut), liefert sie den Wert 1, ist es kleiner, den Wert -1, und ist es gleich, liefert sie den Wert 0 zurück.

Ein Comparer steht in keinem direkten Verhältnis zu den Klassen, die er vergleichen soll; er wird also nicht durch weitere Schnittstellen reglementiert. Aus diesem Grund muss der Comparer selber dafür Sorge tragen, dass ihm nur die Objekte zum Vergleichen angeliefert werden, die er verarbeiten will. Bekommt er andere, muss er eine Ausnahme auslösen.

In unserem Fall muss der Comparer noch ein bisschen mehr können. Er muss die Funktionalität zur Verfügung stellen, durch die der Entwickler steuern kann, nach welchem Kriterium der Adresse-Klasse er vergleichen will. Aus diesen Gründen ergibt sich folgender Code für einen Comparer unseres Beispiels:

---

**BEGLEITDATEIEN:** Sie finden das erweiterte Projekt im Verzeichnis .\VB 2005 - Entwicklerbuch\E - Datentypen\Kap19\IComparer02\:

---

```
'Nur für den einfachen Umgang mit der Klasse.  
Public Enum ZuVergleichen  
    Name  
    PLZ  
    Ort  
End Enum  
  
Public Class AdressenVergleicher  
    Implements IComparer  
  
    'Speichert die Einstellung, nach welchem Kriterium verglichen wird.  
    Protected myZuVergleichenMit As ZuVergleichen  
  
    Sub New(ByVal ZuVergleichenMit As ZuVergleichen)  
        myZuVergleichenMit = ZuVergleichenMit  
    End Sub
```

```

Public Function Compare(ByVal x As Object, ByVal y As Object) As Integer _
    Implements System.Collections.IComparer.Compare
    'Nur erlaubte Typen durchlassen:
    If (Not (TypeOf x Is Adresse)) Or (Not (TypeOf y Is Adresse)) Then
        Dim up As New InvalidCastException("Compare' der Klasse 'Adressenvergleicher' kann nur " +
            "Vergleiche vom Typ 'Adresse' durchführen!")
        Throw up
    End If

    'Beide Objekte in den richtigen Typ casten, damit das Handling einfacher wird:
    Dim locAdr1 As Adresse = DirectCast(x, Adresse)
    Dim locAdr2 As Adresse = DirectCast(y, Adresse)

    'Hier passiert die eigentliche Steuerung,
    'nach welchem Kriterium verglichen wird:
    If myZuVergleichenMit = ZuVergleichen.Name Then
        Return locAdr1.Name.CompareTo(locAdr2.Name)
    ElseIf myZuVergleichenMit = ZuVergleichen.Ort Then
        Return locAdr1.Ort.CompareTo(locAdr2.Ort)
    Else
        Return locAdr1.PLZ.CompareTo(locAdr2.PLZ)
    End If
End Function

'Legt die Vergleichseinstellung offen.
Public Property ZuVergleichenMit() As ZuVergleichen
    Get
        Return myZuVergleichenMit
    End Get
    Set(ByVal Value As ZuVergleichen)
        myZuVergleichenMit = Value
    End Set
End Property

```

End Class

Zum Beweis, dass alles wie gewünscht läuft, ergänzen Sie das Hauptprogramm um folgende Zeilen (fettgedruckt im folgenden Listing):

Module ComparerBeispiel

```

Sub Main()
    Dim locAdressen(5) As Adresse

    locAdressen(0) = New Adresse("Löffelmann", "Klaus", "11111", "Soest")
    locAdressen(1) = New Adresse("Heckhuis", "Jürgen", "99999", "Gut Uhlenbusch")
    locAdressen(2) = New Adresse("Sonntag", "Miriam", "22222", "Dortmund")
    locAdressen(3) = New Adresse("Sonntag", "Christian", "33333", "Wuppertal")
    locAdressen(4) = New Adresse("Ademmer", "Uta", "55555", "Bad Waldholz")
    locAdressen(5) = New Adresse("Kaiser", "Wilhelm", "12121", "Ostenwesten")

```

```

Console.WriteLine("Adressenliste:")
Console.WriteLine(New String("=",c, 40))
DruckeAdressen(locAdressen)

Console.WriteLine()
Console.WriteLine("Adressenliste (sortiert nach Postleitzahl):")
Console.WriteLine(New String("=",c, 40))
Array.Sort(locAdressen, New AdressenVergleicher(ZuVergleichen.PLZ))
DruckeAdressen(locAdressen)
Console.ReadLine()
End Sub
.
.
.

End Module

```

Wenn Sie das Programm nun starten, erhalten Sie folgende Ausgabe auf dem Bildschirm:

Adressenliste:

```
=====
Löffelmann, Klaus, 11111 Soest
Heckhuis, Jürgen, 99999 Gut Uhlenbusch
Sonntag, Miriam, 22222 Dortmund
Sonntag, Christian, 33333 Wuppertal
Ademmer, Uta, 55555 Bad Waldholz
Kaiser, Wilhelm, 12121 Ostenwesten
```

Adressenliste (sortiert nach Postleitzahl):

```
=====
Löffelmann, Klaus, 11111 Soest
Kaiser, Wilhelm, 12121 Ostenwesten
Sonntag, Miriam, 22222 Dortmund
Sonntag, Christian, 33333 Wuppertal
Ademmer, Uta, 55555 Bad Waldholz
Heckhuis, Jürgen, 99999 Gut Uhlenbusch
```

## Enumeratoren

Wenn Sie Arrays oder – wie Sie später sehen werden – Auflistungen für die Speicherung von Daten verwenden, dann müssen Sie in der Lage sein, diese Daten abzurufen. Mit Indexern gibt Ihnen .NET eine einfache Möglichkeit. Sie verwenden eine Objektvariable und versehen sie mit einem Index, der auch durch eine andere Variable repräsentiert werden kann. Ändern Sie diese Variable, die als Index dient, können Sie dadurch programmtechnisch bestimmen, welches Element eines Arrays Sie gerade verarbeiten wollen. Typische Zählschleifen, mit denen Sie durch die Elemente eines Arrays iterieren, sind die Folge – etwa im folgenden Stil:

```

For count As Integer = 0 To Array.Length - 1
    TuWasMit(Array(count))
Next

```

---

**HINWEIS:** Enumeratorn haben nichts mit Enums zu tun. Lediglich die Namen sind sich etwas ähnlich. Enums sind aufgezählte Benennungen von bestimmten Werten im Programmlisting, während Enumeratorn die Unterstützung von For/Each zur Verfügung stellen!

---

Wenn ein Objekt allerdings die Schnittstelle `IEnumerable` implementiert, gibt es eine elegantere Methode die verschiedenen Elemente, die das Objekt zur Verfügung stellt, zu durchlaufen. Glücklicherweise implementiert `System.Array` die Schnittstelle `IEnumerable`, sodass Sie auf Array-Elemente mit dieser eleganten Methode – namentlich mit For/Each – zugreifen können. Ein Beispiel:

```
'Deklaration und Definition von Elementen im Char-Array
Dim locCharArray As Char() = {"V"c, "B"c, ".c, "N"c, "E"c, "T"c, " "c, _
                               "r"c, "u"c, "l"c, "e"c, "s"c, !"c}
For Each c As Char In locCharArray
    Console.WriteLine(c)
Next
Console.ReadLine()
Console.ReadLine()
```

Wenn Sie dieses Beispiel laufen lassen, sehen Sie im Konsolenfenster den folgenden Text:

VB.NET rules!

Enumeratorn werden von allen typdefinierten Arrays unterstützt und ebenso von den meisten Auflistungen. Enumeratorn können Sie aber auch in Ihren eigenen Klassen einsetzen, wenn Sie erlauben möchten, dass der Entwickler, der mit Ihrer Klasse arbeitet, durch Elemente mit For/Each iterieren kann.

## Benutzerdefinierte Enumeratorn mit `IEnumerable`

Enumeratorn können allerdings nicht nur für die Aufzählung von gespeicherten Elementen in Arrays oder Auflistungen eingesetzt werden. Sie können Enumeratorn auch dann einsetzen, wenn eine Klasse ihre Enumerations-Elemente durch Algorithmen zurück liefert.

Als Beispiel dafür möchte ich Ihnen zunächst einen Codeausschnitt zeigen, der nicht funktioniert – bei dem es aber in einigen Fällen wünschenswert wäre, wenn er funktionierte:

```
Das würde nicht funktionieren:
for d as Date=#24/12/2004# to #31/12/2004
    Console.WriteLine("Datum in Aufzählung: {0}", d)
Next d
```

Dennoch könnte es für bestimmte Anwendungen sinnvoll sein, tageweise einen bestimmten Datumsbereich zu durchlaufen. Etwa wenn Ihre Anwendung herausfinden muss, wie viele Mitarbeiter, deren Daten Ihre Anwendung speichert, in einem bestimmten Monat Geburtstag haben.

Wenn das, was wir vorhaben, nicht mit For/Next funktioniert, vielleicht können wir dann aber eine Klasse schaffen, die einen Enumerator zur Verfügung stellt, sodass das Vorhaben mit For/Each gelingt. Diese Klasse sollte beim Instanzieren Parameter übernehmen, mit denen wir bestimmen können, welcher Datumsbereich in welcher Schrittweite durchlaufen werden soll. Damit Sie mit For/Each durch die Elemente einer Klasse iterieren können, muss die Klasse die Schnittstelle `IEnumerable` einbinden.

Das kann sie nur, wenn sie gleichzeitig eine Funktion GetEnumerator zur Verfügung stellt, die erst das Objekt mit dem eigentlichen Enumerator liefert. Doch eines nach dem anderen.

---

**BEGLEITDATEIEN:** Schauen wir uns zunächst die Basisklasse an, die die Grundfunktionalität zur Verfügung stellt – Sie finden das Projekt im Verzeichnis .\VB 2005 - Entwicklerbuch\E - Datentypen\Kap19\Enumerators\). Starten Sie das Programm mit **Strg+F5**.

---

```
Public Class Datumsaufzählung
    Implements IEnumerable
    Dim locDatumsaufzähler As Datumsaufzähler

    Sub New(ByVal StartDatum As Date, ByVal EndDatum As Date, ByVal Schrittweite As TimeSpan)
        locDatumsaufzähler = New Datumsaufzähler(StartDatum, EndDatum, Schrittweite)
    End Sub
    Public Function GetEnumerator() As System.Collections.IEnumerable _
        Implements System.Collections.IEnumerable.GetEnumerator
        Return locDatumsaufzähler
    End Function
End Class
```

Sie sehen, dass diese Klasse selbst nichts Großartiges macht – sie schafft durch die ihr übergebenen Parameter lediglich die Rahmenbedingungen und stellt die von `IEnumerable` verlangte Funktion `GetEnumerator` zur Verfügung. Die eigentliche Aufgabe wird von der Klasse `Datumsaufzähler` erledigt, die auch als Rückgabewert von `GetEnumerator` zurückgegeben wird.

Eine Instanz dieser Klasse wird bei der Instanzierung von `Datumsaufzählung` erstellt. Was in dieser Klasse genau passiert, zeigt der folgende Code:

```
Public Class Datumsaufzähler
    Implements IEnumerable

    Private myStartDatum As Date
    Private myEndDatum As Date
    Private mySchrittweite As TimeSpan
    Private myAktuellesDatum As Date

    Sub New(ByVal StartDatum As Date, ByVal EndDatum As Date, ByVal Schrittweite As TimeSpan)
        myStartDatum = StartDatum
        myAktuellesDatum = StartDatum
        myEndDatum = EndDatum
        mySchrittweite = Schrittweite
    End Sub

    Public Property StartDatum() As Date
        Get
            Return myStartDatum
        End Get
        Set(ByVal Value As Date)
            myStartDatum = Value
        End Set
    End Property
```

```

Public Property EndDatum() As Date
    Get
        Return myEndDatum
    End Get
    Set(ByVal Value As Date)
        myEndDatum = Value
    End Set
End Property

Public Property Schrittweite() As TimeSpan
    Get
        Return mySchrittweite
    End Get
    Set(ByVal Value As TimeSpan)
        mySchrittweite = Value
    End Set
End Property

Public ReadOnly Property Current() As Object Implements System.Collections.IEnumerator.Current
    Get
        Return myAktuellesDatum
    End Get
End Property

Public Function MoveNext() As Boolean Implements System.Collections.IEnumerator.MoveNext
    myAktuellesDatum = myAktuellesDatum.Add(Schrittweite)
    If myAktuellesDatum > myEndDatum Then
        Return False
    Else
        Return True
    End If
End Function

Public Sub Reset() Implements System.Collections.IEnumerator.Reset
    myAktuellesDatum = myStartDatum
End Sub
End Class

```

Der Konstruktor und die beiden Eigenschaftsprozeduren sollen hier nicht so sehr interessieren. Vielmehr von Interesse ist, dass diese Klasse eine weitere Schnittstelle namens `IEnumerator` einbindet, und sie stellt die eigentliche Aufzählungsfunktionalität zur Verfügung. Sie muss dazu die Eigenschaft `Current`, die Funktion `MoveNext` sowie die Methode `Reset` implementieren.

Wenn eine `For/Each`-Schleife durchlaufen wird, dann wird das aktuell bearbeitete Objekt der Klasse durch die `Current`-Eigenschaft des eigentlichen Enumerators ermittelt. Anschließend zeigt `For/Each` mit dem Aufruf der Funktion `MoveNext` dem Enumerator an, dass es auf das nächste Objekt zugreifen möchte. Erst wenn `MoveNext` mit `False` als Rückgabewert anzeigt, dass es keine weiteren Objekte mehr zur Verfügung stellen kann (oder will), ist die umgebende `For/Each`-Schleife beendet.

In unserem Beispiel müssen wir bei `MoveNext` nur dafür sorgen, dass unsere interne Datums-Zähl-Variable um den Wert erhöht wird, den wir bei ihrer Instanzierung als Schrittweite bestimmt haben. Hat die Addition der Schrittweite auf diesen Datumszähler den Endwert noch nicht überschritten, liefern wir `True` als Funktionsergebnis zurück – `For/Each` darf mit seiner Arbeit fortfahren. Ist das

Datum allerdings größer als der Datums-Endwert, wird die Schleife abgebrochen – der Vorgang wird beendet.

Das Programm, das von dieser Klasse Gebrauch machen kann, lässt sich nun sehr elegant einsetzen, wie die folgenden Beispielcodezeilen zeigen:

```
Module Enumerators
    Sub Main()
        Dim locDatumsaufzählung As New Datumsaufzählung(#12/24/2004#, _
                                                       #12/31/2004#, _
                                                       New TimeSpan(1, 0, 0, 0))
        For Each d As Date In locDatumsaufzählung
            Console.WriteLine("Datum in Aufzählung: {0}", d)
        Next
    End Sub
End Module
```

Das Ergebnis sehen Sie anschließend in Form einer Datumsfolge im Konsolenfenster.

## Grundsätzliches zu Auflistungen

Arrays haben in .NET einen entscheidenden Nachteil. Sie können zwar dynamisch zur Laufzeit vergrößert oder verkleinert werden, aber der Programmieraufwand dazu ist doch eigentlich recht aufwändig. Wenn Sie sich schon früher mit Visual Basic beschäftigt haben, dann ist Ihnen »Auflistung« sicherlich ein Begriff.

Auflistungen erlauben es dem Entwickler, Elemente genau wie Arrays zu verwalten. Im Unterschied zu Arrays wachsen Auflistungen jedoch mit Ihren Speicherbedürfnissen.

Damit ist aber auch klar, dass das Indizieren mit Nummern zum Abrufen der Elemente nur bedingt funktionieren kann. Wenn ein Array 20 Elemente hat, und Sie möchten das 21. Element hinzufügen, dann können Sie das dem Array nicht einfach so mitteilen. Ganz anders bei Auflistungen: Hier fügen Sie ein Element mit der Add-Methode hinzu.

Intern werden (fast alle) Auflistungen ebenfalls wie (oder besser: als) Arrays verwaltet. Rufen Sie eine neue Auflistung ins Leben, dann hat dieses Array, wenn nichts anderes gesagt wird, eine Größe von 16 Elementen. Wenn die Auflistungsklasse später, sobald Ihr Programm richtig »in Action« ist, »merkt«, dass ihr die Puste mengentechnisch ausgeht, dann legt sie ein Array nunmehr mit 32 Elementen an, kopiert die vorhandenen Elemente in das neue Array, arbeitet fortan mit dem neuen Array und tut ansonsten so, als wäre nichts gewesen.

Dieser anfängliche Load-Faktor erhöht sich bei jedem Neuanlegen des Arrays, um die Kopiervorgänge zu minimieren. Man geht einfach davon aus, dass wenn der anfängliche Load-Faktor von 16 Elementen nicht ausreicht, 32 beim nächsten Mal auch zu wenig sind – und gerade in unserem Beispiel ist das ja auch richtig. So sind es beim 2. Mal bereits 32 Elemente, die dazukommen, beim nächsten Mal 64 usw. bis der maximale Load-Faktor mit 2048 Elementen erreicht ist.

In etwa entspricht also die Grundfunktionsweise einer Auflistung stark vereinfacht der folgenden Klasse, die Ihnen in den vergangenen Kapiteln schon mehrfach begegnet ist:

```

Class DynamicList
    Implements IEnumerable

    Protected myStepIncreaser As Integer
    Protected myCurrentArraySize As Integer
    Protected myCurrentCounter As Integer
    Protected myArray() As Object

    Sub New()
        MyClass.New(16)
    End Sub

    Sub New(ByVal StepIncreaser As Integer)
        myStepIncreaser = StepIncreaser
        myCurrentArraySize = myStepIncreaser
        ReDim myArray(myCurrentArraySize)
    End Sub

    Sub Add(ByVal Item As Object)

        'Prüfen, ob aktuelle Arraygrenze erreicht wurde
        If myCurrentCounter = myCurrentArraySize - 1 Then
            'Neues Array mit mehr Speicher anlegen,
            'und Elemente hinüberkopieren. Dazu:

            'Neues Array wird größer:
            myCurrentArraySize += myStepIncreaser

            'temporäres Array erstellen
            Dim locTempArray(myCurrentArraySize - 1) As Object

            'Elemente kopieren
            'Wichtig: Um das Kopieren müssen Sie sich,
            'anders als bei VB6, selber kümmern!
            Array.Copy(myArray, locTempArray, myArray.Length)

            'temporäres Array dem Memberarray zuweisen
            myArray = locTempArray

            'Beim nächsten Mal werden mehr Elemente reserviert!
            myStepIncreaser *= 2
        End If

        'Element im Array speichern
        myArray(myCurrentCounter) = Item

        'Zeiger auf nächstes Element erhöhen
        myCurrentCounter += 1
    End Sub

```

```

'Liefert die Anzahl der vorhandenen Elemente zurück
Public Overridable ReadOnly Property Count() As Integer
    Get
        Return myCurrentCounter
    End Get
End Property

'Erlaubt das Zuweisen und Abfragen
Default Public Overridable Property Item(ByVal Index As Integer) As Object
    Get
        Return myArray(Index)
    End Get

    Set(ByVal Value As Object)
        myArray(Index) = Value
    End Set
End Property

'Liefert den Enumerator der Basis (dem Array) zurück
Public Function GetEnumerator() As System.Collections.IEnumerator Implements
System.Collections.IEnumerable.GetEnumerator
    Return myArray.GetEnumerator
End Function
End Class

```

Nun könnte man meinen, diese Vorgehensweise könnte sich zu einer Leistungsproblematik entwickeln, da alle paar Elemente der komplette Array-Inhalt kopiert werden muss.

---

**BEGLEITDATEIEN:** Im Verzeichnis *VB 2005 - Entwicklerbuch\E - Datentypen\Kap19\DynamicList\* finden Sie das Projekt *DynamicList*, das Ihnen das Gegenteil beweist:

---

Wie lange, glauben Sie, dauert es, ein Array mit 200.000 Zufallszahlen (mit Nachkommastellen) auf diese Weise anzulegen? 3 Sekunden? 2 Sekunden? Finden Sie es selbst heraus, indem Sie das Programm starten:

Anlegen von 200000 zufälligen Double-Elementen...  
...in 20 Millisekunden!

Gerade mal 20 Millisekunden benötigt das Programm für diese Operation<sup>4</sup> – beeindruckend, wie schnell Visual Basic dieser Tage ist, finden Sie nicht?

Nun muss ich Ihnen leider verraten: Die Klasse *DynamicList* werden Sie nie benötigen. Bestandteil des Frameworks ist nämlich eine Klasse, die das schon kann. Sogar noch ein kleines bisschen schneller. Und: Sie hat zusätzlich noch andere Möglichkeiten, die Ihnen die selbst gestrickte Klasse nicht bietet.

Im Beispielprojekt finden Sie eine Sub *Beispiel2*, die Ihnen die gleiche Prozedur mit der Klasse *ArrayList* demonstriert:

---

<sup>4</sup> Auf einem Athlon 4400+.

```

Sub Beispiel2()
    Dim locZeitmesser As New HighSpeedTimeGauge
    Dim locAnzahlElemente As Integer = 50000
    Dim locDynamicList As New ArrayList
    Dim locRandom As New Random(Now.Millisecond)

    Console.WriteLine("Anlegen von {0} zufälligen Double-Elementen...", locAnzahlElemente)
    locZeitmesser.Start()
    For count As Integer = 1 To locAnzahlElemente
        locDynamicList.Add(locRandom.NextDouble * locRandom.Next)
    Next
    locZeitmesser.Stop()
    Console.WriteLine("...in {0:#,##0} Millisekunden!", locZeitmesser.DurationInMilliSeconds)
    Console.ReadLine()
End Sub

```

Dessen Geschwindigkeit ist auch nicht von schlechten Eltern:

Anlegen von 50000 zufälligen Double-Elementen...  
...in 8 Millisekunden!

Im Prinzip arbeitet `ArrayList` nach dem gleichen Verfahren, das Sie in `DynamicList` kennen gelernt haben. `ArrayList` verfährt auch mit dem gleichen Trick, um möglichst viel Leistung herauszuholen. Es verdoppelt die jeweils nächste Größe des neuen Arrays im Vergleich zu der vorherigen Größe des Arrays. Damit reduziert sich der Gesamtaufwand des Kopierens erheblich. Da die Methode aber Bestandteil des Frameworks ist, muss sie nicht zur Laufzeit »geJITted« werden, was der Geschwindigkeit zusätzlich zugute kommt.

Im Übrigen werden die Daten der einzelnen Elemente ja nicht wirklich bewegt. Lediglich die Zeiger auf die Daten werden kopiert – vorhandene Elemente bleiben im Managed Heap an ihrem Platz.

## Die wichtigen Auflistungen der Base Class Library

Die BCL des Frameworks enthält eine ganze Reihe von Auflistungs-Typen, von denen Sie einen der wichtigsten – `ArrayList` – schon im Einsatz gesehen haben. In diesem Abschnitt möchte ich Ihnen die wichtigen dieser Auflistungen kurz vorstellen und darauf hinweisen, für welchen Einsatz sie am besten geeignet sind oder welche Besonderheiten Sie bei ihrem Gebrauch beachten sollten. Für eine genauere Beschreibung ihrer Eigenschaften und Methoden verwenden Sie bitte die Online-Hilfe von Visual Studio.

### **ArrayList – universelle Ablage für Objekte**

`ArrayList` können Sie als Container für Objekte aller Art verwenden. Sie instanzieren ein `ArrayList`-Objekt und weisen ihm mithilfe seiner `Add`-Funktion das jeweils nächste Element zu. Mit der `Default`-Eigenschaft `Item` können Sie schon vorhandene Elemente abrufen oder neu definieren. `AddRange` erlaubt Ihnen, die Elemente einer vorhandenen `ArrayList` einer anderen `ArrayList` hinzuzufügen.

Mit der `Count`-Eigenschaft eines `ArrayList`-Objektes finden Sie heraus, wie viele Elemente es beherbergt.

`Clear` löscht alle Elemente einer `ArrayList`. Mit `Remove` löschen Sie ein Objekt aus der `ArrayList`, das Sie der `Remove`-Methode als Parameter übergeben. Wenn mehrere gleiche Objekte (die `Equals`-Methode jedes Objektes wird dabei verwendet) existieren, wird das erste gefundene Objekt gelöscht. Mit `RemoveAt` löschen Sie ein Element an einer bestimmten Position. `RemoveRange` erlaubt Ihnen schließlich, einen ganzen Bereich von `ArrayList`-Elementen ab einer bestimmten Position im `ArrayList`-Objekt zu löschen.

`ArrayList`-Objekte können in einfache Arrays umgewandelt werden. Dabei ist jedoch einiges zu beachten: Die Elemente der `ArrayList` müssen ausnahmslos alle dem Typ des Arrays entsprechen, in den sie umgewandelt werden sollen. Die Konvertierung nehmen Sie mit der `ToArrayList`-Methode des entsprechenden `ArrayList`-Objektes vor. Dabei bestimmen Sie, wenn Sie in ein typdefiniertes Array (wie `Integer()` oder `String()`) umwandeln, den Grundtyp (nicht den Arraytyp!) als zusätzlichen Parameter. Wenn Sie ein Array in eine `ArrayList` umwandeln wollen, verwenden Sie den entsprechenden Konstruktor der `ArrayList` – die entsprechende Konstruktorroutine nimmt die Konvertierung in eine `ArrayList` anschließend vor.

---

**HINWEIS:** Bitte schauen Sie sich dazu auch das weiter unten gezeigte Listing an, insbesondere was die Konvertierungshinweise von `ArrayList`-Objekten in Arrays betrifft.

---

`ArrayList` implementiert die Schnittstelle `IEnumerable`. Aus diesem Grund stellt die Klasse einen Enumerator zur Verfügung, mit dem Sie mithilfe von `For/Each` durch die Elemente der `ArrayList` iterieren können. Beachten Sie dabei, den richtigen Typ für die Schleifenvariable zu verwenden. `ArrayList`-Elemente sind nicht typsicher, und eine Typ-Verletzung ist nur dann ausgeschlossen, wenn Sie genau wissen, welche Typen gespeichert sind (das nachfolgende Beispiel demonstriert diesen Fehler recht anschaulich):

Hier die Beispiele, die das gerade Gesagte näher erläutern und den Code dazu dokumentieren:

---

**HINWEIS:** Sie finden die im Folgenden besprochenen Codeausschnitte im Verzeichnis `.\VB 2005 - Entwicklerbuch\E-Datentypen\Kap19\CollectionsDemo`.

---

```
Sub ArrayListDemo()
    Dim locMännerNamen As String() = {"Jürgen", "Uwe", "Klaus", "Christian", "José"}
    Dim locFrauenNamen As New ArrayList
    Dim locNamen As New ArrayList

    'ArrayList aus vorhandenem Array/ArrayList erstellen.
    locNamen = New ArrayList(locMännerNamen)

    'ArrayList mit Add definieren.
    locFrauenNamen.Add("Ute") : locFrauenNamen.Add("Miriam")
    locFrauenNamen.Add("Melanie") : locFrauenNamen.Add("Anja")
    locFrauenNamen.Add("Stephanie") : locFrauenNamen.Add("Heidrun")

    'ArrayList einer anderen ArrayList hinzufügen:
    locNamen.AddRange(locFrauenNamen)

    'ArrayList in eine ArrayList einfügen.
    Dim locHundenamen As String() = {"Hasso", "Bello", "Wauzi", "Wuffi", "Basko", "Franz"}
    'Einfügen *vor* dem 6. Element
```

```

locNamen.InsertRange(5, locHundenamen)

'ArrayList ein Array zurückwandeln.
Dim locAlleNamen As String()

'Vorsicht: Fehler!
'locAlleNamen = DirectCast(locNamen.ToArray, String())

'Vorsicht: Ebenfalls Fehler!
'locAlleNamen = DirectCast(locNamen.ToArray(GetType(String)), String())

'So ist es richtig.
locAlleNamen = DirectCast(locNamen.ToArray(GetType(String)), String())

'Repeat legt eine ArrayList aus wiederholten Items an.
locNamen.AddRange(ArrayList.Repeat("Dublettenname", 10))

'Ein Element im Array mit der Item-Eigenschaft ändern.
locNamen(10) = "Fiffi"
'Mit der Item-Eigenschaft geht es auch:
locNamen.Item(13) = "Miriam"

'Löschen des ersten zutreffenden Elementes aus der Liste.
locNamen.Remove("Basko")

'Löschen eines Elementes an einer bestimmten Position.
locNamen.RemoveAt(4)

'Löschen eines bestimmten Bereichs aus der ArrayList mit RemoveRange.
'Count ermittelt die Anzahl der Elemente in der ArrayList
locNamen.RemoveRange(locNamen.Count - 6, 5)

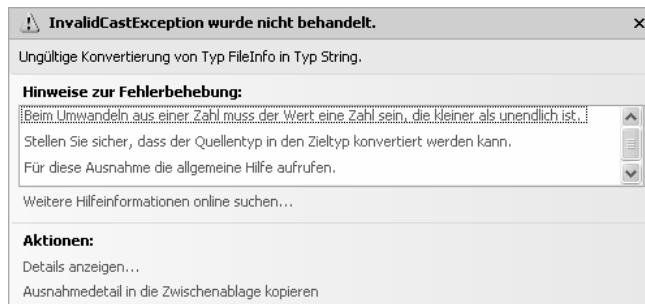
'Ausgeben der Elemente über die Default-Eigenschaft der ArrayList (Item).
For i As Integer = 0 To locNamen.Count - 1
    Console.WriteLine("Der Name Nr. {0} lautet {1}", i, locNamen(i).ToString)
Next

'Anderes als ein String-Objekt der ArrayList hinzufügen,
'um den folgenden Fehler "vorzubereiten".
locNamen.Add(New FileInfo("C:\TEST.TXT"))

'Diese Schleife kann nicht bis zum Ende ausgeführt werden,
'da ein Objekt nicht vom Typ String mit von der Partie ist!
For Each einString As String In locNamen
    'Hier passiert irgendetwas mit dem String.
    'nicht von Interesse, deswegen kein Rückgabewert.
    einString.EndsWith("Peter")
Next
Console.ReadLine()
End Sub

```

Wenn Sie dieses Beispiel laufen lassen, dann sehen Sie zunächst die erwarteten Ausgaben auf dem Bildschirm. Doch der Programmcode erreicht nie die Anweisung `Console.ReadLine`, um auf Ihre letzte Bestätigung zu warten. Stattdessen löst er eine Ausnahme aus, etwa wie in Abbildung 19.2 zu sehen.



**Abbildung 19.2:** Achten Sie beim Iterieren mit `For/Each` durch eine Auflistung darauf, dass die Elemente dem SchleifenvariablenTyp entsprechen, um solche Ausnahmen zu vermeiden!

Der Grund dafür: Das letzte Element in der `ArrayList` ist kein `String`. Aus diesem Grund wird die Ausnahme ausgelöst. Achten Sie deshalb stets darauf, dass die Schleifenvariable eines `For/Each`-Konstrukts immer den Typen entspricht, die in einer Auflistung gespeichert sind.

## Typsichere Auflistungen auf Basis von `CollectionBase`

Um den im vorherigen Abschnitt aufgetretenen Fehler definitiv zu vermeiden, sollte eine Auflistung in der Lage sein, nur Elemente bestimmten Typs zu definieren und zurückzuliefern. Die `ArrayList`-Klasse kann diese Arbeit nicht leisten, weil sie – beispielsweise mit der `Add`-Methode – `Objects` entgegennimmt und damit für alle Objekttypen zugänglich ist.

Gerade wenn Sie Softwareentwicklung im Team betreiben, ist es sicherer, dass Sie den Entwicklern, die mit Ihren Klassen arbeiten, nur solche zur Verfügung stellen, die narrensicher zu bedienen sind. Haben Sie beispielsweise einen Klassensatz entwickelt, der Adressendetails in einer `Adresse`-Klasse und mehrere Adressen in einer `Adressen`-Auflistung verwaltet, dann sollte Letztere auch nur Daten vom Typ `Adresse` entgegennehmen. Sie sollte damit typsicher sein.

Mit dem Framework der Version 2.0 gibt es grundsätzlich zwei Möglichkeiten, dabei Typsicherheit zu garantieren. Sie können die Klasse von der generischen Klasse `Collection(Of )` bzw. `List(Of )` ableiten – und haben die Aufgabe erledigt. Das anschließende ► Kapitel 20 verrät Ihnen mehr zu dieser Möglichkeit.

Um Kompatibilität mit Framework-Versionen zu wahren, die generische Klassen noch nicht unterstützen (1.0, 1.1) und um ein weiteres, wirklich wichtiges Beispiel für die klassische OOP-Programmierung kennen zu lernen, lesen Sie den folgenden Abschnitt.

Das Framework bietet zu diesem Zweck eine abstrakte Klasse namens `CollectionBase` als Vorlage an, die Sie für diese Zwecke ableiten und erweitern können.

---

**BEGLEITDATEIEN:** Sie finden die im Folgenden besprochenen Codeauszüge im Verzeichnis `.\VB 2005 - Entwicklerbuch\E - Datentypen\Kap19\TypeSafeCollections` im Verzeichnis der CD zum Buch.

---

Der folgende Code zeigt, wie die `Item`-Methode und die `Add`-Methode in einer benutzerdefinierten Auflistung realisiert sind:

```

Public Class Adressen
    Inherits CollectionBase

    Public Overridable Function Add(ByVal Adr As Adresse) As Integer
        Return MyBase.List.Add(Adr)
    End Function

    Default Public Overridable Property Item(ByVal Index As Integer) As Adresse
        Get
            Return DirectCast(MyBase.List(Index), Adresse)
        End Get
        Set(ByVal Value As Adresse)
            MyBase.List(Index) = Value
        End Set
    End Property
End Class

```

Das entsprechende Beispielprogramm, das die Klasse testet, sieht folgendermaßen aus:

Module TypesafeCollections

```

Sub Main()
    Dim locAdressen As New Adressen
    Dim locAdresse As New Adresse("Christian", "Sonntag", "99999", "Trinken")
    Dim locAndererTyp As New FileInfo("C:\Test.txt")

    'Kein Problem:
    locAdressen.Add(locAdresse)

    'Schon der Editor mault!
    'locAdressen.Add(locAndererTyp)

    'Auch kein Problem.
    locAdresse = locAdressen(0)

    For Each eineAdresse As Adresse In locAdressen
        Console.WriteLine(eineAdresse)
    Next
    Console.ReadLine()
End Sub

End Module

```

Sie sehen anhand des Testprogramms, dass Typsicherheit in dieser Auflistung gewährleistet ist. Allerdings sollte dieses Programm, wenn Sie das Klassen-Kapitel dieses Buches aufmerksam studiert haben, auch Fragen aufwerfen: Wenn Sie sich die Beschreibung zu `CollectionBase` in der Online-Hilfe ansehen, finden Sie dort folgenden Prototyp:

```

<Serializable>
MustInherit Public Class CollectionBase
    Implements IList, ICollection, IEnumerable
    .
    .
    .

```

CollectionBase bindet unter anderem die Schnittstelle IList ein. Die Schnittstelle IList schreibt vor, dass eine Add-Funktion mit der folgenden Signatur in der Klasse implementiert sein muss, die sie implementiert:

```
Function Add(ByVal value As Object) As Integer
```

Erkennen Sie die Unregelmäßigkeit? Unsere Klasse erbt von CollectionBase, CollectionBase bindet IList ein, IList verlangt die Methode Add, in unserer Klasse müsste demzufolge eine Add-Methode vorhanden sein, die wir zu überschreiben hätten! Aber sie ist es nicht, und das ist auch gut so, denn: Wäre sie vorhanden, dann müssten wir ihre Signatur zum Überschreiben übernehmen. Als Parameter nimmt sie aber ein Object entgegen – unsere Typsicherheit wäre dahin. Alternativ könnten wir sie überladen, aber in diesem Fall könnte man ihr dennoch ein Object übergeben – wieder wäre es aus mit der Typsicherheit.

Wir kommen dem Geheimnis auf die Spur, wenn wir uns die Add-Funktion unserer neuen Adressen-Auflistung ein wenig genauer anschauen. Dort heißt es:

```
Public Overridable Function Add(ByVal Adr As Adresse) As Integer
    Return MyBase.List.Add(Adr)
End Function
```

MyBase greift auf die Basisklasse zurück und verwendet letzten Endes die Add-Funktion des Objektes, das die List-Eigenschaft zurückliefert, um das hinzuzufügende Element weiterzureichen. Was aber macht List? Welches Objekt liefert es zurück? Um Sie vielleicht zunächst komplett zu verwirren: List liefert die Instanz unserer Klasse zurück<sup>5</sup> – und wir betreiben hier Polymorphie in Vollendung!

Schauen wir, was die Online-Hilfe von Visual Studio zur Beschreibung von List zu sagen hat: »Ruft eine IList mit der Liste der Elemente in der CollectionBase-Instanz ab.« – Wow, das ist informativ!

Der ganze Umstand wird klar, wenn Sie sich das folgende Konstrukt anschauen. Behalten Sie dabei im Hinterkopf, dass eine Möglichkeit gefunden werden muss, die Add-Funktion einer Schnittstelle in einer Klasse zu implementieren, ohne einer sie einbindenden Klasse die Möglichkeit zu nehmen, eine Add-Funktion mit einer ganz anderen Signatur zur Verfügung zu stellen:

```
Interface ITest
    Function Add(ByVal obj As Object) As Integer
End Interface

MustInherit Class ITestKlasse
    Implements ITest

    Private Function ITestAdd(ByVal obj As Object) As Integer Implements ITest.Add
        Trace.WriteLine("ITestAdd:" + obj.ToString())
    End Function

End Class
```

---

<sup>5</sup> Dieses Konstrukt erinnerte mich spontan an eine Star-Trek-Voyager-Episode, in der Tom Paris mit B'Elanna Torres das Holodeck besucht, um mit ihr in einem holografisch projizierten Kino einen »altägyptischen« Film mit 3D-Brille zu sehen. Ihr Kommentar dazu: „Lass mich das mal klarstellen: Du hast diesen Aufwand betrieben, um eine dreidimensionale Umgebung so zu programmieren, das sie ein zweidimensionales Bild projiziert, und nun bittest Du mich, diese Brille zu tragen, damit es wieder dreidimensional ausschaut?« ...

```

Class ITestKlasseAbleitung
    Inherits ITestKlasse

    Public ReadOnly Property Test() As ITest
        Get
            Return DirectCast(Me, ITest)
        End Get
    End Property

    Public Sub Add(ByVal TypeSafe As Adresse)
        Test.Add(TypeSafe)
    End Sub
End Class

```

Die Schnittstelle dieser Klasse schreibt vor, dass eine Klasse, die diese Schnittstelle einbindet, eine Add-Funktion implementieren muss. Die abstrakte Klasse `ITestKlasse` bindet diese Schnittstelle auch ordnungsgemäß ein – allerdings stellt sie die Funktion nicht der Öffentlichkeit zur Verfügung; die Funktion ist dort nämlich als privat definiert. Außerdem – und das ist der springende Punkt – nennt sich die Funktion nicht `Add`, sondern `ITestAdd` – möglich wird das, da Schnittstelleneinbindungen in Visual Basic grundsätzlich explizit funktionieren, genauer gesagt durch das Schlüsselwort `Implements` am Ende der Funktionsdeklaration.

Halten wir fest: Wir haben nun eine saubere Schnittstellenimplementierung, *und* wir können die der Schnittstelle zugeordnete Funktion von außen nicht sehen. Wie rufen wir die Funktion `IListAdd` der Basisklasse `ITestKlasse` dann überhaupt auf? Der Rätsels Lösung ist: Die Funktion ist nicht ganz so privat, wie sie scheint. Denn wir können über eine Schnittstellenvariable auf sie zugreifen. Wenn wir die eigene Instanz der Klasse in eine Schnittstellenvariable vom Typ `ITest` casten, dann können wir über die Schnittstellenfunktion `ITest.Add` dennoch auf die private Funktion `ITestAdd` der Klasse `ITestKlasse` zugreifen – und wir sind am Ziel!

Zu unserer Bequemlichkeit stellt die Klasse `ITestKlasse` bereits eine Eigenschaft (`Property Test`) zur Verfügung, die die aktuelle Klasseninstanz als `ITest`-Schnittstelle zurück liefert. Wir können uns dieser also direkt bedienen.

Genau das Gleiche haben wir in unserer aus `CollectionBase` entstandenen Klasse `Adressen` gemacht – um zum alten Beispiel zurückzukommen. Die `List`-Eigenschaft der Klasse `CollectionBase` entspricht der `Test`-Eigenschaft der Klasse `ITestKlasse` unseres Erklärungsbeispiels.

## Hashtables – für das Nachschlagen von Objekten

Hashtable-Objekte sind das ideale Werkzeug, wenn Sie eine Datensammlung aufbauen wollen, aber die einzelnen Objekte nicht durch einen numerischen Index, sondern durch einen Schlüssel abrufen wollen. Ein Beispiel soll das verdeutlichen.

Angenommen, Sie haben eine Adressenverwaltung programmiert, bei der Sie einzelne Adressen durch eine Art Matchcode abrufen wollen (Kundennummer, Lieferantennummer, was auch immer). Bei der Verwendung einer `ArrayList` müssten Sie schon einen Aufwand betreiben, um an ein Array-Element auf Basis des Matchcode-Namens zu gelangen: Sie müssten zum Finden eines Elements in der Liste für die verwendete Adressklasse eine `CompareTo`-Methode implementieren, damit die Liste mittels `Sort` sortiert werden könnte. Anschließend könnten Sie mit `BinarySearch` das Element finden –

vorausgesetzt, die CompareTo-Methode würde eine Instanz der Adressklasse über ihren Matchcode-Wert vergleichen.

Hashtable-Objekte vereinfachen ein solches Szenario ungemein: Wenn Sie einer Hashtable ein Objekt hinzufügen, dann nimmt dessen Add-Methode nicht nur das zu speichernde Objekt (den hinzuzufügenden Wert) entgegen, sondern auch ein weiteres. Dieses zusätzliche Objekt (das genau genommen als erster Parameter übergeben wird) stellt den Schlüssel – den Key – zum Wiederauffinden des Objektes dar. Sie rufen ein Objekt aus einer Hashtable anschließend nicht wie bei der ArrayList mit

```
Element = eineArrayList(5)
```

ab, sondern mit dem entsprechenden Schlüssel, etwa:

```
Element = eineHashtable("ElementKey")
```

Voraussetzung bei diesem Beispiel ist natürlich, dass der Schlüssel zuvor ein entsprechender String gewesen ist.

## Anwenden von Hashtables

---

**BEGLEITDATEIEN:** Im Verzeichnis .\VB 2005 - Entwicklerbuch\E - Datentypen\Kap19\ HashtableDemo01\ des Verzeichnisses der CD zum Buch finden Sie ein Beispielprojekt, das den Einsatz der Hashtable-Klasse demonstrieren soll. Es enthält eine Klasse, die eine Datenstruktur – eine Adresse – abbildet. Eine statische Funktion erlaubt darüber hinaus, eine beliebige Anzahl von Zufallsadressen zu erstellen, die zunächst in einer ArrayList gespeichert werden. Eine solche mit zufälligen Daten gefüllte ArrayList soll zum Experimentieren mit der Hashtable-Klasse dienen. Der Vollständigkeit halber (und des späteren einfacheren Verständnisses) möchte ich Ihnen diese Adresse-Klasse kurz vorstellen – wenn auch in verkürzter Form):

---

---

**WICHTIG:** Sie benötigen mindestens 1 GByte-Arbeitsspeicher, damit Sie die folgenden Hashtable-Beispiele nachvollziehen können. Die Menge des zu reservierenden Speichers für das Beispiel ist deshalb so groß, weil durch die hohen Prozessorgeschwindigkeiten eine genügend große Anzahl von Elementen vorhanden sein muss, um überhaupt Geschwindigkeitsvergleiche anstellen zu können.

---

```
Public Class Adresse
    'Member-Variablen, die die Daten halten:
    Protected myMatchcode As String
    Protected myName As String
    Protected myVorname As String
    Protected myPLZ As String
    Protected myOrt As String

    'Konstruktor - legt eine neue Instanz an.
    Sub New(ByVal Matchcode As String, ByVal Name As String, ByVal Vorname As String, _
            ByVal Plz As String, ByVal Ort As String)
        myMatchcode = Matchcode
        myName = Name
        myVorname = Vorname
        myPLZ = Plz
        myOrt = Ort
    End Sub
```

```

'Mit Region ausgeblendet:
'die Eigenschaften der Klasse, um die Daten offen zu legen.
#Region "Eigenschaften"
.

.

.

#End Region
Public Overrides Function ToString() As String
    Return Matchcode + ":" + Name + ", " + Vorname + ", " + PLZ + " " + Ort
End Function

Public Shared Function ZufallsAdressen(ByVal Anzahl As Integer) As ArrayList
    Dim locArrayList As New ArrayList(Anzahl)
    Dim locRandom As New Random(Now.Millisecond)

    Dim locNachnamen As String() = {"Heckhuis", "Löffelmann", "Thiemann", "Müller", _
        "Meier", "Tiemann", "Sonntag", "Ademmer", "Westermann", "Vüllers", _
        "Hollmann", "Vielstedde", "Weigell", "Weichel", "Weichelt", "Hoffmann", _
        "Rode", "Trouw", "Schindler", "Neumann", "Jungemann", "Hörstmann", _
        "Tinoco", "Albrecht", "Langenbach", "Braun", "Plenge", "Englisch", _
        "Clarke"}

    Dim locVornamen As String() = {"Jürgen", "Gabriele", "Uwe", "Katrín", "Hans", _
        "Rainer", "Christian", "Uta", "Michaela", "Franz", "Anne", "Anja", _
        "Theo", "Momo", "Katrín", "Guido", "Barbara", "Bernhard", "Margarete", _
        "Alfred", "Melanie", "Britta", "José", "Thomas", "Daja", "Klaus", "Axel", _
        "Lothar", "Gareth"}

    Dim locStädte As String() = {"Wuppertal", "Dortmund", "Lippstadt", "Soest", _
        "Liebenburg", "Hildesheim", "München", "Berlin", "Rheda", "Bielefeld", _
        "Braunschweig", "Unterschleißheim", "Wiesbaden", "Straubing", _
        "Bad Waldliesborn", "Lippetal", "Stirpe", "Erwitte"}

    For i As Integer = 1 To Anzahl
        Dim locName, locVorname, locMatchcode As String
        locName = locNachnamen(locRandom.Next(locNachnamen.Length - 1))
        locVorname = locVornamen(locRandom.Next(locVornamen.Length - 1))
        locMatchcode = locName.Substring(0, 2)
        locMatchcode += locVorname.Substring(0, 2)
        locMatchcode += i.ToString("00000000")
        locArrayList.Add(New Adresse( _
            locMatchcode, _
            locName, _
            locVorname, _
            locRandom.Next(99999).ToString("00000"), _
            locStädte(locRandom.Next(locStädte.Length - 1)))))
    Next
    Return locArrayList
End Function

```

```

Shared Sub AdressenAusgeben(ByVal Adressen As ArrayList)
    For Each Item As Object In Adressen
        Console.WriteLine(Item)
    Next
End Sub

End Class

```

Die eigentliche Klasse zum Speichern einer Adresse ist Kinderkram. Wichtig ist, dass Sie wissen, welche besondere Bewandtnis es mit dem Matchcode einer Adresse hat. Ein Matchcode ist in einem Satz von Adressen immer eindeutig. Die Prozedur in diesem Beispiel, die zufällige Adressen erzeugt, stellt das sicher. Der Matchcode setzt sich aus den ersten beiden Buchstaben des Nachnamens, den ersten beiden Buchstaben des Vornamens und einer fortlaufenden Nummer zusammen. Würden Sie 15 verschiedene Zufallsadressen mit diesem Code

```

Sub AdressenTesten()
    '15 zufällige Adressen erzeugen.
    Dim locDemoAdressen As ArrayList = Adresse.ZufallsAdressen(15)

    'Adressen im Konsolenfenster ausgeben.
    Console.WriteLine("Liste mit zufällig erzeugten Personendaten")
    Console.WriteLine(New String("=", 30))
    Adresse.AdressenAusgeben(locDemoAdressen)

End Sub

```

erstellen und ausgeben, sähen Sie etwa folgendes Ergebnis im Konsolenfenster:

```

Liste mit zufällig erzeugten Personendaten
=====
HeMo00000001: Heckhuis, Momo, 06549 Straubing
SoGu00000002: Sonntag, Guido, 21498 Liebenburg
ThA100000003: Thiemann, Alfred, 51920 Bielefeld
HöJü00000004: Hörstmann, Jürgen, 05984 Liebenburg
TiMa00000005: Tiemann, Margarete, 14399 München
TiAn00000006: Tiemann, Anja, 01287 Dortmund
ViGu00000007: Vielstedde, Guido, 72762 Wuppertal
RoMe00000008: Rode, Melanie, 94506 Hildesheim
TiDa00000009: Tiemann, Daja, 54134 Lippstadt
BrJo00000010: Braun, José, 14590 Soest
WeJü00000011: Westermann, Jürgen, 83128 Wuppertal
HeKa00000012: Heckhuis, Katrin, 13267 Bad Waldliesborn
TrJü00000013: Trouw, Jürgen, 54030 Lippstadt
PlGa00000014: Plenge, Gabriele, 97702 Braunschweig
WeJü00000015: Weichel, Jürgen, 39992 Unterschleißheim

```

Falls Sie sich nun fragen, wieso ich Ihnen soviel Vorgeplänkel erzähle: Es ist wichtig für die richtige Anwendung von Hashtable-Objekten und das richtige Verständnis, wie Elemente in Hashtables gespeichert werden. Denn wenn Sie ein Objekt in einer Hashtable speichern, muss der Schlüssel eindeutig sein – anderenfalls hätten Sie nicht die Möglichkeit, wieder an alle Elemente heranzukommen. (Welches von zwei Elementen sollte die Hashtable schließlich durch einen Schlüssel indiziert zurückliefern, wenn die Schlüssel die gleichen wären?)

## Verarbeitungsgeschwindigkeiten von Hashtables

Jetzt lassen Sie uns eine Hashtable in Aktion sehen und schauen, was sie zu leisten vermag. Aus den Elementen der ArrayList, die uns die ZufallsAdressen-Funktion liefert, bauen wir eine Hashtable mit nicht weniger als 500.000 Elementen auf. Gleichzeitig erzeugen wir ein weiteres Array mit 50 zufälligen Elementen der ArrayList und merken uns deren Matchcode. Diese Matchcodes verwenden wir anschließend, um uns Elemente aus der Liste herauszupicken und messen dabei die Zeit, die das Zugreifen benötigt. Das Programm dazu sieht folgendermaßen aus:

Module HashtableDemo

```
Sub Main()

    'AdressenTesten()
    'Console.ReadLine()
    'Return

    Dim locAnzahlAdressen As Integer = 1000000
    Dim locZugriffsElemente As Integer = 2
    Dim locMessungen As Integer = 3
    Dim locZugriffe As Integer = 1000000
    Dim locVorlageAdressen As ArrayList
    Dim locAdressen As New Hashtable
    Dim locTestKeys(locZugriffsElemente) As String
    Dim locZeitmesser As New HighSpeedTimeGauge
    Dim locRandom As New Random(Now.Millisecond)

    'Warten auf Startschuss.
    Console.WriteLine("Drücken Sie Return, um zu beginnen", locZeitmesser.DurationInMilliSeconds)
    Console.ReadLine()

    'Viele Adressen erzeugen:
    Console.Write("Erzeugen von {0} zufälligen Adresseneinträgen...", locAnzahlAdressen)
    locZeitmesser.Start()
    locVorlageAdressen = adresse.ZufallsAdressen(locAnzahlAdressen)
    locZeitmesser.Stop()
    Console.WriteLine("fertig nach {0} ms", locZeitmesser.DurationInMilliSeconds)
    locZeitmesser.Reset()

    'Aufbauen der Hashtable.
    Console.Write("Aufbauen der Hashtable mit zufälligen Adresseneinträgen...", locAnzahlAdressen)
    locZeitmesser.Start()
    For Each adresse As Adresse In locVorlageAdressen
        locAdressen.Add(adresse.Matchcode, adresse)
    Next
    locZeitmesser.Stop()
    Console.WriteLine("fertig nach {0} ms", locZeitmesser.DurationInMilliSeconds)
    locZeitmesser.Reset()

    '51 zufällige Adressen rauspicken.
    For i As Integer = 0 To locZugriffsElemente
        locTestKeys(i) = DirectCast(locVorlageAdressen(locRandom.Next(locAnzahlAdressen)), _
            Adresse).Matchcode
    Next
End Sub
```

Next

```
Dim locTemp As Object
Dim locTemp2 As Object

'Zugreifen und Messen, wie lange das dauert,
'das ganze 5 Mal, um die Messung zu bestätigen.
For z As Integer = 1 To locMessungen
    Console.WriteLine()
    Console.WriteLine("{0}. Messung:", z)
    For i As Integer = 0 To locZugriffsElemente
        Console.Write("{0} Zugriffe auf: {1} in ", locZugriffe, locTestKeys(i))
        locTemp = locTestKeys(i)
        locZeitmesser.Start()
        For j As Integer = 1 To locZugriffe
            locTemp2 = locAdressen(locTemp)
        Next j
        locZeitmesser.Stop()
        locTemp = locTemp2.GetType
        Console.WriteLine("{0} ms", locZeitmesser.DurationInMilliseconds)
    Next

    'Zugriff auf ArrayList für Vergleich
    For i As Integer = 0 To locZugriffsElemente
        Console.Write("{0} Zugriffe auf ArrayList-Element in ", locZugriffe)
        locZeitmesser.Start()
        For j As Integer = 1 To locZugriffe
            locTemp2 = locVorlageAdressen(0)
        Next j
        locZeitmesser.Stop()
        locTemp = locTemp2.GetType
        Console.WriteLine("{0} ms", locZeitmesser.DurationInMilliseconds)
    Next

    Console.ReadLine()
End Sub
```

Wenn Sie das Programm starten, erzeugt es eine Ausgabe, etwa wie im folgenden Bildschirmauszug:

Drücken Sie Return, um zu beginnen

```
Erzeugen von 1000000 zufälligen Adresseneinträgen...fertig nach 3594 ms
Aufbauen der Hashtable mit zufälligen Adresseneinträgen...fertig nach 869 ms
```

1. Messung:

```
1000000 Zugriffe auf: NeAx00563508 in 212 ms
1000000 Zugriffe auf: PlCh00288965 in 213 ms
1000000 Zugriffe auf: VüBe00917935 in 208 ms
1000000 Zugriffe auf ArrayList-Element in 15 ms
1000000 Zugriffe auf ArrayList-Element in 14 ms
1000000 Zugriffe auf ArrayList-Element in 14 ms
```

2. Messung:

```
1000000 Zugriffe auf: NeAx00563508 in 209 ms
1000000 Zugriffe auf: PlCh00288965 in 214 ms
1000000 Zugriffe auf: VüBe00917935 in 224 ms
1000000 Zugriffe auf ArrayList-Element in 16 ms
1000000 Zugriffe auf ArrayList-Element in 18 ms
1000000 Zugriffe auf ArrayList-Element in 18 ms
```

3. Messung:

```
1000000 Zugriffe auf: NeAx00563508 in 209 ms
1000000 Zugriffe auf: PlCh00288965 in 208 ms
1000000 Zugriffe auf: VüBe00917935 in 211 ms
1000000 Zugriffe auf ArrayList-Element in 15 ms
1000000 Zugriffe auf ArrayList-Element in 15 ms
1000000 Zugriffe auf ArrayList-Element in 14 ms
```

Nach dem Programmstart legt das Programm hier im Beispiel 1.000.000 Testelemente an und baut daraus die Hashtable auf. Anschließend pickt es sich drei Beispieleinträge heraus und misst die Zeit, die es für 1.000.000 Zugriffe auf jeweils diese Elemente der Hashtable benötigt. Das Ganze macht es dreimal, um eine Konstanz in den Verarbeitungsgeschwindigkeiten sicherzustellen. Um im Gegenzug nachzuweisen, wie schnell ein Zugriff auf ein ArrayList-Element erfolgt, führt es eine Messung dazu anschließend durch.

### **Wieso die Zugriffszeit auf Hashtable-Elemente nahezu konstant ist ...**

Sie können die Parameter am Anfang des Programms verändern, um weitere Eindrücke der unglaublichen Geschwindigkeit von .NET zu sammeln. Sie werden bei allen Experimenten jedoch eines herausfinden: Ganz gleich, wie Sie auch an den Schrauben drehen, die Zugriffsgeschwindigkeit auf ein einzelnes Element bleibt nahezu konstant. In den seltensten Fällen benötigt der Zugriff auf ein Element das Doppelte der Durchschnittszeit – und diese Ausreißer sind vergleichsweise selten.

Das Geheimnis für die auf der einen Seite sehr hohe, auf der anderen Seite durchschnittlich gleich bleibende Geschwindigkeit beim Zugriff liegt am Aufbau der Hashtable und an der Behandlung der Schlüssel. Die schnellste Art und Weise, auf ein Element eines Arrays zuzugreifen, ist das direkte Auslesen des Elementes über seinen Index (auch das hat das vorherige Beispiel mit dem Zugriff auf die ArrayList-Elemente gezeigt). Daraus folgt, dass es am günstigsten ist, die Elemente der Hashtable nicht nach einem Schlüssel durchsuchen zu müssen, sondern die Positionsnummer eines Elementes der Hashtable irgendwie zu berechnen. Und genau hier setzt das *Hashing*-Konzept an. *Hashing* bedeutet eigentlich »zerhacken«. Das klingt sehr negativ, doch das Zerhacken des Schlüssels, das auf eine bestimmte Weise tatsächlich stattfindet, dient hier einem konstruktiven Zweck: Wenn der Schlüssel, der beispielsweise in Form einer Zeichenkette vorliegt, *gehashed* wird, geht daraus eine Kennzahl hervor, die Aufschluss über die Wertigkeit der Zeichenkette gibt. *Wertigkeit* in diesem Zusammenhang bedeutet, dass gleiche Zeichenketten nicht nur gleiche Hash-Werte ausweisen, sondern auch, dass größere Zeichenketten größere Hash-Werte bedeuten.

Ein einfaches Beispiel soll diese Zusammenhänge klarstellen: Angenommen, Sie haben 26 Wörter, die alle mit einem anderen Buchstaben beginnen. Diese Wörter liegen in unsortierter Reihenfolge vor. Nehmen wir weiter an, dass Ihr Hash-Algorithmus ganz einfach gestrickt ist: Der Hashcode einer Zeichenfolge entspricht der Nummer des Anfangsbuchstabens jedes Wortes. In dieser vereinfachten Konstellation haben Sie das Problem des Positionsberechnens bereits gelöst. Ihr Hashcode ist die

Indexnummer im Array; sowohl das Einsortieren als auch das spätere Auslesen funktioniert in Windeseile über die Zeichenfolge selbst (genauer über seinen Anfangsbuchstaben).

Das Problem gestaltet sich in der Praxis natürlich nicht ganz so einfach. Mehr Zeichen (um beim Beispiel Zeichenfolgen für Schlüssel zu bleiben) müssen berücksichtigt werden, um den Hash zu berechnen, und bei langen Zeichenketten und begrenzter Hashcode-Präzision kann man nicht ausschließen, dass unterschiedliche Zeichenketten gleiche Hashcodes ergeben. In diesem Fall müssen Kollisionen bei der Indexberechnung berücksichtigt werden. So groß gestaltet sich das Problem aber gar nicht, denn wenn beispielsweise beim Einsortieren der Elemente der sich durch den Hashcode des Schlüssels ergebende Index bereits belegt ist, nimmt man eben den nächsten freien.

Um beim Beispiel zu bleiben: Sie haben nun 26 Elemente, von denen alle mit einem anderen Anfangsbuchstaben beginnen, mit einer Unregelmäßigkeit: Sie haben zwei A-Wörter, ein B-Wort und kein C-Wort. Der Hash-Algorithmus bleibt der gleiche. Sie sortieren das B-Wort in Slot 2, anschließend das erste A-Wort in Slot 1. Nun bekommen Sie das zweite A-Wort zum Einsortieren, und laut Hashcode zeigt es auch auf Slot 1. In diesem Fall fangen Sie an zu suchen, und testen Slot 2, der durch das B-Wort schon belegt ist, aber Sie finden anschließend einen freien Slot 3. Der gehört eigentlich zu C, aber in diesem Fall ist er momentan nicht nur frei, sondern wird auch nie beansprucht werden, da es kein C-Wort in der einzusortierenden Liste gibt. Im ungünstigsten Fall gibt es in diesem Beispiel zwar ein C-Wort, dafür aber kein Z-Wort, und das zweite A-Wort wird als letztes Element eingesortiert. Jetzt verläuft die Suche über alle Elemente und landet schließlich auf dem letzten Elemente für das Z.

### **... und wieso Sie das wissen sollten!**

Sie können sich die Verminderung solcher Fälle mit zusätzlichem Speicherplatz erkaufen. Angenommen, Sie reservieren in unserem Beispiel doppelt so viel Speicher für die Schlüssel, dann sind zwar ganz viele Slots leer, aber das zweite A-Wort muss nicht den ganzen Weg bis zum Z-Index antreten. Was ganz wichtig ist: Sie wissen jetzt, was ein *Load-Faktor* ist. So nennt man nämlich den Faktor, der das Verhältnis zwischen Wahrscheinlichkeiten von Zuordnungskollisionen und benötigtem Speicher angibt. Weniger Kollisionswahrscheinlichkeit erfordert höheren Speicher und umgekehrt. Und Sie können diesen Zusammenhang auch am Beispielprogramm austesten.

---

**WICHTIG:** Um dieses Beispiel mit plausiblen Ergebnissen nachvollziehen zu können, benötigen Sie mindestens 1 GByte-Hauptspeicher (unter Windows XP). Andernfalls beginnt Ihr Computer während des Programmlaufs Speicher auf die Festplatte auszulagern, und dieser Vorgang kann die Messergebnisse natürlich eklatant verfälschen!

---

Am Anfang des Moduls in der Sub Main des Beispielprogramms finden Sie einen Block mit auskommentierten Deklarationsanweisungen. Tauschen Sie die Auskommentierung der beiden Blöcke, um folgende Parameter für den nächsten Versuch zu Grunde zu legen:

```
Dim locAnzahlAdressen As Integer = 1000000
Dim locZugriffsElemente As Integer = 25
Dim locMessungen As Integer = 3
Dim locZugriffe As Integer = 1000000
Dim locVorlageAdressen As ArrayList
Dim locAdressen As New Hashtable(100000, 1)
Dim locTestKeys(locZugriffsElemente) As String
Dim locZeitmesser As New HighSpeedTimeGauge
Dim locRandom As New Random(Now.Millisecond)
```

Mit diesem Block erhöhen wir die Anzahl der Elemente, die es zu testen gilt, auf 50, und damit erhöhen wir natürlich auch die Wahrscheinlichkeit, Elemente zu finden, die kollidieren. Gleichzeitig verändern wir – im Codelisting fett markiert – den Load-Faktor der Hashtable. Bei der Instanzierung einer Hashtable können Sie, neben der Anfangskapazität, als zweiten Parameter den Load-Faktor bestimmen. Gültig sind dabei Werte zwischen 0.1 und 1. Für den ersten Durchlauf bestimmen wir einen Load-Faktor von 1. Dabei wird auf Speicherplatz auf Kosten von vielen zu erwartenden Kollisionen und damit auf Kosten von Geschwindigkeit verzichtet (dieser Wert entspricht übrigens der Voreinstellung, die dann verwendet wird, wenn Sie keinen Parameter für den Load-Faktor angeben).

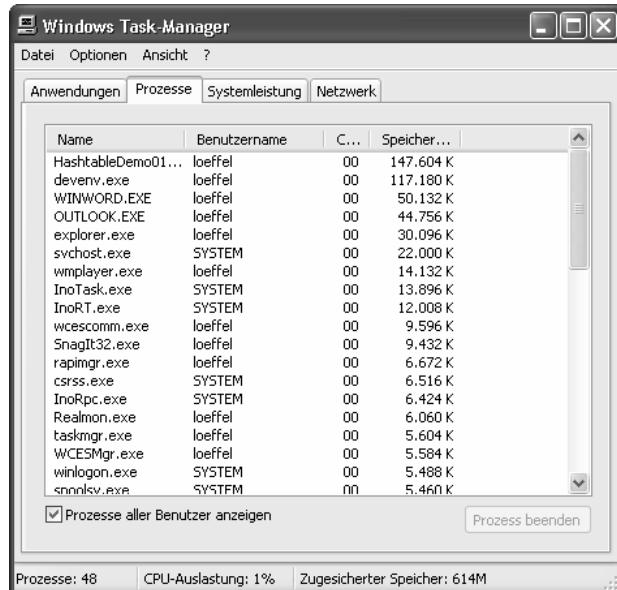
Starten Sie das Programm mit diesen Einstellungen, und beenden Sie es zunächst **nicht**, nachdem der Messdurchlauf abgeschlossen wurde! Je nach Leistungsfähigkeit Ihres Rechners sehen Sie bei der Ausgabe der Testergebnisse Messungen, von denen ich nur einige ausschnittsweise im Folgenden zeigen möchte:

```
1000000 Zugriffe auf: HoGa00919471 in 308 ms
1000000 Zugriffe auf: RoMi00603881 in 250 ms
1000000 Zugriffe auf: HeUw00854353 in 302 ms
1000000 Zugriffe auf: LaTh00018037 in 479 ms
1000000 Zugriffe auf: HeGu00415902 in 227 ms
1000000 Zugriffe auf: ViKl00627961 in 329 ms
1000000 Zugriffe auf: NeJo00414018 in 232 ms
1000000 Zugriffe auf: JuKl00179451 in 344 ms
1000000 Zugriffe auf: VüJo00984374 in 301 ms
1000000 Zugriffe auf: HoUw00216841 in 224 ms
1000000 Zugriffe auf: AlJo00275939 in 249 ms
1000000 Zugriffe auf: AlHa00486261 in 251 ms
1000000 Zugriffe auf: WeKa00572480 in 249 ms
1000000 Zugriffe auf: HeAl00400375 in 229 ms
1000000 Zugriffe auf: SoKl00216384 in 298 ms
```

Sie erkennen, dass die Zugriffszeit auf die Elemente in dieser Liste erheblich streut. Die Zugriffszeit auf einige Elemente liegt teilweise doppelt so hoch wie im Durchschnitt.

Rufen Sie nun, ohne das Programm zu beenden, den Task-Manager Ihres Betriebssystems auf. Dazu klicken Sie mit der rechten Maustaste auf die Task-Leiste (auf einen freien Bereich neben dem Startmenü) und wählen den entsprechenden Menüeintrag aus.

Wenn der Task-Manager-Dialog dargestellt ist, wählen Sie die Registerkarte *Prozesse*. Suchen Sie den Eintrag *HashtableDemo*, und merken Sie sich zunächst den Wert, der dort als Speicherauslastung angegeben wurde (etwa wie in Abbildung 19.3 zu sehen).



**Abbildung 19.3:** Bei einem großen Wert für den Parameter Load-Faktor einer Hashtable hält sich der Speicherbedarf in Grenzen, dafür gibt es mehr Zuordnungskollisionen, die Zeit kosten ...

Beenden Sie das Programm anschließend.

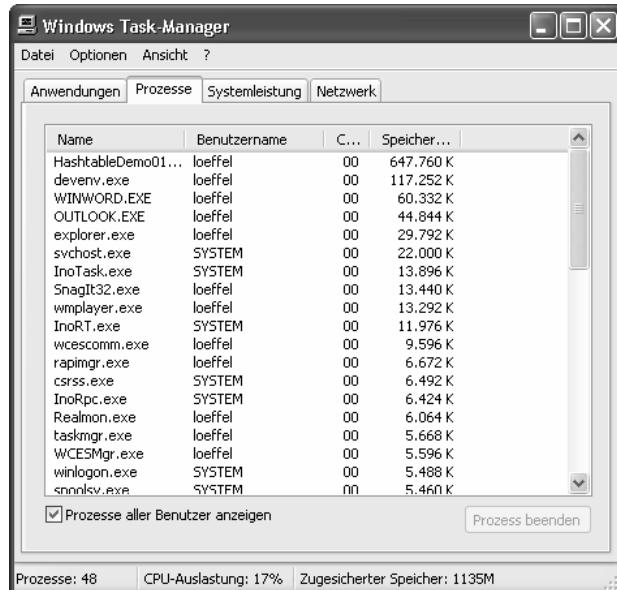
Jetzt verändern Sie den Load-Faktor der Hashtable auf den Wert 0.1 und wiederholen die Prozedur:

```
Dim locAdressen As New Hashtable(100000, 0.1)
```

Wenn Sie nicht gerade über mindestens 1.5 GByte Speicher in Ihrem Rechner verfügen, werden Sie den ersten deutlichen Unterschied bereits beim Aufbauen der Hashtable bemerken, das sehr viel mehr Zeit in Anspruch nimmt. Schuld daran ist in diesem Fall auch das Betriebssystem, das Teile des Hauptspeichers zunächst auf die Platte auslagern muss.

Beobachten Sie anschließend, wie sich der kleinere Load-Faktor auf die Zugriffsgeschwindigkeit der Elemente ausgewirkt hat:

```
1000000 Zugriffe auf: MüGa00374433 in 243 ms
1000000 Zugriffe auf: HeBe00168505 in 227 ms
1000000 Zugriffe auf: WeLo00343022 in 225 ms
1000000 Zugriffe auf: ScKa00611039 in 232 ms
1000000 Zugriffe auf: HoDa00523323 in 225 ms
1000000 Zugriffe auf: LöUw00353805 in 231 ms
1000000 Zugriffe auf: WeKa00855227 in 223 ms
1000000 Zugriffe auf: LöUt00696865 in 230 ms
1000000 Zugriffe auf: WeBr00146001 in 226 ms
1000000 Zugriffe auf: JuJü00334704 in 242 ms
1000000 Zugriffe auf: JuKl00583541 in 226 ms
1000000 Zugriffe auf: BrUw00869665 in 229 ms
1000000 Zugriffe auf: ViBr00974013 in 226 ms
1000000 Zugriffe auf: ScMi00635272 in 224 ms
1000000 Zugriffe auf: WeMi00729698 in 230 ms
```



**Abbildung 19.4:** ... auf der anderen Seite bedeutet ein kleiner Load-Faktor zwar konstantere Zugriffszeiten aber auf Kosten des Speichers

Sie sehen, dass im Gegensatz zum vorherigen Beispiel die Zugriffszeiten nahezu konstant sind. Natürlich kann es bei einer sehr unglücklichen Verteilung immer noch vorkommen, dass es den einen oder anderen Ausreißer gibt. Aber die Anzahl der Ausreißer ist deutlich gesunken.

Betrachten Sie anschließend den Speicherbedarf im Task-Manager, dann sehen Sie sofort, auf wessen Kosten die konstantere Zugriffsgeschwindigkeit erkauft wurde (Abbildung 19.4).

Der Speicherbedarf der Beispielapplikation hat sich nahezu verdreifacht!

Es gibt allerdings noch weiteres Beachtenswertes, das Sie im Hinterkopf behalten sollten, wenn Sie mit Hashtable-Objekten programmieren. Diese Dinge zu beachten wird Ihnen jetzt, da Sie wissen, wie Hashtables prinzipiell funktionieren, abermals leichter fallen.

## Verwenden eigener Klassen als Schlüssel (Key)

In allen bisherigen Beispielen zur Hashtable wurde eine Zeichenkette als Schlüssel eingesetzt. Der Index auf die Tabelle hat sich dabei aus dem Hashcode der als Schlüssel verwendeten Zeichenkette ergeben.

In unserem konkreten Beispiel ist das Ermitteln des Hashcodes des Strings eigentlich ein zu komplizierter und damit zu lange dauernder Algorithmus. Da wir eine eindeutige »Kundennummer« im Matchcode versteckt haben, können wir rein theoretisch auch diese Nummer als Hashcode verwenden, und der Zugriff auf die Elemente sollte damit – unabhängig vom Load-Faktor – gleich bleibend sein, denn die Kundennummer ist immer eindeutig (im Gegensatz zum Hashcode des Matchcodes, bei dem leicht Dubletten und damit Kollisionen beim Einsortieren in die Hashtable auftreten können).

Wenn Sie Objekte eigener Klassen als Schlüssel verwenden wollen, müssen Sie die beiden folgenden Punkte beherzigen:

- Die Klasse muss ihre Equals-Funktionen überschreiben, damit die Hashtable-Klasse die Gleichheit zweier Schlüssel prüfen kann.
- Die Klasse muss die GetHashCode-Funktion überschreiben, damit die Hashtable-Klasse überhaupt einen Hashcode ermitteln kann.

---

**WICHTIG:** Wenn Sie die Equals-Funktion überschreiben, achten Sie bitte darauf, die richtige Überladungsversion dieser Funktion zu überschreiben. Da die Basisfunktion (die Equals-Funktion von Object) sowohl als nicht statische als auch als statische Funktion implementiert ist, müssen Sie das Overloads-Schlüsselwort anwenden. Da Sie die Funktion überschreiben wollen, müssen Sie Overrides ebenfalls verwenden. Overrides alleine lässt der Compiler nicht zu, da die Funktion schon in die Basisklasse überladen ist. Allerdings – und das ist das Gefährliche – lässt er ein einzelnes Overloads zu, und das führt zu einer Überschattung der Basisfunktion, *ohne* sie zu überschreiben. Das Ergebnis: Der Compiler zeigt Ihnen keine Fehlermeldung (nicht einmal eine Warnung, obwohl er das sollte!), doch Ihre Funktion wird nicht polymorph behandelt und damit nie aufgerufen.

---

Mit diesem Wissen können wir nun unsere eigene Key-Klasse in Angriff nehmen. Bei unserem stark vereinfachten Beispiel bietet sich ein Teil der Matchcode-Zeichenkette an, direkt als *Hashcode* zu fungieren. Da die laufende Nummer einer Adresse Teil des Matchcodes ist, können wir genau diese als *Hashcode* verwenden. Der Code für eine eigene Key-Klasse könnte sich folgendermaßen gestalten:

---

**BEGLEITDATEIEN:** Sie finden das veränderte Beispielprogramm unter *.\\VB 2005 - Entwicklerbuch\\E – Datentypen\\Kap19\\HashtableDemo02\\* im Verzeichnis der CD zum Buch. Veränderungen in Beispielcode sind im Folgenden fett markiert.

---

```
Public Class AdressenKey

    Private myMatchcode As String
    Private myKeyValue As Integer

    Sub New(ByVal Matchcode As String)
        myKeyValue = Integer.Parse(Matchcode.Substring(4))
        myMatchcode = Matchcode
    End Sub

    'Wird benötigt, um bei Kollisionen den richtigen
    'Schlüssel zu finden.
    Public Overloads Function Equals(ByVal obj As Object) As Boolean
        'If Not (TypeOf obj Is AdressenKey) Then
        '    Dim up As New InvalidCastException("AdressenKey kann nur mit Objekten gleichen Typs verglichen werden")
        '    Throw up
        'End If
        Return myKeyValue.Equals(DirectCast(obj, AdressenKey).KeyValue)
    End Function

    'Wird benötigt, um den Index zu "berechnen".
    Public Overrides Function GetHashCode() As Integer
        Return myKeyValue
    End Function

```

```

Public Overrides Function ToString() As String
    Return myKeyValue.ToString
End Function

Public Property KeyValue() As Integer
    Get
        Return myKeyValue
    End Get
    Set(ByVal Value As Integer)
        myKeyValue = Value
    End Set
End Property

End Class

```

Am ursprünglichen Testprogramm selbst sind nur einige kleinere Änderungen notwendig, um die neue Key-Klasse zu implementieren. Die Adresse-Klasse muss dabei überhaupt keine Änderung erfahren. Lediglich am Hauptprogramm müssen einige Zeilen geändert werden, um die Änderung wirksam werden zu lassen.

```

Module HashtableDemo

Sub Main()

    Dim locAnzahlAdressen As Integer = 1000000
    Dim locZugriffsElemente As Integer = 50
    Dim locMessungen As Integer = 5
    Dim locZugriffe As Integer = 1000000
    Dim locVorlageAdressen As ArrayList
    Dim locAdressen As New Hashtable(100000, 0.1)
    Dim locTestKeys(locZugriffsElemente) As AdressenKey
    Dim locZeitmesser As New HighSpeedTimeGauge
    Dim locRandom As New Random(Now.Millisecond)

    .
    . ' Aus Platzgründen weggelassen.

    .
    'Aufbauen der Hashtable
    Console.WriteLine("Aufbauen der Hashtable mit zufälligen Adresseneinträgen...", locAnzahlAdressen)
    locZeitmesser.Start()
    For Each adresse As Adresse In locVorlageAdressen
        'Änderung: Nicht den String, sondern ein Key-Objekt verwenden
        locAdressen.Add(New AdressenKey(adresse.Matchcode), adresse)
    Next
    locZeitmesser.Stop()
    Console.WriteLine("fertig nach {0} ms", locZeitmesser.DurationInMilliseconds)
    locZeitmesser.Reset()

    '51 zufällige Adressen rauspicken.
    'Änderung: Die Keys werden abgespeichert, nicht der Matchcode.
    For i As Integer = 0 To locZugriffsElemente
        locTestKeys(i) = New AdressenKey(
            DirectCast(locVorlageAdressen(locRandom.Next(locAnzahlAdressen)), Adresse).Matchcode)
    Next

```

```

'Änderung: Kein Object mehr, sondern direkt ein AdressenKey.
Dim locTemp As AdressenKey
Dim locTemp2, locTemp3 As Object

'Zugreifen und messen, wie lange das dauert,
'Das ganze fünfmal, um die Messung zu bestätigen.
For z As Integer = 1 To locMessungen
    Console.WriteLine()
    Console.WriteLine("{0}. Messung:", z)
    For i As Integer = 0 To locZugriffsElemente
        Console.Write("{0} Zugriffe auf: {1} in ", locZugriffe, locTestKeys(i))
        locTemp = locTestKeys(i)
        locZeitmesser.Start()
        For j As Integer = 1 To locZugriffe
            locTemp2 = locAdressen(locTemp)
        Next j
        locZeitmesser.Stop()
        locTemp3 = locTemp2.GetType
        Console.WriteLine("{0} ms", locZeitmesser.DurationInMilliseconds)
    Next
    'Zugriff auf ArrayList für Vergleich
    .
    . ' Aus Platzgründen ebenfalls weggelassen.
    .
    Next
    Console.ReadLine()
End Sub

```

Die entscheidende Zeile im Beispielcode hat übrigens gar keine Änderung erfahren müssen. `locTemp` dient nach wie vor als Objektvariable für den Schlüssel, nur ist sie nicht mehr vom Typ `String` definiert, sondern als `AdressenKey`. `locTemp3` (und ehemals `locTemp`) dienen übrigens nur dazu, dass der Compiler die innere Schleife nicht wegoptimiert<sup>6</sup> und damit Messergebnisse verfälscht.

Wenn Sie dieses Programm starten, werden sie zwei Dinge feststellen. Der Zugriff auf die Daten ist spürbar schneller geworden, und: Der Zugriff auf die Daten erfolgt unabhängig vom Load-Faktor immer gleich schnell. Da der Schlüssel auf die Daten nun eindeutig ist, braucht sich die Hashtable nicht mehr um Kollisionen zu kümmern – es gibt nämlich keine mehr. Folglich bringt die Reservierung zusätzlicher Elemente auch keinen nennenswerten Vorteil mehr. Ganz im Gegenteil: Sie würden Speicher verschwenden, der gar nicht benötigt würde.

In diesem Beispiel sind die zu verwaltenden Datensätze nicht sonderlich groß gewesen. Wenn Sie überlegen, wie viele Zugriffe auf die Objekte der Hashtable tatsächlich notwendig gewesen sind, um überhaupt in einen messbaren Bereich zu gelangen, dann wird deutlich, dass sich der betriebene Aufwand für dieses Beispiel eigentlich nicht gelohnt hat. Dennoch: Denken Sie immer daran, dass Maschinen, auf denen Ihre Software später läuft, in der Regel nicht so leistungsfähig sind wie die Maschinen, auf denen sie entwickelt wird.

---

<sup>6</sup> Dieser Handgriff dient nur als Vorsichtsmaßnahme. Ich gebe zu, nicht wirklich überprüft zu haben, ob die Zeile wegoptimiert werden würde. Bei der Intelligenz moderner Compiler (oder JITter) ist das aber stark anzunehmen.

## Schlüssel müssen unveränderlich sein!

Wenn Sie sich dazu entschlossen haben, eigene Klassen für die Verwaltung von Schlüsseln in einer Hashtable zu entwickeln, sollten Sie zusätzlich zum Gesagten Folgendes unbedingt beherzigen: Schlüssel müssen unveränderlich sein. Achten Sie darauf, dass Sie den Inhalt eines Key-Objektes nicht von außen verändern, solange er einer Hashtable zugeordnet ist. In diesem Fall würden Sie riskieren, dass die GetHashCode-Funktion unter Umständen einen falschen Hashcode für ein Key-Objekt zurückliefert. Der Nachschlagealgorismus der Hashtable hätte dann keine Chance mehr, das Objekt in der Datentabelle wieder zu finden.

## Enumerieren von Datenelementen in einer Hashtable

Die Enumeration einer *Hashtable* (das Iterieren durch sie mit `For/Each`) ist prinzipiell möglich. Allerdings müssen sie zwei Dinge dabei beachten:

- Datenelemente werden in einer Hashtable nicht in sequentieller Reihenfolge gespeichert. Da der Hashcode eines zu speichernden Objektes ausschlaggebend für die Position des Objektes innerhalb der Datentabelle ist, kann die wirkliche Position eines Objektes nur bei ganz einfachen Hashcode-Algorithmen vorausgesagt werden. Wenn Sie – und das wird wahrscheinlich am häufigsten der Fall sein – ein String-Objekt als Schlüssel verwenden, wirken die zu speichernden Objekte schon mehr oder weniger zufällig in der Tabelle verteilt.
- Objekte werden innerhalb einer Hashtable in so genannten Bucket-Strukturen gespeichert. Ein Schlüssel gehört untrennbar zu seinem eigentlichen Objekt, und beide werden in einem Bucket-Element in der Tabelle abgelegt. Eine Iteration durch die Datentabelle kann aus diesem Grund nur mit einem speziellen Objekt erfolgen – nämlich vom Typ `DictionaryEntry` (etwa: *Wörterbucheintrag*).

Eine Iteration durch eine Hashtable könnte für unser Beispiel folgendermaßen aussehen:

```
'Iterieren durch die Hashtable
For Each locDE As DictionaryEntry In locAdressen
    'in unserem Beispiel für den Key
    Dim locAdressenKey As AdressenKey = DirectCast(locDE.Key, AdressenKey)
    'in unserem Beispiel für das Objekt
    Dim locAdresse As Adresse = DirectCast(locDE.Value, Adresse)
Next
```

## Typsichere Hashtable

Bei der Entwicklung einer typsicheren Hashtable stehen wir vor ähnlichen Problemen, wie bei einer typsicheren ArrayList. Da ich aus Platzgründen die komplette Theorie nicht wiederholen möchte, gehe ich einfach davon aus, dass Sie den ►Abschnitt »Typsichere Auflistungen auf Basis von CollectionBase« ab Seite 568 durchgearbeitet haben.

Das folgende Beispiel demonstriert den Einsatz einer typsicheren Hashtable an der Erweiterung des vorherigen Beispiels.

---

**BEGLEITDATEIEN:** Im Ordner `\VB 2005 - Entwicklerbuch\E - Datentypen\Kap19\Hashtable03\` finden Sie das veränderte Beispielprogramm im Verzeichnis der CD zum Buch. Veränderungen im Beispielcode sind fett markiert.

---

Zunächst finden Sie in der Klassendatei *Daten.vb* des Projektes zusätzlich zu der vorhandenen Klasse die Klasse Adressen. Sie ist aus der abstrakten Klasse *DictionaryBase* abgeleitet, die die Grundfunktionalität der Hashtable beinhaltet:

```
'Typsichere Adressen-Auflistung auf Wörterbuchbasies
Public Class Adressen
    Inherits DictionaryBase

    'Default-Eigenschaft erlaubt das Auslesen der typsicheren Hashtable.
    Default Public Property Item(ByVal key As AdressenKey) As Adresse
        Get
            Return DirectCast(Dictionary(key), Adresse)
        End Get
        Set(ByVal Value As Adresse)
            Dictionary(key) = Value
        End Set
    End Property

    'Liefert eine ICollection aller Keys zurück.
    Public ReadOnly Property Keys() As ICollection
        Get
            Return Dictionary.Keys
        End Get
    End Property

    'Liefert eine ICollection aller Werte zurück.
    Public ReadOnly Property Values() As ICollection
        Get
            Return Dictionary.Values
        End Get
    End Property

    'Erlaubt das Hinzufügen eines Eintrags typsicher.
    Public Sub Add(ByVal key As AdressenKey, ByVal value As Adresse)
        Dictionary.Add(key, value)
    End Sub

    'Überprüft, ob ein bestimmter Key in der Liste enthalten ist.
    Public Function Contains(ByVal key As AdressenKey) As Boolean
        Return Dictionary.Contains(key)
    End Function

    'Entfernt ein Element aus der Liste mithilfe seines Keys.
    Public Sub Remove(ByVal key As AdressenKey)
        Dictionary.Remove(key)
    End Sub
End Class
```

Anstelle der Hashtable setzen wir anschließend im Hauptprogramm diese Klasse ein. Die Änderungen dafür sind minimal:

```
Dim locAnzahlAdressen As Integer = 1000000
Dim locZugriffsElemente As Integer = 50
Dim locMessungen As Integer = 5
Dim locZugriffe As Integer = 1000000
Dim locVorlageAdressen As ArrayList
```

```

Dim LocAdressen As New Adressen
Dim locTestKeys(locZugriffsElemente) As AdressenKey
Dim locZeitmesser As New HighSpeedTimeGauge
Dim locRandom As New Random(Now.Millisecond)

```

Die erste Änderung betrifft die Deklaration der Hashtable am Anfang des Programms (siehe fett markierte Zeile). Sie wird hier nicht mehr als Hashtable-Objekt, sondern als Objekt vom Typ Adressen deklariert – damit wird die verwendete Hashtable typsicher.

```

Dim locTemp As AdressenKey
'Typsichere Hashtable: Indexer liefert direkt Adresse-Objekt zurück
Dim locTemp2 As Adresse
Dim locTemp3 As Object

'Zugreifen und Messen, wie lange das dauert,
'Das ganze fünfmal, um die Messung zu bestätigen.
For z As Integer = 1 To locMessungen
    Console.WriteLine()
    Console.WriteLine("{0}. Messung:", z)
    For i As Integer = 0 To locZugriffsElemente
        Console.Write("{0} Zugriffe auf: {1} in ", locZugriffe, locTestKeys(i))
        locTemp = locTestKeys(i)
        locZeitmesser.Start()
        For j As Integer = 1 To locZugriffe
            locTemp2 = locAdressen(locTemp)
        Next j
        locZeitmesser.Stop()
        locTemp3 = locTemp2.GetType()
        Console.WriteLine("{0} ms", locZeitmesser.DurationInMilliseconds)
    Next
    .
    .
    .
    Next
    Console.ReadLine()

```

Die zweite Änderung betrifft den Rückgabewerttyp, der zurückgeliefert wird, wenn die typsichere Hashtable per Index ausgelesen wird. locTemp2 ist nunmehr direkt vom Typ Adresse definiert, und die Zuweisung aus einem Element der Hashtable läuft – dank Typsicherheit – ohne Casting ab.

---

**HINWEIS:** Wie Sie selbst anhand der ermittelten Zeiten erkennen können, geht der Einbau der Typsicherheit auf Kosten der Geschwindigkeit. Anders ist das, wenn Sie generische Wörterbuchaufstellungsklassen wie beispielsweise KeyedCollection verwenden, über die Sie in ► Kapitel 20 genauere Ausführungen finden.

---

## Queue – Warteschlangen im FIFO-Prinzip

»First in first out« (als erstes rein, als erstes raus) – nach diesem Muster arbeitet die Queue-Klasse der BCL. Angewendet haben Sie dieses Prinzip selbst schon sicherlich einige Male in der Praxis – und zwar immer dann, wenn Sie unter Windows mehrere Dokumente hintereinander gedruckt haben. Das Drucken unter Windows funktioniert gemäß dem Warteschlangenprinzip. Das Dokument, das als Erstes in die Warteschlange eingereiht (*enqueue* – einreihen) wurde, wird als Erstes verarbeitet

und anschließend wieder aus ihr entfernt (*dequeue* – ausreihen). Aus diesem Grund verwenden Sie die Methoden *Enqueue*, um Elemente der Queue hinzuzufügen und *Dequeue*, um sie zurückzubekommen und gleichzeitig aus der Warteschlange zu entfernen.

---

**BEGLEITDATEIEN:** Falls Sie mit der Queue-Klasse experimentieren möchten, verwenden Sie dazu am besten nochmals das *CollectionsDemo*-Projekt, das Sie unter *.|VB 2005 - Entwicklerbuch|E - Datentypen|Kap19\CollectionsDemo\* finden. Verändern Sie das Programm so, dass es die Sub *QueueDemo* aufruft, um das folgende Beispiel nachzuvollziehen:

---

```
Sub QueueDemo()

    Dim locQueue As New Queue
    Dim locString As String

    locQueue.Enqueue("Erstes Element")
    locQueue.Enqueue("Zweites Element")
    locQueue.Enqueue("Drittes Element")
    locQueue.Enqueue("Viertes Element")

    'Nachschauen, was am Anfang steht, ohne es zu entfernen.
    Console.WriteLine("Element am Queue-Anfang:" + locQueue.Peek().ToString)
    Console.WriteLine()

    'Iterieren funktioniert auch.
    For Each locString In locQueue
        Console.WriteLine(locString)
    Next
    Console.WriteLine()

    'Alle Elemente aus Queue entfernen und Ergebnis im Konsolenfenster anzeigen.
    Do
        locString = CStr(locQueue.Dequeue)
        Console.WriteLine(locString)
    Loop Until locQueue.Count = 0
    Console.ReadLine()
End Sub
```

Wenn Sie dieses Programm ablaufen lassen, produziert es folgende Ausgabe im Konsolenfenster:

```
Element am Queue-Anfang:Erstes Element
```

```
Erstes Element
Zweites Element
Drittes Element
Viertes Element
```

```
Erstes Element
Zweites Element
Drittes Element
Viertes Element
```

# Stack – Stapelverarbeitung im LIFO-Prinzip

Die Stack-Klasse arbeitet nach dem Prinzip »Last in first out« (»als letztes rein, als erstes raus«), arbeitet also genau umgekehrt zum FIFO-Prinzip der Queue-Klasse. Mit der Push-Methode schieben Sie ein Element auf den Stapel, mit Pull ziehen Sie es wieder herunter und erhalten es damit zurück. Das Element, das Sie zuletzt auf den Stapel geschoben haben, wird also mit Pull auch als Erstes wieder entfernt.

---

**BEGLEITDATEIEN:** Falls Sie mit der Stack-Klasse experimentieren möchten, verwenden Sie das *CollectionsDemo*-Projekt, das Sie unter .\VB 2005 - Entwicklerbuch\E - Datentypen\Kap19\CollectionsDemo\ finden. Verändern Sie das Programm, sodass es die Sub StackDemo aufruft, um das folgende Beispiel nachzuvollziehen:

---

```
Sub StackDemo()
    Dim locStack As New Stack
    Dim locString As String

    locStack.Push("Erstes Element")
    locStack.Push("Zweites Element")
    locStack.Push("Drittes Element")
    locStack.Push("Viertes Element")

    'Nachschauen, was oben auf dem Stapel liegt, ohne das Element zu entfernen.
    Console.WriteLine("Element zu oberst auf dem Stapel: " + locStack.Peek.ToString)
    Console.WriteLine()

    'Iterieren funktioniert auch.
    For Each locString In locStack
        Console.WriteLine(locString)
    Next
    Console.WriteLine()

    'Alle Elemente vom Stack ziehen und Ergebnis im Konsolenfenster anzeigen.
    Do
        locString = CStr(locStack.Pop)
        Console.WriteLine(locString)
    Loop Until locStack.Count = 0
    Console.ReadLine()
End Sub
```

Wenn Sie dieses Programm ablaufen lassen, produziert es folgende Ausgabe im Konsolenfenster:

Element zu oberst auf dem Stapel: Viertes Element

Viertes Element  
Drittess Element  
Zweites Element  
Erstes Element

Viertes Element  
Drittess Element  
Zweites Element  
Erstes Element

# SortedList – Elemente ständig sortiert halten

Wenn Sie Elemente schon direkt nach dem Einfügen in der richtigen Reihenfolge in einer Auflistung halten wollen, dann ist die `SortedList`-Klasse das richtige Werkzeug für diesen Zweck. Allerdings sollten Sie beachten: Von allen Auflistungsklassen ist die `SortedList`-Klasse diejenige, die die meisten Ressourcen verschlingt. Für zeitkritische Applikationen sollten Sie überlegen, ob Sie Ihre Daten auch anders organisieren und stattdessen lieber auf eine unsortierte Hashtable oder gar auf `ArrayList` zurückgreifen können.

Der Vorteil von `SortedList` ist, dass sie quasi aus einer Mischung von `ArrayList`- und `Hashtable`-Funktionen besteht (obwohl sie algorithmisch gesehen, überhaupt nichts mit `Hashtable` zu tun hat). Sie können auf der einen Seite über einen Schlüssel, auf der anderen Seite aber auch über einen Index auf die Elemente von `SortedList` zugreifen.

Das folgende erste Beispiel zeigt den generellen Umgang mit `SortedList`.

---

**BEGLEITDATEIEN:** Sie finden dieses Projekt unter `.\VB 2005 - Entwicklerbuch\E - Datentypen\Kap19\SortedListDemo`. Es besteht aus zwei Codedateien. In der Datei `Daten.vb` finden Sie die schon bekannte Adresse-Klasse (bekannt, falls Sie die vorherigen Abschnitte ebenfalls durchgearbeitet haben) – allerdings in leicht veränderter Form. Der Matchcode der Zufallsadressen beginnt in dieser Version mit einer laufenden Nummer und endet mit der Buchstabenkombination des Nach- und Vornamens. Damit wird vermieden, dass eine Sortierung des Matchcodes grob auch die Adressen nach Namen und Vornamen sortiert und etwaige Nachweise eines bestimmten Programmverhaltens nicht geführt werden können.

---

```
Module SortedListDemo
    Sub Main()
        Dim locZufallsAdressen As ArrayList = Adresse.ZufallsAdressen(6)
        Dim locAdressen As New SortedList

        Console.WriteLine("Ursprungsanordnung:")
        For Each locAdresse As Adresse In locZufallsAdressen
            Console.WriteLine(locAdresse)
            locAdressen.Add(locAdresse.Matchcode, locAdresse)
        Next

        ' Zugriff per Index:
        Console.WriteLine()
        Console.WriteLine("Zugriff per Index:")
        For i As Integer = 0 To locAdressen.Count - 1
            Console.WriteLine(locAdressen.GetByIndex(i).ToString())
        Next

        Console.WriteLine()
        Console.WriteLine("Zugriff per Index:")
        ' Zugriff per Enumerator
        For Each locDE As DictionaryEntry In locAdressen
            Console.WriteLine(locDE.Value.ToString())
        Next
        Console.ReadLine()
    End Sub
End Module
```

Wenn Sie dieses Programm starten, generiert es in etwa die folgenden Ausgaben im Konsolenfenster (die Adressen werden zufällig generiert, deswegen kann die Darstellung in Ihrem Konsolenfenster natürlich wieder von der hier gezeigten abweichen).

Ursprungsanordnung:

```
00000005PlKa: Plenge, Katrin, 26201 Liebenburg
00000004PlKa: Plenge, Katrin, 93436 Liebenburg
00000003AlMa: Albrecht, Margarete, 65716 Bad Waldliesborn
00000002HoBa: Hollmann, Barbara, 96807 Liebenburg
00000001LöLo: Löffelmann, Lothar, 21237 Lippetal
00000000AdKa: Ademmer, Katrin, 49440 Unterschleißheim
```

Zugriff per Index:

```
00000000AdKa: Ademmer, Katrin, 49440 Unterschleißheim
00000001LöLo: Löffelmann, Lothar, 21237 Lippetal
00000002HoBa: Hollmann, Barbara, 96807 Liebenburg
00000003AlMa: Albrecht, Margarete, 65716 Bad Waldliesborn
00000004PlKa: Plenge, Katrin, 93436 Liebenburg
00000005PlKa: Plenge, Katrin, 26201 Liebenburg
```

Zugriff per Index:

```
00000000AdKa: Ademmer, Katrin, 49440 Unterschleißheim
00000001LöLo: Löffelmann, Lothar, 21237 Lippetal
00000002HoBa: Hollmann, Barbara, 96807 Liebenburg
00000003AlMa: Albrecht, Margarete, 65716 Bad Waldliesborn
00000004PlKa: Plenge, Katrin, 93436 Liebenburg
00000005PlKa: Plenge, Katrin, 26201 Liebenburg
```

Sie erkennen, dass die Liste in der Tat nach dem Schlüssel umsortiert wurde. Dies gilt sowohl für den Zugriff über den Index als auch über den Enumerator mit For/Each.

### Zugriff auf Elemente der SortedList per Indexer

Wenn Sie per Index auf die Elemente der SortedList zugreifen, verwenden Sie dazu deren GetByIndex-Eigenschaft, so wie in der fett markierten Zeile im Listing zu sehen. Möchten Sie ein Element in der SortedList verändern und dabei nicht einen Schlüssel zur Positionsbestimmung, sondern seinen Index verwenden, verwenden Sie die Methode SetByIndex.

---

**HINWEIS:** Wenn Sie eine bestimmte Synchronisation zwischen Ihrem Schlüssel und dem eigentlichen Objekt einhalten müssen, sollten Sie diese letzte Methode allerdings nicht verwenden. Da Sie in diesem Fall nur das eigentliche Datenobjekt in der Liste, nicht aber dessen Schlüssel verändern, könnte unter Umständen Ihre eigene Datenstruktur bei der Datenverwaltung verletzt werden.

---

## Sortierung einer SortedList nach beliebigen Datenkriterien

Beim Hinzufügen eines Schlüssel-/Wertepaars zu einem SortedList-Objekt ist dessen Schlüssel ausschlaggebend für die Sortierreihenfolge – es ist eigentlich nicht möglich, die Liste nach bestimmten Eigenschaften des hinzugefügten Objektes sortieren zu lassen, auch nicht mit speziellen IComparer-eingebundenen Klassen. Da doppelte Werte innerhalb der von SortedList verwalteten Liste vorkommen können, können Sie auch nicht jede beliebige Eigenschaft des einzusortierenden Objekts verwenden.

tes als Schlüssel verwenden, da die Schlüssel eindeutig sein müssen. Dummerweise ist das genau der Knackpunkt, denn: CompareTo der jeweils verwendeten Key-Klasse wird ja ebenfalls dazu verwendet, damit SortedList herausfinden kann, ob es bereits einen Schlüssel in der Liste gibt.

Allerdings: Sie können den folgenden Trick anwenden, um eine SortedList dennoch nach beliebigen Eigenschaften der zu speichernden Klasse zu sortieren, auch wenn das erheblich auf Kosten der Performance geht!

---

**WARNUNG:** Deswegen die dringende Warnung an dieser Stelle: Mit dem folgenden Kniff tricksen wir den Algorithmus zur Platzierung der Elemente einer SortedList komplett aus – und zwar auf die Kosten seiner Effektivität. Wenden Sie diesen Algorithmus nur an, wenn Sie wenige Elemente sortieren müssen.

---

Schauen Sie sich den folgenden Code der Klasse AdressenKey an, deren instanzierte Objekte als Schlüssel der Daten fungieren sollen:

```
Public Class AdressenKey
    Implements IComparable

    Private myMatchcode As String
    Private myKeyValue As Integer
    Private myDataToSort As String

    Sub New(ByVal Matchcode As String, ByVal DataToSort As String)
        myKeyValue = Integer.Parse(Matchcode.Substring(0, 8))
        myMatchcode = Matchcode
        myDataToSort = DataToSort
    End Sub

    'Wird benötigt, um bei Kollisionen den richtigen
    'Schlüssel zu finden.
    Public Overrides Function Equals(ByVal obj As Object) As Boolean
        Return myKeyValue.Equals(DirectCast(obj, AdressenKey).KeyValue)
    End Function

    'Wird benötigt, um den Index zu "berechnen".
    Public Overrides Function GetHashCode() As Integer
        Return myKeyValue
    End Function

    Public Overrides Function ToString() As String
        Return myKeyValue.ToString()
    End Function

    Public Property KeyValue() As Integer
        Get
            Return myKeyValue
        End Get
        Set(ByVal Value As Integer)
            myKeyValue = Value
        End Set
    End Property
```

```

Public Property DataToSort() As String
    Get
        Return myDataToSort
    End Get
    Set(ByVal Value As String)
        myDataToSort = Value
    End Set
End Property

Public Function CompareTo(ByVal obj As Object) As Integer Implements System.IComparable.CompareTo
    If myMatchcode = DirectCast(obj, AdressenKey).myMatchcode Then
        Return 0
    End If
    If DataToSort.CompareTo(DirectCast(obj, AdressenKey).DataToSort) = 0 Then
        Return -1
    Else
        Return DataToSort.CompareTo(DirectCast(obj, AdressenKey).DataToSort)
    End If
End Function
End Class

```

Sie stellen fest, dass der Konstruktor der Klasse nicht nur den eigentlichen als Schlüssel fungierenden Wert, sondern ein weiteres Datenfeld aufnimmt. Dieses weitere Datenfeld ist – um das Beispiel einfach zu halten – ein String, könnte rein theoretisch aber auch jedes andere Objekt sein.

Der Trick funktioniert nun folgendermaßen: Die CompareTo-Methode muss primär so funktionieren, dass doppelte Schlüssel ermittelt und durch eine Ausnahme, die SortedList im Bedarfsfall erzeugt, ausgeschlossen werden können. CompareTo liefert also dann 0 zurück, wenn es sich bei den zu vergleichenden AdressenKey-Instanzen um gleiche Schlüssel handelt. Wenn das nicht der Fall ist, werden allerdings nicht die wirklichen Schlüssel der Größe nach verglichen, sondern die Sortierfelder. Natürlich können diese ebenfalls gleich sein, doch sind dieses Mal Dubletten gestattet. SortedList darf davon aber nichts mitbekommen, weil SortedList sonst wiederum von der Gleichheit der Schlüssel ausgehen würde – eine Ausnahme wäre die Folge. Also liefern wir, selbst wenn die beiden Sortierfelder gleich sind, den Wert für *kleiner* als Ergebnis zurück. Da es sowieso egal ist, wenn beispielsweise ausschließlich nach Nachnamen sortiert wird, welcher der doppelten »Thiemänner« an erster Stelle kommt, hat das Zurückliefern des »falschen« Vergleichsergebnisses in der Praxis keine Auswirkungen.

Diesen Trick sehen Sie im oben gezeigten Listing in den fettgedruckten Zeilen umgesetzt. Der Wert 0 für *gleich* wird nur bei gleichen Matchcodes (den Schlüsseln) zurückgeliefert. Gleiche Matchcodes sind also auch in unserer Auflistung nicht gestattet, damit eine eindeutige Auflösung von Matchcodes zu eigentlichen Datensätzen gewährleistet bleibt. Wenn die Matchcodes ungleich gewesen sind, liefert der nächste Teil der CompareTo-Methode aber das Vergleichsergebnis der Werte zurück, nach denen sortiert werden soll – es sei denn, sie wären gleich. Sind sie es, wird gemogelterweise der Wert *-1* für *kleiner* zurückgeliefert, und dass diese kleine Mogelei in der Praxis keine Auswirkungen hat, zeigt das Testprogramm in Form von Sub SortedByFieldDemo, das Sie laufen lassen können, wenn Sie die Auskommentierung der beiden Zeilen am Anfang des Moduls zurücknehmen:

```

Sub SortedByFieldDemo()
    Dim locZufallsAdressen As ArrayList = Adresse.ZufallsAdressen(20)
    Dim locAdressen As New SortedList
    Dim locAdressenKey As AdressenKey

```

```

Console.WriteLine("Ursprungsanordnung:")
For Each locAdresse As Adresse In locZufallsAdressen
    locAdressenKey = New AdressenKey(locAdresse.Matchcode, locAdresse.Name)
    locAdressen.Add(locAdressenKey, locAdresse)
Next

Console.WriteLine()
Console.WriteLine("Zugriff per Index:")
'Zugriff per Enumerator
For Each locDE As DictionaryEntry In locAdressen
    Console.WriteLine(locDE.Key.ToString + ":: " + locDE.Value.ToString)
    'Console.WriteLine(locDE.Value.ToString)
Next
Console.ReadLine()
End Sub

```

Wenn Sie dieses Programm laufen lassen, sehen Sie in etwa folgende Zeilen im Konsolenfenster:

Ursprungsanordnung:

```

00000014WeUt: Weichelt, Uta, 18364 Bielefeld
00000013ThKa: Thiemann, Katrin, 80995 Berlin
00000012JuDa: Jungemann, Daja, 31318 Bad Waldliesborn
00000011TiMa: Tinoco, Margarete, 67423 Rheda
00000010EnCh: Englisch, Christian, 28395 Lippstadt
00000009MeJo: Meier, José, 48230 Soest
00000008SoUt: Sonntag, Uta, 52796 Straubing
00000007MeAn: Meier, Anne, 92606 Rheda
00000006WeJü: Westermann, Jürgen, 76188 Lippetal
00000005TiDa: Tinoco, Daja, 46492 Rheda
00000004SoBr: Sonntag, Britta, 89217 Dortmund
00000003LöUt: Löffelmann, Uta, 93934 Bad Waldliesborn
00000002RoAn: Rode, Anne, 05647 München
00000001AlMe: Albrecht, Melanie, 45944 Wiesbaden
00000000HöLo: Hörstmann, Lothar, 29607 Straubing

```

Zugriff per Index:

```

00000001AlMe: Albrecht, Melanie, 45944 Wiesbaden
00000010EnCh: Englisch, Christian, 28395 Lippstadt
00000000HöLo: Hörstmann, Lothar, 29607 Straubing
00000012JuDa: Jungemann, Daja, 31318 Bad Waldliesborn
00000003LöUt: Löffelmann, Uta, 93934 Bad Waldliesborn
00000009MeJo: Meier, José, 48230 Soest
00000007MeAn: Meier, Anne, 92606 Rheda
00000002RoAn: Rode, Anne, 05647 München
00000008SoUt: Sonntag, Uta, 52796 Straubing
00000004SoBr: Sonntag, Britta, 89217 Dortmund
00000013ThKa: Thiemann, Katrin, 80995 Berlin
00000011TiMa: Tinoco, Margarete, 67423 Rheda
00000005TiDa: Tinoco, Daja, 46492 Rheda
00000014WeUt: Weichelt, Uta, 18364 Bielefeld
00000006WeJü: Westermann, Jürgen, 76188 Lippetal

```

- 
- 595 **Wertetypen, die Nothing speichern können – Nullable(Of )**
  - 599 **Generische Auflistungen (Generic Collections)**
  - 607 **Auflistungen und Aktionen (Actions), Aussageprüfer (Predicates) und Vergleiche (Comparisons)**
- 

Wie Sie generische Typen entwickeln, haben Sie in ► Kapitel 14 bereits kennen gelernt. ► Kapitel 16 hat – wenn auch in einem anderen Zusammenhang – den Einsatz von generischen Typen das erste Mal demonstriert. Dort konnten Sie auch schon sehen, dass gerade was Auflistungen anbelangt, es viele generische Datentypen bereits zur direkten Verwendung im .NET-Framework gibt. Dieses Kapitel gibt Ihnen einen Überblick über die wichtigsten generischen Datentypen und Auflistungen.

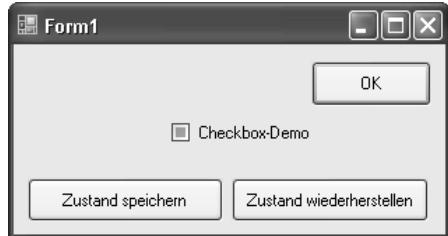
## **Wertetypen, die Nothing speichern können – Nullable(Of )**

Kaum eine Neuerung hat für so viel Wirbel, Spekulationen und auch Stress in den eigenen (Entwickler-)Reihen gesorgt, wie der Datentyp `Nullable`, von dem in diesem Abschnitt die Rede sein wird.

`Nullable` ist ein generischer Datentyp mit einer Einschränkung auf Wertetypen. Er ermöglicht, dass ein beliebiger Wertetyp neben seiner eigentlichen Werteart einen weiteren Zustand »speichern« kann – nämlich `Nothing`.

Ist das wichtig? Oh ja! Beispielsweise in der Datenbankprogrammierung. Wenn Sie bereits Erfahrungen in der Datenbankprogrammierung haben, wissen Sie auch sicherlich, dass Ihre Datenbanktabellen über Datenfelder verfügen können, die den »Wert« `Null` »speichern« können – als Zeichen dafür, dass eben nichts in diesem Feld gespeichert wurde.

Ein anderes Beispiel sind CheckBox-Steuerelemente in Windows Forms-Anwendungen: Sie verfügen über einen Zwischenzustand, der den Zustand »nicht definiert« anzeigen soll. Eine einfache boolesche Variable könnte alle möglichen Zustände nicht aufnehmen – `True` und `False` sind dafür einfach zu wenig. Anders ist es, wenn Sie eine Variable vom Typ `Nullable(Of Boolean)` definieren würden. In diesem Fall könnte man eine Fallunterscheidung zwischen allen möglichen Zuständen folgendermaßen realisieren:



**Abbildung 20.1:** Ein Nullable(Of Boolean) eignet sich beispielsweise dazu, auch Zwischenzustände eines CheckBox-Steuerelements – wie hier im Bild zu sehen – zu speichern

---

**BEGLEITDATEIEN:** Sie finden das folgende Beispielprojekt unter .\VB 2005 - Entwicklerbuch\E - Datentypen\Kap20\NullableUndCheckbox.

---

Dieses Beispiel demonstriert, wie alle Zustände eines CheckBox-Steuerelements, dessen ThreeState-Eigenschaft zur Anzeige aller *drei* Zustände auf True gesetzt wurde, in einer Member-Variablen vom Typ Nullable(Of Boolean) gespeichert werden können. Klicken Sie auf *Zustand speichern*, um den Zustand des CheckBox-Steuerelements in der Member-Variablen zu sichern, verändern Sie anschließend den Zustand, und stellen Sie den ursprünglichen Zustand des CheckBox-Steuerelements mit der entsprechenden Schaltfläche wieder her.

Der entsprechende Code dazu lautet folgendermaßen:

```
Public Class Form1
    Private myCheckBoxZustand As Nullable(Of Boolean)

    Private Sub btnOK_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnOK.Click
        Me.Close()
    End Sub

    Private Sub btnSpeichern_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles _
        btnSpeichern.Click
        If chkDemo.CheckState = CheckState.Indeterminate Then
            myCheckBoxZustand = Nothing
        Else
            myCheckBoxZustand = chkDemo.Checked
        End If
    End Sub

    Private Sub btnWiederherstellen_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
        Handles btnWiederherstellen.Click
        If Not myCheckBoxZustand.HasValue Then
            chkDemo.CheckState = CheckState.Indeterminate
        Else
            If myCheckBoxZustand.Value Then
                chkDemo.CheckState = CheckState.Checked
            Else
                chkDemo.CheckState = CheckState.Unchecked
            End If
        End If
    End Sub
End Class
```

Die Zeilen, in denen die Member-Variable `Nullable(Of Boolean)` zum Einsatz kommt, sind im Listing fett markiert. Dabei fällt Folgendes auf:

- **Wertzuweisung:** Wenn Sie einen Wert des zugrunde liegenden Typs an `Nullable(Of )` zuweisen wollen, können Sie die implizite Konvertierung verwenden, den entsprechenden Wert also direkt zuweisen, etwa wie in der Zeile

```
myCheckBoxZustand = chkDemo.Checked
```

zu sehen.

- **Auf Nothing zurücksetzen:** Möchten Sie eine `Nullable`-Instanz auf `Nothing` zurücksetzen, weisen sie ihr einfach den »Wert« `Nothing` zu – wie im Listing an dieser Stelle zu sehen:

```
myCheckBoxZustand = Nothing
```

- **Auf Wert prüfen:** Möchten Sie wissen, ob eine `Nullable`-Instanz einen Wert oder `Nothing` enthält, verwenden Sie deren Eigenschaft `HasValue`. Auch dafür gibt es ein Beispiel im Listing:

```
If Not myCheckBoxZustand.HasValue Then  
    chkDemo.CheckState = CheckState.Indeterminate  
Else  
    .  
    .  
    .
```

- **Wert abrufen:** Und schließlich müssen Sie natürlich auch den Wert, den eine `Nullable`-Instanz trägt, wenn sie nicht `Nothing` ist, ermitteln können. Dazu dient die Eigenschaft `Value`. Ein Beispiel dafür:

```
If myCheckBoxZustand.Value Then  
    chkDemo.CheckState = CheckState.Checked  
Else  
    chkDemo.CheckState = CheckState.Unchecked  
End If  
. . .
```

---

**HINWEIS:** Erst in diesem Beispiel fiel auf, dass man offensichtlich ein CheckBox-Steuerelement, dessen `ThreeState`-Eigenschaft gesetzt ist und das den *Intermediate*-Zustand momentan trägt, nicht mit seiner `Checked`-Eigenschaft in einen anderen Zustand versetzen kann (`Checked` oder `Unchecked`). Sie können in diesem Fall nur die `CheckState`-Eigenschaft verwenden, um das CheckBox-Steuerelement programmgesteuert wieder aus dem *Intermediate*-Zustand herauszuholen!

---

## Besonderheiten bei `Nullable(Of )` beim Boxen

Der Datentyp `Nullable(Of )` ist das, was man in Visual Basic als Struktur programmieren würde, also ein Wertetyp. Doch Sie könnten diesen Wertetyp nicht 1:1 nachprogrammieren, denn er erfährt durch die Common Language Runtime eine besondere Behandlung – und das ist auch gut so.

---

**BEGLEITDATEIEN:** Das folgende Beispiel finden Sie im Verzeichnis .\VB 2005 - Entwicklerbuch\E - Datentypen\Kap20\NullableDemo).

---

Wenn Sie eine Instanz einer beliebigen Struktur – also eines beliebigen Wertetyps – verarbeiten, kommt irgendwann der Zeitpunkt, zu dem Sie diesen Wertetyp in einer Objektvariablen boxen müssen – beispielsweise wenn Sie ihn als Bestandteil eines Arrays oder einer Auflistung (Collection) speichern.

Wann immer Sie einen definierten Wertetyp in einem Objekt boxen, kann dieses Objekt logischerweise nicht Nothing sein, ganz egal, welchen »Wert« diese Struktur hat. Im Falle des Typs Nullable(Of ) ist das anders, wie das folgende Beispiel zeigt:

Module NullableDemo

```
Sub Main()
    Dim locObj As Object
    Dim locNullOfInt As Nullable(Of Integer) = Nothing

    'Es gibt natürlich eine verwendbare Instanz, denn
    'Nullable(of ) ist ein instanzierter Wertetyp!
    Console.WriteLine("Hat locNullOfInt einen Wert:" & locNullOfInt.HasValue)

    'Und dennoch ergibt das folgende Konstrukt True,
    'als würde locObj keine Referenz haben!
    locObj = locNullOfInt
    Console.WriteLine("Ist locObj Nothing?" & (locObj Is Nothing).ToString)
    Console.WriteLine()

    'Und auch das "Entboxen" geht!
    'Es gibt keine Null-Exception!
    locNullOfInt = DirectCast(locObj, Nullable(Of Integer))

    'Und geht das dann auch? - Natürlich!
    locNullOfInt = DirectCast(Nothing, Nullable(Of Integer))

    'Und noch weiter. Wir boxen einen Nullable(of Integer)
    locNullOfInt = 10
    '

    locObj = locNullOfInt
    Dim locInt As Integer = DirectCast(locObj, Integer)

    Console.WriteLine("Taste drücken zum Beenden!")
    Console.ReadKey()

    'Das geht übrigens nicht, obwohl Nullable die Contraints-Einschränkung im
    'Grunde genommen erfüllt!
    'Dim locNullOfInt As Nullable(Of Nullable(Of Integer))
End Sub
End Module
```

Wenn Sie dieses Beispiel ausführen, gibt es die folgenden Zeilen aus:

```
Hat locNull0fInt einen Wert:False  
Ist locObj Nothing?True
```

Taste drücken zum Beenden!

Das, was hier passiert, ist beileibe keine Selbstverständlichkeit – aber dennoch sauberes Design der CLR, denn: Zwar wird `locNull0fInt` nicht initialisiert (oder, um es in diesem Beispiel deutlich zu machen, mit `Nothing` – aber das kommt auf dasselbe raus), aber natürlich existiert dennoch eine Instanz der Struktur. Sie spiegelt eben nur den Wert `Nothing` wider. Gemäß den bekannten Regeln müsste das anschließende Boxen in der Variablen `locObj` auch ergeben, dass `lobObj` einen Zeiger auf eine Instanz der `Nothing` widerspiegelnden `locNull0fInt` enthält und keinen Null-Zeiger. Doch das ist nicht der Fall, denn die anschließende Ausgabe von

```
Console.WriteLine("Ist locObj Nothing?" & (locObj Is Nothing).ToString)
```

zeigt

```
Ist locObj Nothing?True  
auf dem Bildschirm an.
```

Das »zurückcasten« von `Nothing` in ein `Nullable(Of )` ist damit natürlich genau so gestattet, wie ebenfalls im Listing zu sehen.

Und noch eine Unregelmäßigkeit erfahren Nullables, nämlich wenn es darum geht, einen geboxeden Typ (vorausgesetzt er ist eben nicht `Nothing`) in seinen Grundtyp zurückzukasten, wie der folgende Codeausschnitt zeigt:

```
'Und noch weiter. Wir boxen einen Nullable(of Integer)  
locNull0fInt = 10  
'  
locObj = locNull0fInt  
Dim locInt As Integer = DirectCast(locObj, Integer)
```

Hier wird ein `Nullable(Of )`-Datentyp in einem Objekt geboxed, aber später zurück in seinen *Grunddatentypen* gewandelt. Ein, wie ich finde, logisches Design, was allerdings dem »normalen« Vorgehen beim Boxen von Wertetypen in Objekten völlig widerspricht.

Kleine Randnotiz: Dieses Verhalten ist erst zu einem sehr, sehr späten Zeitpunkt beim Entwickeln von Visual Studio 2005 und dem Framework in die CLR eingebaut worden und hat für erhebliche Mehrarbeit bei allen Entwicklerteams und viel zusätzlichen Testaufwand gesorgt. Dass sich Microsoft dennoch für das nunmehr implementierte Verhalten entschieden hat, geht nicht zuletzt auf das Drängen von Kunden und Betatestern zurück, die das Design mit der ursprünglichen, »normalen« CLR-Behandlung von Nullables nicht akzeptieren konnten und als falsch erachteten.

## Generische Auflistungen (Generic Collections)

Generische Auflistungen haben im Vergleich zu »herkömmlichen« Auflistungen, die Sie bereits im vorherigen Kapitel kennen gelernt haben, einen entscheidenden Vorteil: Sie sind grundsätzlich typsicher. Anders als »normale« Auflistungen wie beispielsweise die `ArrayList`-Klasse nehmen sie, da

sie nicht auf Object basieren, nicht jeden Datentyp als Element entgegen, sondern beschränken sich auf den Datentyp, der ihrer Definition zugrunde liegt.

Das bedeutet: Definieren Sie beispielsweise eine Collection auf Basis von Integer, etwa mit

```
Dim locGenColl As New Collection(Of Integer)
```

dann laufen Sie nicht Gefahr, später versehentlich der Auflistung ein Element hinzuzufügen, das nicht vom Typ Integer ist – oder mit anderen Worten: Die Zeile

```
locGenColl.Add("Ein Element")
```

würde bereits zur Entwurfszeit im Editor als fehlerhaft gekennzeichnet.

Ihre Programme werden durch den Einsatz von generischen Auflistungen somit robuster, und auch die Entwicklungszeit reduziert sich, da Sie sich nicht mit Laufzeitfehlern herumärgern müssen, sondern bereits zur Entwurfszeit Fehler korrigieren können. Und weniger Programmtests bedeuteten weniger Entwicklungszeit und -kosten.

Nun gibt es nicht wenige generische Auflistungen seit dem Framework 2.0, und sie alle im Detail zu beschrieben ist nicht nur unnötig – schließlich funktionieren viele von ihnen wie ihre nicht generischen Verwandten – es würde auch den Rahmen des Buches sprengen.

Aus diesem Grund möchte ich mich auf die in meinen Augen wichtigen Besonderheiten beschränken, die Sie bei nicht-generischen Auflistungen nicht antreffen. Eine Tabelle, die Sie im Folgenden finden, gibt darüber hinaus Auskunft, welche generischen Auflistungen Ihnen zur Verfügung stehen, und zu welchem Zweck sie dienen.

Namespace	Auflistung	Beschreibung
System.Collection.ObjectModel	Collection(Of type) Ein Beispiel finden Sie in ► Kapitel 15.	Stellt eine Standardauflistung für die einfache, unsortierte Verwaltung von Elementen eines bestimmten Typs zur Verfügung.  <b>Besonderheiten:</b> Im Gegensatz zu List(Of type) gibt es überschreibbare Methoden, mit denen man das Verhalten beim Einfügen, Löschen und Neuzuweisen von Elementen der Auflistung in abgeleiteten Klassen beeinflussen kann.
System.Collection.ObjectModel	KeyedCollection(Of key,type) Ein Beispiel finden Sie im ► Abschnitt »KeyedCollection – Schlüssel/Wörterbuch-Auflistung mit zusätzlichen Index-Abrufen« ab Seite 602.	Stellt eine Auflistung von Elementen zur Verfügung, die sowohl das Nachschlagen über einen Schlüssel (Key) eines bestimmten Typs sowie den Einsatz eines Indexers erlaubt.  <b>Besonderheiten:</b> Diese Auflistung können Sie nur in Ableitungen verwenden, da der Typ, den Sie speichern, selber einen Standard-Schlüssel erzeugen muss, und für diesen Umstand müssen Sie in der Ableitung sorgen. <b>WICHTIG:</b> Vermeiden Sie den Einsatz von numerischen Schlüsseln (Integer, Long, etc.), da es hier beim Serialisieren der Auflistung zu Problemen kommt (Stand: Framework-Version 2.0.50727). ►

Namespace	Auflistung	Beschreibung
System.Collection.ObjectModel	ReadOnlyCollection(Of type)	<p>Stellt eine Auflistung dar, deren Elemente nur gelesen werden können.</p> <p><b>Besonderheiten:</b> Elemente, die in einer anderen generischen Auflistung gleichen Typs vorliegen, können nur bei der Instanzierung im Konstruktor dieser Auflistung übergeben werden. Ansonsten sind die Elemente nur lesbar und können nach der Instanzierung nicht mehr verändert werden.</p>
System.Collections.Generic	Dictionary(Of key, type)	<p>Stellt eine Auflistung von Schlüsseln und Werten dar.</p> <p><b>Besonderheiten:</b> Stellt eine Zuordnung von einem Satz von Schlüsseln zu einem Satz von Werten bereit. Jede Hinzufügung zum Wörterbuch besteht aus einem Wert und dem zugeordneten Schlüssel. Ein Wert kann anhand des zugehörigen Schlüssels sehr schnell abgerufen werden (beinahe ein O(1)-Vorgang), da die Dictionary-Klasse in Form einer Hashtable implementiert ist. Mehr zum Konzept von Hashtables finden Sie in ► Kapitel 19.</p>
System.Collections.Generic	<p>LinkedList(Of type)</p> <p>Ein Beispiel finden Sie im ► Abschnitt »Elementverkettungen mit LinkedList(Of )« ab Seite 605.</p>	<p>Stellt eine doppelt verknüpfte Liste dar.</p> <p><b>Besonderheiten:</b> Ist eine verknüpfte Liste mit einzelnen Knoten vom Typ <code>LinkedListNode</code>; das Einfügen und Entfernen einzelner Elemente geht extrem schnell vonstatten.</p>
System.Collections.Generic	<p>List(Of type)</p> <p>Ein Beispiel finden Sie in ► Kapitel 27.</p>	<p>Stellt eine Standardauflistung für die einfache, unsortierte Verwaltung von Elementen eines bestimmten Typs zur Verfügung.</p> <p><b>Hinweis:</b> Diese Klasse eignet sich nicht für Ableitungen in eigenen Auflistungsklassen, bei denen Sie mit Code Einfluss auf die Bearbeitung der Liste nehmen müssen. Verwenden Sie stattdessen die <code>Collection(Of )</code>-Auflistung (siehe oben).</p>
System.Collections.Generic	Queue(Of type)	<p>Stellt eine FIFO-Auflistung (First-In-First-Out) von Objekten dar.</p> <p><b>Hinweis:</b> Prinzipiell funktioniert diese Auflistung wie ihr nicht generischer Verwandter. Mehr zur nicht generischen Version dieser Auflistung erfahren Sie in ► Kapitel 19.</p>
System.Collections.Generic	SortedDictionary(Of key, type)	<p>Stellt eine Auflistung von Schlüssel-Wert-Paaren dar, deren Reihenfolge anhand des Schlüssels bestimmt wird.</p> <p><b>Hinweis:</b> Im Gegensatz zu <code>SortedList</code> erfolgt die Sortierung über den Schlüssel und nicht über das verwaltete Element! ►</p>

Namespace	Auflistung	Beschreibung
System.Collections.Generic	SortedList(Of key, type)	<p>Verwaltet eine sortierte Liste, deren Elemente über Schlüssel abrufbar sind.</p> <p><b>Hinweis:</b> Prinzipiell funktioniert diese Auflistung wie ihr nicht generischer Verwandter. Mehr zur nicht generischen Version dieser Auflistung erfahren Sie in ► Kapitel 19.</p> <p>Im Gegensatz zu SortedDictionary erfolgt die Sortierung über das Element und nicht über den Schlüssel!</p>
System.Collections.Generic	Stack(Of type)	<p>Stellt eine LIFO-Auflistung (Last-In-First-Out) von Objekten dar.</p> <p><b>Hinweis:</b> Prinzipiell funktioniert diese Auflistung wie ihr nicht generischer Verwandter. Mehr zur nicht generischen Version dieser Auflistung erfahren Sie in ► Kapitel 19.</p>

**Tabelle 3.1:** Die wichtigsten generischen Auflistungstypen

## KeyedCollection – Schlüssel/Wörterbuch-Auflistung mit zusätzlichem Index-Abrufen

Um es ohne Umschweife zu sagen: Die KeyedCollection ist in meinen eigenen Projekten neben der Collection(Of ) am häufigsten anzutreffen. Warum? Sie hat zwei entscheidende Vorteile:

- Sie erlaubt es, einen beliebigen Typ (na ja – *fast* beliebigen Typ, doch dazu später mehr) als Schlüssel anzugeben. Damit können Sie sie als Wörterbuchaufstellung verwenden und ihre Elemente beispielsweise über einen Matchcode abfragen.
- Sie können Sie zusätzlich wie eine ganz normale Collection behandeln, also mit For/Each durch ihre Elemente iterieren oder auch einzelne Elemente über einen numerischen Index abrufen.

Da KeyedCollection auch einen Schlüsselwert zum Abrufen jedes ihrer Elemente braucht, benötigt sie einen Mechanismus, der aus dem Element, das es hinzuzufügen gilt, einen Schlüssel erzeugt. Aus diesem Grund ist die KeyedCollection-Klasse auch eine abstrakte Basisklasse: Sie müssen die KeyedCollection vererben und ihre Methode GetKeyForItem überschreiben, in der Sie dann regeln, wie aus dem Element, das es hinzuzufügen gilt, ein eindeutiger Schlüssel kreiert werden soll. Die KeyedCollection-Klasse sollte deswegen nur Elemente verwalten, die über wenigstens eine Eigenschaft verfügen, mit der sich ein Element eindeutig von einem anderen unterscheiden lässt. Das kann beispielsweise eine Personalnummer, eine Kundennummer, ein eindeutiger Matchcode oder Ähnliches sein.

---

**WICHTIG:** KeyedCollection hat aber auch einen entscheidenden Nachteil, oder besser: einen Design-Fehler. Wenn Sie ihren Inhalt serialisieren – also beispielsweise mit einem bestimmten im Framework eingebauten Mechanismus als XML-Datei abspeichern –, können Sie als Schlüssel *keinen* Integer-Datentyp verwenden, da das Framework beim Serialisierungsversuch mit einer Ausnahme »aussteigt«. ► Kapitel 22 geht näher auf dieses Problem ein und hält auch einen Workaround zu dieser »Designunzulänglichkeit« bereit.

---

---

**BEGLEITDATEIEN:** Sie finden das Projekt für das folgende Beispiel im Verzeichnis \VB 2005 - Entwicklerbuch\E-Datentypen\Kap20\KeyedCollectionDemo\.

---

Im Projekt finden Sie zwei Codedateien, *Daten.vb* und *KeyedCollection.vb*. *Daten.vb* enthält eine Adressenklasse, die Sie aus dem vorherigen Kapitel schon kennen. Aufgabe dieser Klasse ist es, die Daten einer Adresse zu speichern. Sie stellt darüber hinaus auch eine statische Funktion bereit, die eine bestimmbare Anzahl von Zufallsadressen generiert und in einer ArrayList zurückliefert (mehr zu ArrayList finden Sie ebenfalls im vorherigen Kapitel).

Damit die Klasse *Adresse* in diesem Beispiel als Verwaltungselement für die KeyedCollection in Frage kommt, muss sie eine Anforderung erfüllen: Sie muss über eine Eigenschaft verfügen, mit der sich ein Adresse-Element von jedem anderen Adresse-Element unterscheiden lässt.

---

**HINWEIS:** Die Funktionalität zur Prüfung auf Eindeutigkeit kann die Adresse-Klasse natürlich nicht selbst implementieren, da sie die anderen Elemente einer Auflistung nicht kennt; deswegen ist die Bestimmung einer solchen Dateneigenschaft an dieser Stelle nur eine theoretische Definition. Es ist Aufgabe des späteren Hauptprogramms zu prüfen, ob es keine »Schlüsseldoublette« in einer Elementauflistung gibt, bevor ein neues Element (wie Adresse) der Auflistung hinzugefügt wird.

---

KeyedCollection ist eine *abstrakte* Basisklasse, was bedeutet, dass Sie sie nicht direkt verwenden können. Sie müssen Sie vererben und um eine bestimmte Funktionalität erweitern. Deswegen definieren Sie – wie Sie es von abstrakten Klassen gelernt haben – zunächst einen neuen Klassennamen, etwa mit

```
Public Class Adressen
```

und fügen darunter die Zeile

```
Inherits KeyedCollection(Of String, Adresse)
```

ein. Mit dieser Zeile leiten Sie von KeyedCollection ab und bestimmten gleichzeitig, dass String der Typ für den Schlüssel und Adresse der Typ der zu verwaltenden Elemente sein soll.

In dem Moment, in dem Sie die Zeile mit **Eingabe** abschließen, fügt der Visual Basic-Editor automatisch den Rumpf der Methode ein, den Sie in der Basisklasse überschreiben müssen:

```
Protected Overrides Function GetKeyForItem(ByVal item As Adresse) As String  
End Function
```

Mit dieser Funktion »holt« sich KeyedCollection den Wert, der vom Schlüsseldatentyp sein muss, immer dann, wenn Sie ein neues Element der Auflistung (beispielsweise durch Add) hinzufügen oder ein Element verändern (zum Beispiel mit Item(Index)=New Adresse(..)). So wird sichergestellt, dass jedes Element über einen eindeutigen Schlüssel verfügt.

In unserem Beispiel gibt es eine Eigenschaft in der Klasse *Adresse*, die ein Element vom Typ *Adresse* eindeutig kennzeichnet: *Matchcode*. Diese Eigenschaft ist vom Typ *String* – also ist der Schlüsseltyp für die KeyedCollection ebenfalls *String*. Die komplett implementierte KeyedCollection sieht also folgendermaßen aus:

```

''' <summary>
''' Verwaltet Objekte vom Typ Adresse als Wörterbuch-Auflistung.
''' Abgeleitet von der abstrakten Basisklasse KeyedCollection.
''' </summary>
''' <remarks></remarks>
Public Class Adressen
    Inherits KeyedCollection(Of String, Adresse)

    'Diese Methode muss überschrieben werden, um zu garantieren,
    'dass für jedes Element ein Schlüssel existiert.
    Protected Overrides Function GetKeyForItem(ByVal item As Adresse) As String
        'Hier wird bestimmt, dass der eindeutige Key der Matchcode ist.
        Return item.Matchcode
    End Function
End Class

```

Diese Implementierung ist auch schon alles, was Sie brauchen, um anschließend mit der KeyedCollection arbeiten zu können. Das Hauptmodul *KeyedCollection.vb* demonstriert den Umgang und stellt insbesondere heraus, dass Elemente einer auf KeyedCollection basierenden Klasse sowohl über einen Index als auch über den entsprechenden Schlüssel abgerufen und bearbeitet werden können:

```

Imports System.Collections.ObjectModel

Module KeyedCollection

Sub Main()
    'Erstmal eine normale ArrayList definieren,
    'die aus 100 Zufallsadressen besteht.
    Dim locArrayList As ArrayList = Adresse.ZufallsAdressen(100)

    'Das ist die "selbst gestrickte" KeyedCollection
    Dim locKeyedAdressen As New Adressen

    'Mit den Elementen der ArrayList füllen wir die KeyedCollection
    For Each locAdressItem As Adresse In locArrayList
        locKeyedAdressen.Add(locAdressItem)
    Next

    'Abrufen der Elemente aus der KeyedCollection:
    'Variation Nr.1 - abrufen über den Index
    For c As Integer = 0 To 10
        Console.WriteLine(locKeyedAdressen(c).ToString())
    Next

    'Leerzeile - der besseren Übersicht wegen:
    Console.WriteLine()

    'Irgendeine herauspicken, um an den Matchcode zu kommen:
    Dim locMatchcode As String = locKeyedAdressen(10).Matchcode

```

```

'Variation Nr. 2: Eine Adresse kann auch über den Key
'(in diesem Fall über den Matchcode) abgerufen werden.
Dim locAdresse As Adresse = locKeyedAdressen(locMatchcode)
Console.WriteLine("Die Adresse mit dem Matchcode " & locMatchcode & " lautet:")
Console.WriteLine(locAdresse.ToString())
Console.WriteLine()

Console.WriteLine("Taste drücken zum Beenden!")
Console.ReadKey()

End Sub

End Module

```

## Elementverkettungen mit LinkedList(Of )

Erwähnenswert bei den generischen Auflistungen ist die `LinkedList`-Klasse, die das erste Mal mit dem .NET-Framework 2.0 die Möglichkeit zur Verwaltung von Elementen in Form von verketteten Listen einführt.

`LinkedList` stellt eine verknüpfte Liste mit einzelnen Knoten vom Typ `LinkedListNode` dar. Durch das Konzept von `LinkedList`, bei dem die zu speichernden Elemente jeweils selbst auf das nächste und das vorherige Elemente »zeigen«, kann das Einfügen bzw. Löschen von Elementen sehr schnell vonstatten gehen, da – sinnbildlich – die Kette an einer Stelle nur aufgeknüpft werden muss, um ein neues Glied in die Kette einzufügen.

Durch die Art der Elementverwaltung bietet `LinkedList` Methoden und Eigenschaften, die Sie in anderen Auflistungen nicht finden. Die folgende Tabelle gibt Aufschluss über die besonderen Methoden und Eigenschaften der `LinkedList`-Klasse.

Elemente werden listenintern nicht wie bei anderen Auflistungsklassen direkt gespeichert, sondern in einem Knoten-Objekt namens `LinkedListNode` abgelegt, das für die Verkettung zum nächsten bzw. vorherigen Knoten der Liste sorgt. Auch diese Klasse ist generisch ausgelegt. Knoten, die der Liste hinzugefügt werden, müssen dabei auf Basis desselben Typs definiert werden, wie die Liste selbst.

Methode	Beschreibung
AddAfter	Fügt nach einem vorhandenen Knoten in der <code>LinkedList</code> einen neuen Knoten oder Wert hinzu.
AddBefore	Fügt vor einem vorhandenen Knoten in der <code>LinkedList</code> einen neuen Knoten oder Wert hinzu.
AddFirst	Fügt am Anfang der <code>LinkedList</code> einen neuen Knoten oder Wert hinzu.
AddLast	Fügt am Ende der <code>LinkedList</code> einen neuen Knoten oder Wert hinzu.
Find	Sucht den ersten Knoten, der den angegebenen Wert enthält.
FindLast	Sucht den letzten Knoten, der den angegebenen Wert enthält.
First	Ruft den ersten Knoten der <code>LinkedList</code> ab. Diese Eigenschaft kann nur gelesen werden.
Last	Ruft den letzten Knoten der <code>LinkedList</code> ab. Diese Eigenschaft kann nur gelesen werden.
Remove	Entfernt das erste Vorkommen eines Knotens oder Werts aus der <code>LinkedList</code> . ►

Methode	Beschreibung
RemoveFirst	Entfernt den Knoten am Anfang der LinkedList.
RemoveLast	Entfernt den Knoten am Ende der LinkedList.

**Tabelle 20.2:** Die besonderen Methoden und Eigenschaften der LinkedList-Klasse

**BEGLEITDATEIEN:** Das folgende Beispiel, das den generellen Umgang mit den Klassen LinkedList und LinkedListNode demonstriert, finden Sie im Verzeichnis „\VB 2005 - Entwicklerbuch\E - Datentypen\Kap20\LinkedList“.

Module LinkedList

```

Sub Main()
    Dim locLinkedList As New LinkedList(Of Adresse)
    Dim locAdressen As ArrayList = Adresse.ZufallsAdressen(50)
    Dim locAdresse As Adresse = Nothing

    'Acht Adressen an das jeweilige Ende der LinkedList anfügen
    For c As Integer = 0 To 49

        'den 25. merken wir uns für die spätere Suche in der Liste
        If c = 25 Then
            locAdresse = DirectCast(locAdressen(c), Adresse)
        End If
        locLinkedList.AddLast(DirectCast(locAdressen(c), Adresse))
    Next

    Dim locLinkedListNode As LinkedListNode(Of Adresse)
    'Den Knoten finden, der dem 25. Adresseneintrag entsprach.
    locLinkedListNode = locLinkedList.Find(locAdresse)
    Console.WriteLine("Vor " & locLinkedListNode.Value.ToString & " kommt " & _
                      locLinkedListNode.Previous.Value.ToString & _
                      " und danach kommt " & locLinkedListNode.Next.Value.ToString)
    Console.WriteLine()

    'Eine neue Adresse vor dem 25. einfügen:
    Console.WriteLine("Sarah Halek vorher einfügen!")
    locLinkedList.AddBefore(locLinkedListNode,
                           New Adresse("SasiMatch", "Halek", "Sarah", "99999", "Musterhausen"))

    Console.WriteLine()

    'Das Gleiche nochmal ausgeben, es sollte die Änderungen jetzt widerspiegeln.
    Console.WriteLine("Vor " & locLinkedListNode.Value.ToString & " kommt " & _
                      locLinkedListNode.Previous.Value.ToString & _
                      " und danach kommt " & locLinkedListNode.Next.Value.ToString)
    Console.WriteLine()
    Console.WriteLine("Taste drücken zum Beenden")
    Console.ReadKey()
End Sub
End Module

```

Wenn Sie dieses Programm ausführen, gibt es in etwa Folgendes auf dem Bildschirm aus:

Vor "Hörstmann, Hans" kommt "Englisch, Margarete" und danach kommt "Löffelmann, Katrin"

Sarah Halek vorher einfügen!

Vor "Hörstmann, Hans" kommt "Halek, Sarah" und danach kommt "Löffelmann, Katrin"

Taste drücken zum Beenden

## Auflistungen und Aktionen (Actions), Aussageprüfer (Predicates) und Vergleiche (Comparisons)

Über das folgende Thema konnte man bislang noch nicht viel erfahren. Auch die Online-Hilfe schweigt sich darüber einigermaßen aus – wie ich finde zu Unrecht, denn ein weiteres Mal zeigt sich, dass man sich mit dem Konzept von generischen Datentypen die Arbeit an einigen Stellen erheblich erleichtern kann.

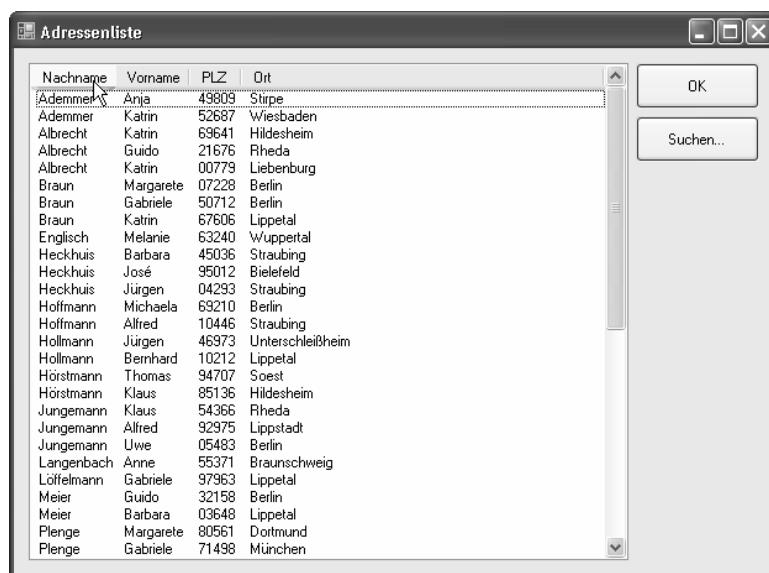


Abbildung 20.2: Sortierung und Suche steuern Sie über die Spaltenköpfe in dieser Adressenliste

---

**BEGLEITDATEIEN:** Sie finden das folgende Beispielprojekt im Verzeichnis `.\VB 2005 - Entwicklerbuch\E - Datentypen\Kap20\StaticArrayFunctions`.

---

Wenn Sie das Programm starten, sehen Sie einen Dialog, etwa wie Abbildung 20.2 zu sehen. Die im Formular vorhandene ListView wird mit 50 Zufallsadressen gefüllt. Sie können die Liste sortieren, indem Sie auf den entsprechenden Spaltenkopf klicken – so, wie Sie es vom Windows-Explorer in der Details-Ansicht gewohnt sind.

Eine Suchfunktion, die Sie über die entsprechende Schaltfläche erreichen, gestattet es Ihnen, nach dem Begriff innerhalb der Liste zu suchen. Gefunden wird dabei der Eintrag, dessen Text in der Spalte, nach der zuletzt sortiert wurde, dem eingegebenen Suchbegriff entspricht.

Soweit ist dieses Beispiel noch nichts Besonderes. Doch wenn Sie einen Blick in die Umsetzung riskieren, wird klar, wie alternativ der Lösungsansatz ist. Das geht schon beim Schreiben der Zufallsadressen in die Liste los:

## ForEach und die generische Action-Klasse

Wenn Sie den Inhalt eines Arrays oder einer Auflistung in einer Liste darstellen möchten, liegt die Vorgehensweise, um das zu erreichen, eigentlich auf der Hand: Sie iterieren mit einem For/Each-Konstrukt durch die Auflistung, verarbeiten damit jedes Element und bringen es mit geeigneten Methoden oder Eigenschaften in die sichtbare Liste eines Formulars.

Dieses Beispiel nutzt einen anderen Weg, wie im folgenden Listingausschnitt zu sehen:

```
Sub ElementeDarstellen()

    'Unterdrückt Neuzeichnen-Ereignisse bis zum
    'nächsten EndUpdate; dadurch geht der Aufbau
    'der Elemente schneller und wackelt nicht.
    Me.lvAdressen.BeginUpdate()

    'Alle Elemente der ListView löschen.
    Me.lvAdressen.Items.Clear()

    'Für jedes Element der Liste wird ElementInListe aufgerufen.
    myAdressen.ForEach(New Action(Of Adresse)(AddressOf ElementInListe))

    'So werden die Spaltenbreiten optimal angepasst.
    For Each locCol As ColumnHeader In Me.lvAdressen.Columns
        locCol.Width = -2
    Next

    'Aufbau der ListView ist beendet.
    Me.lvAdressen.EndUpdate()
End Sub

'Der Action-Delegat: für jedes Element der Liste wird diese Aktion durchgeführt.
Sub ElementInListe(ByVal Element As Adresse)
    'Neues ListView-Element - Name kommt zuerst.
    Dim locLvwItem As New ListViewItem(Element.Name)

    'Die Untereinträge setzen
    With locLvwItem.SubItems
        .Add(Element.Vorname)
        .Add(Element.PLZ)
        .Add(Element.Ort)
    End With
End Sub
```

```

'Zum Wiederfinden Referenz in Tag
lclvItem.Tag = Element

'Zur Listview hinzufügen
lvwAdressen.Items.Add(lclvItem)
End Sub

```

Sie können die `ForEach`-Methode verwenden, um durch eine Auflistung iterieren *zu lassen* und dabei für jedes Element einen Delegaten aufrufen, der eine bestimmte Aktion ausführt. Dieses Verfahren nutzen wir an dieser Stelle, um die Liste aufzubauen. Der Delegat wird im Beispiel nicht durch ein Delegate-Objekt (mehr zu Delegaten finden Sie in ► Kapitel 15) sondern durch die direkte Angabe einer Prozedur mithilfe des `Address Of`-Operators angegeben – sozusagen als Delegatenkonstante. Aber natürlich könnten an dieser Stelle auch »richtige« Delegaten zum Einsatz kommen, die in Abhängigkeit von bestimmten Programmzuständen andere Prozeduren verwendeten, und genau darin besteht der Reiz des Einsatzes dieser Verfahren.

Wichtig dabei ist, dass die Routinen, die Sie mithilfe der `Action`-Klasse aufrufen, den Signaturanspruch des Delegaten erfüllen, den Sie im Konstruktor von `Action` angeben. Im Fall von `ForEach` und der `Action`-Klasse muss es sich bei der Delegatenprozedur um eine Methode handeln, die kein Funktionsergebnis hat (also um eine Visual Basic-Sub) und als Parameter ein Element entgegen nehmen, dessen Typ auch der Basis der generischen `Action`-Klasse entspricht – Adresse in unserem Beispiel.

Im Ergebnis erreichen wir also, dass für jedes Element des `List(Of Adresse)`-Objektes `myAdressen` die Methode `ElementInListe` aufgerufen wird.

## Sort und die generische Comparison-Klasse

Prinzipiell funktioniert das nächste Pärchen ähnlich, das Sie verwenden, wenn Sie ein Array mit der Methode `Sort` ohne den Einsatz einer speziellen `Comparer`-Klasse (wie in ► Kapitel 19 gezeigt) sortieren möchten. Den relevanten Codeausschnitt des Beispiels finden Sie im Folgenden:

```

'Wird aufgerufen, wenn eine der Spalten angeklickt wird.
Private Sub lvwAdressen_ColumnClick(ByVal sender As Object,
    ByVal e As System.Windows.Forms.ColumnEventArgs) Handles lvwAdressen.ColumnClick

    'Spaltennummer, die in e.Column steht, in AdressenSortierenNach konvertieren
    myNach = CType(e.Column, AdressenSortierenNach)

    'Die Auflistung mithilfe eines Comparison-Delegaten sortieren
    myAdressen.Sort(New Comparison(Of Adresse)(AddressOf AdressenVergleicher))

    'Die Elemente neu sortiert darstellen
    ElementeDarstellen()
End Sub

'Der Comparison-Delegat zum Vergleichen zweier Elemente.
Function AdressenVergleicher(ByVal x As Adresse, ByVal y As Adresse) As Integer

```

```

'Sortierung in Abhängigkeit von zuletzt angeklicktem
'Spaltenkopf durchführen (siehe lvwAdressen_ColumnClick)
Select Case myNach
    Case AdressenSortierenNach.Name
        Return x.Name.CompareTo(y.Name)
    Case AdressenSortierenNach.Ort
        Return x.Ort.CompareTo(y.Ort)
    Case AdressenSortierenNach.PLZ
        Return x.PLZ.CompareTo(y.PLZ)
    Case Else
        Return x.Vorname.CompareTo(y.Vorname)
End Select
End Function

```

Sort arbeitet hier in diesem Beispiel abermals mit einem Delegaten, der durch die generische Comparison-Klasse bereitgestellt wird, und dessen dahinter stehende Methode sich um den eigentlichen Vergleich zweier Objekte (wieder Adresse) kümmert. Damit macht diese Vorgehensweise die Implementierung einer speziellen Comparer-Klasse für die Adresse-Klasse überflüssig.

## Find und die generische Predicate-Klasse

Das letzte generische »Dreamteam« schließlich kommt zum Einsatz, wenn es um das Finden eines bestimmten Objektes in einer generischen Liste geht: Find und der Delegat, der durch die generische Predicate-Klasse eingerichtet wird. Auch hier entspricht die grundsätzliche Vorgehensweise dem bereits Bekannten, wie der entsprechende Code im Beispiel zeigt:

```

Private Sub btnSuchen_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
Handles btnSuchen.Click

    'Suchbegriff abfragen
    Dim locSuchFormular As New frmSuchen

    'Den Suchbegriff merken, damit der Predicate-Delegat darauf
    'zugreifen kann.
    myAktuellerSuchbegriff = locSuchFormular.Suchbegriff
    If String.IsNullOrEmpty(myAktuellerSuchbegriff) Then
        Return
    End If

    'Hier kann die Suche beginnen!
    Dim locGefundeneAdresse As Adresse =
        myAdressen.Find(New Predicate(Of Adresse)(AddressOf AdressenFinder))

    'Wenn ein Element gefunden wurde, dieses markieren.
    If locGefundeneAdresse IsNot Nothing Then

        'Alle ListView-Elemente durchsuchen und überprüfen, ob ...
        For Each locLvwItem As ListViewItem In Me.lvwAdressen.Items

            '... die Tag-Referenz der Referenz des gesuchten Objekts entspricht.
            If locLvwItem.Tag Is locGefundeneAdresse Then

```

```

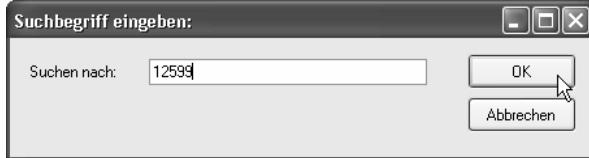
'Gefunden! ListView-Element markieren,
lclLvwItem.Selected = True

'und dafür sorgen, dass es im sichtbaren Bereich liegt.
lclLvwItem.EnsureVisible()
Return
End If
Next
End If
End Sub

'Der Predicate-Delegat zum Suchen eines Elements.
Private Function AdressenFinder(ByVal adr As Adresse) As Boolean

'Gesucht wird immer nach zuletzt ausgewähltem Spaltenkopf.
Select Case myNach
Case AdressenSortierenNach.Name
    Return adr.Name = myAktuellerSuchbegriff
Case AdressenSortierenNach.Ort
    Return adr.Ort = myAktuellerSuchbegriff
Case AdressenSortierenNach.PLZ
    Return adr.PLZ = myAktuellerSuchbegriff
Case Else
    Return adr.Vorname = myAktuellerSuchbegriff
End Select
End Function

```



**Abbildung 20.3:** Der Suchbegriff bezieht sich immer auf die Spalte, nach der zuletzt sortiert wurde

In diesem Fall ist etwas mehr Drumherum notwendig, da der Begriff, nach dem gesucht werden soll, zunächst ermittelt werden muss. Und: Der eigentliche Suchbegriff kann der Prozedur, die sich hinter dem der Predicate-Klasse übergebenden Delegaten verbirgt, nicht mitgegeben werden: er muss also der Delegatenprozedur zur Verfügung stehen und wird daher in `myAktuellerSuchbegriff` als Member-Variable gespeichert.

Eine weitere kleine Herausforderung ist das Selektieren des gefundenen Begriffs in der Liste – und das ist im Übrigen auch der Grund, wieso wir eine Referenz jedes Elements in der Tag-Eigenschaft jedes ListViewItem-Elements speichern: Wenn der Begriff durch `Find` gefunden wurde, liegt uns das entsprechende Adresse-Objekt vor. Mit diesem können wir anschließend durch die ListViewItem-Auflistung iterieren und auf Objektübereinstimmung durch Abfrage der Tag-Eigenschaft testen. Dieser Aufwand ist nötig, da es keine andere Möglichkeit gibt, das richtige ListViewItem-Element zu finden, und nur dieses erlaubt es aber, die richtige Zeile in der Liste durch seine `Select`-Eigenschaft zu selektieren.

---

**TIPP:** Neben `List(Of type)` können Sie auch die generisch ausgelegten, statischen Funktionen von `Array` verwenden, um die hier beschriebenen Arten von Operationen durchzuführen. **Wichtig:** Wenn Sie primitive Datentypen in einer Auflistung speichern, sollten Sie den Einsatz `List(Of type)` anderen Auflistungsklassen vorziehen. Das .NET-Framework ist nämlich in der Lage, den eigentlich notwendigen Boxing-Vorgang bei `List(Of type)` zu umgehen, und damit ist `List(Of type)` die schnellere Alternative.

---

# 21 Reguläre Ausdrücke (Regular Expressions)

---

- 
- 614    RegExperimente mit dem RegExplorer**
  - 616    Erste Gehversuche mit Regulären Ausdrücken**
  - 630    Programmieren von Regulären Ausdrücken**
  - 634    Regex am Beispiel: Berechnen beliebiger mathematischer Ausdrücke**
- 

Reguläre Ausdrücke (*Regular Expressions*) sind eine mächtige Erweiterung der String-Verarbeitung in Visual Basic (eigentlich im Framework). Ungewöhnlich ist auch die Geschichte, die sich dahinter verbirgt, nämlich wie Reguläre Ausdrücke ihren Weg in das Framework gefunden haben. Sie müssen wissen: Die verschiedenen Themengebiete innerhalb des Frameworks werden von eigenen, sehr unabhängigen Entwicklungsteams bei Microsoft entwickelt. Reguläre Ausdrücke waren ursprünglich »nur« eine Erweiterung bzw. ein Werkzeug, die bzw. das im ASP.NET-Team gebraucht wurde. Erst in teamübergreifenden Meetings erkannten auch andere Teams das Vorhandensein von Regulären Ausdrücken, und nun begann ein Tauziehen darum, in welchem Namensbereich die Regex-Klasse, mit der Sie Reguläre Ausdrücke anwenden, letzten Endes ihr zu Hause fand.

Das Ergebnis kennen Sie: Sie finden die Regex-Klasse im Bereich `System.Text.RegularExpressions`. Das bedeutet: Sie müssen die Anweisung

```
Imports System.Text.RegularExpressions
```

an den Anfang einer Klassen-Quellcodedatei setzen, damit Sie auf die Klassen zugreifen können.

Die große Frage, die sich vielen stellt: Was genau sind Reguläre Ausdrücke? Die Wurzeln von Regulären Ausdrücken gehen zurück auf die Arbeiten eines gewissen Stephen Kleene. Stephen Kleene war ein amerikanischer Mathematiker und darüber hinaus einer derjenigen, die die Entwicklung der theoretische Informatik maßgeblich beeinflusst und vorangetrieben haben. Er erfand eine Schreibweise für die, wie er sie nannte, »Algebra regelmäßiger Mengen«. Im Kontext von Suchaufgaben mit dem Computer war das »\*«-Zeichen deshalb bis vor einiger Zeit auch unter dem Namen »Kleene-Star« bekannt.

Und damit sind wir auch schon beim Thema, denn das »\*«-Zeichen als *Joker* oder *Wildcard* hat jeder von Ihnen sicherlich schon einmal unter DOS, zumindest aber in der Konsole verwendet. Wenn Sie in der Konsole beispielsweise alle Dateien anzeigen lassen möchten, die mit ».TXT« enden, geben Sie den Befehl

```
dir *.txt
```

ein. Sie können also bestimmte Sonderzeichen verwenden, um Zeichenfolgen zu finden, welche Regeln unterliegen, die von diesen Sonderzeichen definiert werden. Genau dazu dienen Reguläre Ausdrücke. Dummerweise ist das Demonstrieren von Reguläre Ausdrücke mit simplen Konsolen-Anwendungen nicht sehr anschaulich, vor allen Dingen auch recht mühsam. Denn bei aller Leistungsfähigkeit von Reguläre Ausdrücke haben diese doch einen Nachteil: Sie sind vergleichsweise schwer zu lesen. Wenn Sie sich an die Zusammensetzung von Reguläre Ausdrücke gewöhnt haben, dann wird Ihnen beim zeichenweisen Analysieren klar, wieso die Zeichenfolge

```
[\d, .]+[S]*
```

alle Zahlenkonstanten in einer Formel finden kann. Doch wenn Sie sie anschauen, werden Sie auch mit einiger Übung nicht direkt auf den ersten Blick ihre Funktionsweise durchschauen.

## RegExperimente mit dem RegExplorer

Aus diesem Grund finden Sie in den Begleitdateien (respektive im entsprechenden Verzeichnis Ihrer Festplatte) ein Projekt, mit dem Sie Reguläre Ausdrücke testen können.

---

**BEGLEITDATEIEN:** Sie finden dieses Projekt unter dem Namen *RegExplorer* im Verzeichnis *.\VB 2005 - Entwicklerbuch\E - Datentypen\Kap21\RegExplorer*. Wenn Sie dieses Projekt laden und starten, sehen Sie einen Dialog, wie auch in Abbildung 21.1 zu sehen.

---

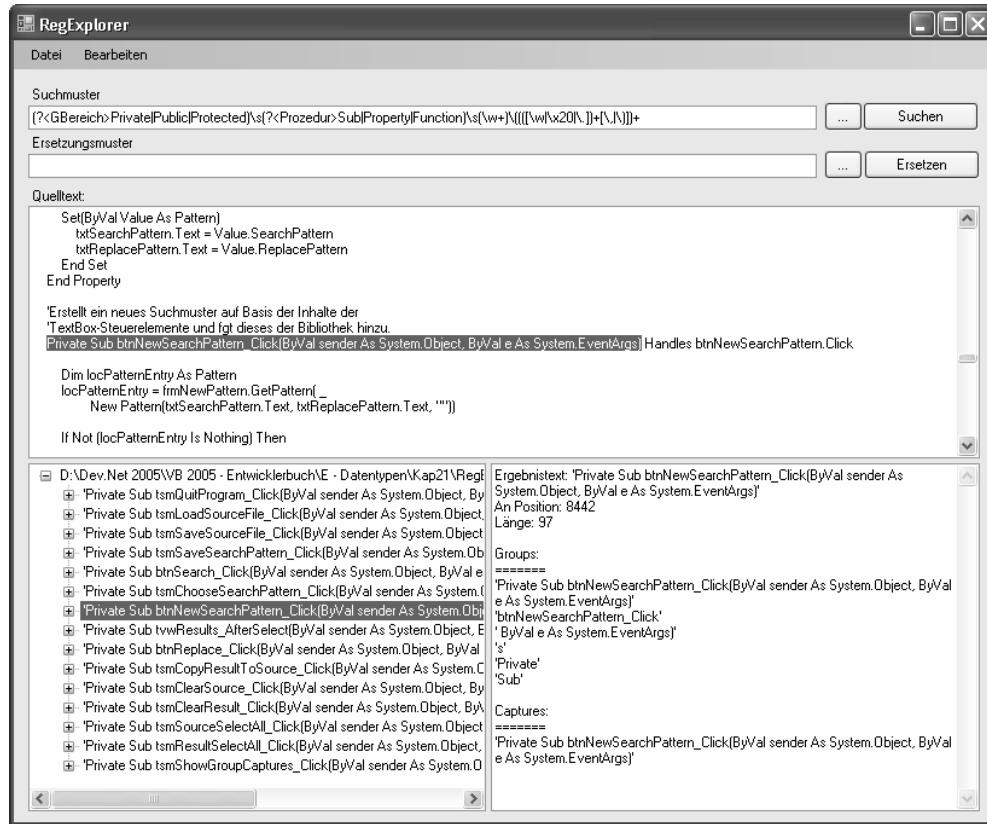
Kurz zur Beschreibung: Nach dem Start des Programms können Sie Texte unter *Quelltext* erfassen, oder Sie können einen Text mit *Datei/Quelltextdatei laden* aus einer beliebigen Datei laden und in der Textbox *Quelltext* anzeigen lassen.

Unter *Suchmuster* können Sie einen Regulären Ausdruck eingeben; ein Mausklick auf die Schaltfläche *Suchen* löst dann die Suche mit dem angegebenen Suchbegriff aus.

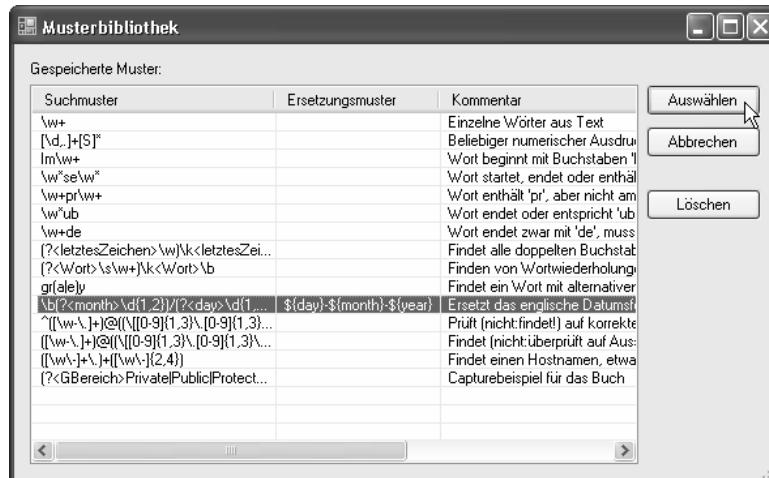
Die verschiedenen Suchergebnisse zeigt Ihnen das TreeView-Steuerelement, das Sie in der linken, unteren Ecke des Programmfensters sehen. Ein Klick auf den Wurzeleintrag bringt die komplette Datei in das rechts daneben stehende Ergebnisfenster.

Ein Mausklick auf einen der untergeordneten Zweige (nur 2. Ebene) zeigt Ihnen Informationen über den gefundenen Begriff an. Gleichzeitig wird der Suchbegriff im Quelltext markiert.

Der RegExplorer hat eine Bibliothek mit häufig verwendeten Regulären Ausdrücken. Sie können einen Ausdruck aus der Bibliothek auswählen, indem Sie auf die Schaltfläche mit der Aufschrift »...« klicken, die sich neben der Schaltfläche *Suchen* befindet.

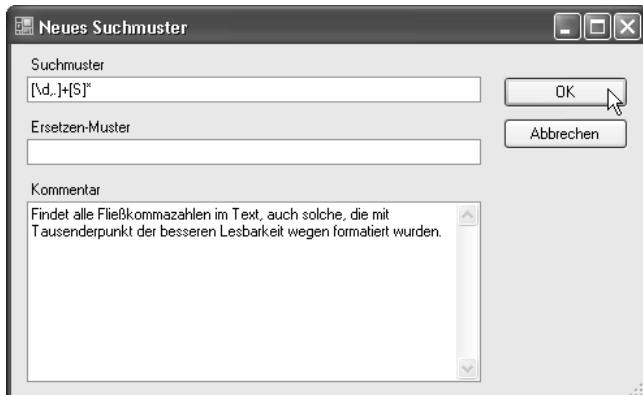


**Abbildung 21.1:** Mit dem RegExplorer können Sie sich mit Regulären Ausdrücken nach Herzenslust austoben, ohne eine Zeile Code schreiben zu müssen!



**Abbildung 21.2:** Damit das Experimentieren leichter wird, können Sie sich Anregungen in der Regex-Bibliothek holen

Möchten Sie einen neuen Bibliothekseintrag anlegen, klicken Sie auf die darunter stehende Schaltfläche, oder wählen Sie aus dem Menü *Datei* den Menüpunkt *Suchmuster speichern*. Der RegExplorer zeigt Ihnen einen weiteren Dialog an, mit dem Sie den neuen Bibliothekseintrag erfassen können (siehe Abbildung 21.3).



**Abbildung 21.3:** Mit diesem Dialog fügen Sie Reguläre Ausdrücke zur Bibliothek hinzu. Dabei müssen mindestens Suchmuster und Kommentar angegeben sein

Klicken Sie auf *Speichern*, um den neuen Eintrag in die Bibliothek aufzunehmen.

Sie können das Programmfenster übrigens nach Belieben in der Größe anpassen und auch die verschiedenen Bereiche innerhalb des Fensters mit den SplitterContainer-Komponenten sowohl horizontal (oberer Parameter- und unterer Ergebnisbereich) als auch vertikal (Verhältnis zwischen linker, unterer Ergebnis-TreeView und rechter, unterer Ergebnis-TextBox) verändern.

## Erste Gehversuche mit Regulären Ausdrücken

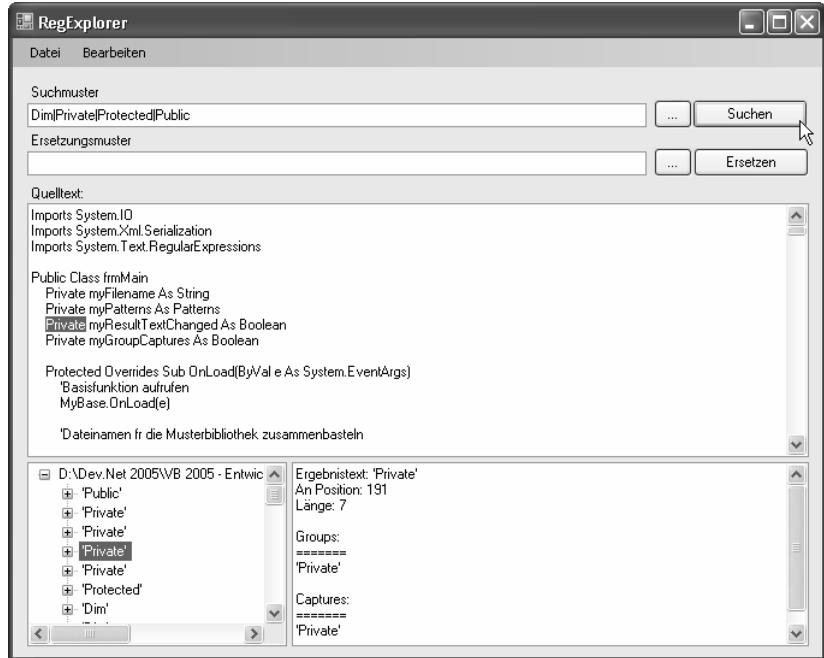
Für die ersten Gehversuche laden Sie am besten den Quellcode des Programms selbst in den Quelltextbereich, um damit experimentieren zu können. Wählen Sie dazu *Quelltext laden* aus dem Menü *Datei*. Im DateiauswahlDialog wählen Sie anschließend den Dateityp *VB-Quelldateien*. Öffnen Sie anschließend die Datei *frmMain.vb*.

### Einfache Suchvorgänge

Zunächst einmal können Sie einfache Zeichenfolgen verwenden, wenn Sie eine exakte Entsprechung für diese im Quelltext finden wollen. Geben Sie beispielsweise

Imports

als Suchbegriff ein und klicken anschließend auf die Schaltfläche *Suchen*, finden Sie in der darunter stehenden Ergebnisliste ausschließlich die Entsprechungen dieses Worts. Bis hierhin ist die Suchfunktion noch nichts Besonderes. Reguläre Ausdrücke werden erst dann interessant, wenn mit Steuerzeichen bestimmte Funktionen beim Suchen (und später auch beim Ersetzen) mit einbezogen werden können.



**Abbildung 21.4:** Wenn Sie nach alternativen Begriffen suchen, trennen Sie durch das »|«-Zeichen. Ein Klick auf den Begriff in der linken, unteren Ergebnisliste markiert übrigens das Wort im Quelltext

Dazu ein simples Beispiel. Mit Hilfe des Oder-Operatorzeichens (»|«) können Sie aus einer Alternative von Zeichenketten Treffer generieren. Suchen Sie beispielsweise nach Wörtern, die wahlweise *Dim*, *Private*, *Protected* oder *Public* heißen sollen, dann formulieren Sie den Suchbegriff folgenderweise:

Dim|Private|Protected|Public

Wenn Sie anschließend auf *Suchen* klicken, sehen Sie ein Ergebnis, etwa wie es Abbildung 21.4 zeigt.

## Einfache Suche nach Sonderzeichen

Nicht alle Zeichen lassen sich über die Tastatur eingeben. Möchten Sie beispielsweise nach doppelten Absätzen suchen, bekommen Sie bei der Eingabe über die Tastatur schon Probleme.

Escape-Zeichen	Beschreibung
Normale Zeichen	Andere Zeichen als \$ ^ { [ ( ) ] * + ? \ } stehen für sich selbst.
\a	Entspricht einem Klingelzeichen (Warnsignal) \u0007. (Bell).
\b	Entspricht in einer [ ]-Zeichenklasse einem Rücktastenzeichen \u0008 (Backspace). <b>Wichtig:</b> Das Escape-Zeichen \b ist ein Sonderfall: In einem Regulären Ausdruck markiert \b eine Wortbegrenzung (zwischen \w und \W-Zeichen), ausgenommen innerhalb einer [ ]-Zeichenklasse, bei der \b das Rücktastenzeichen darstellt. In einem Ersetzungsmuster kennzeichnet \b immer ein Rücktastenzeichen.
\t	Entspricht einem Tabulator \u0009.
\r	Entspricht dem Wagenrücklaufzeichen \u000D (Carriage Return). ►

Escape-Zeichen	Beschreibung
\v	Entspricht dem vertikalen Tabstopnzeichen \u000B, das aber in der Windows-Welt in der Regel keine Anwendung findet.
\f	Entspricht einem Seitenwechselzeichen \u000C (Form Feed).
\n	Entspricht einem Zeilenvorschub \u000A (Line Feed).
\e	Entspricht einem Escape-Zeichen \u001B.
\040	Entspricht einem beliebigen ASCII-Zeichen, das durch eine Oktalzahl (bis zu drei Stellen) repräsentiert wird. Zahlen ohne voran stehende Null sind Rückverweise, wenn sie nur eine Ziffer enthalten oder einer Aufzeichnungsgruppennummer entsprechen. Beispielsweise stellt das Zeichen \040 ein Leerzeichen dar.
\x20	Entspricht einem ASCII-Zeichen in hexadezimaler Darstellung (genau zwei Stellen).
\cC	Entspricht einem ASCII-Steuerzeichen. Beispiel: \cC ist Control-C.
\u0020	Entspricht einem Unicode-Zeichen in hexadezimaler Darstellung (genau vier Stellen).
\	Wird dieses Zeichen von einem Zeichen gefolgt, das nicht als Escape-Zeichen erkannt wird, entspricht es diesem Zeichen. So stellt \. beispielsweise den Punkt oder \\ den Backslash dar.

**Tabelle 21.1:** Gültige Sonderzeichen für Reguläre Ausdrücke

Mit Sonderzeichen, die Sie der Tabelle entnehmen, können Sie das Problem recht simpel lösen:

\r\n\r\n

Diese Sonderzeichen weisen die *Regular Expressions Engine* an, nach dem fortlaufenden Vorkommen von *Carriage Return*, *Line Feed*, *Carriage Return* und *LineFeed* zu suchen – und diese Folge entspricht zwei aufeinander folgenden Absätzen.

## Komplexere Suche mit speziellen Steuerzeichen

Noch flexibler können Sie Ihre Suchvorgänge gestalten, wenn Sie von Steuerzeichen Gebrauch machen, wie sie die folgende Tabelle darstellt:

Zeichenklasse	Beschreibung
.	Entspricht allen Zeichen mit Ausnahme von \n. Bei Modifikation durch die Singleline-Option entspricht ein Punkt einem beliebigen Zeichen. Weitere Informationen hierzu finden Sie unter Optionen für Reguläre Ausdrücke.
[aeiou]	Entspricht einem beliebigen einzelnen Zeichen, das in dem angegebenen Satz von Zeichen enthalten ist.
[^aeiou]	Entspricht einem beliebigen einzelnen Zeichen, das nicht in dem angegebenen Satz von Zeichen enthalten ist.
[0-9a-fA-F]	Durch die Verwendung eines Bindestrichs (-) können aneinander grenzende Zeichenbereiche angegeben werden.
\p{name}	Entspricht einem beliebigen Zeichen in der durch {name} angegebenen benannten Zeichenklasse. Unterstützte Namen sind Unicodegruppen und Blockbereiche. Beispielsweise Ll, Nd, Z, IsGreek, IsBoxDrawing.
\P{name}	Entspricht Text, der nicht in Gruppen und Blockbereichen enthalten ist, die in {name} angegeben werden. ►

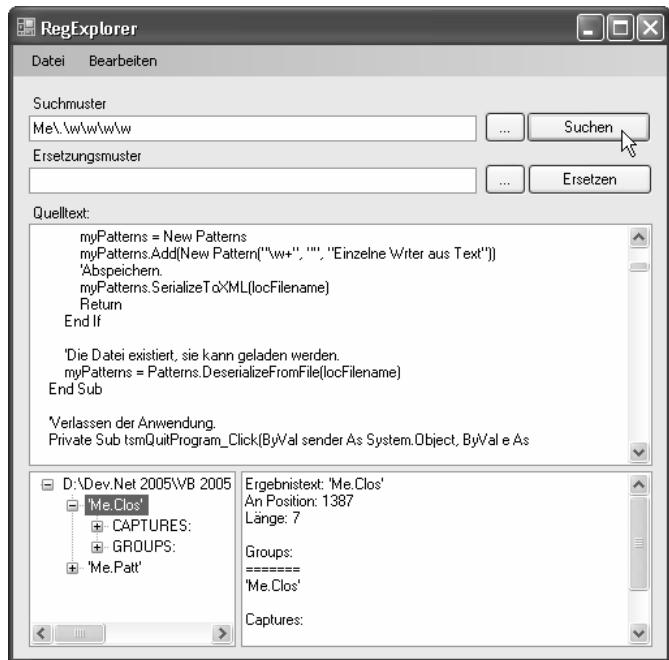
Zeichenklasse	Beschreibung
\w	Entspricht einem beliebigen Wortzeichen. Entspricht den Unicode-Zeichenkategorien [\p{Ll}\p{Lu}\p{Lt}\p{Lo}\p{Nd}\p{Pc}]. Wenn mit der ECMAScript-Option ECMAScript-konformes Verhalten angegeben wurde, ist \w gleichbedeutend mit [a-zA-Z_0-9].
\W	Entspricht einem beliebigen Nichtwortzeichen. Entspricht den Unicodekategorien [^\p{Ll}\p{Lu}\p{Lt}\p{Lo}\p{Nd}\p{Pc}]. Wenn mit der ECMAScript-Option ECMAScript-konformes Verhalten angegeben wurde, ist \W gleichbedeutend mit [^a-zA-Z_0-9].
\s	Entspricht einem beliebigen Leerraumzeichen. Entspricht den Unicode-Zeichenkategorien [\f\n\r\t\v\x85\p{Z}]. Wenn mit der ECMAScript-Option ECMAScript-konformes Verhalten angegeben wurde, ist \s gleichbedeutend mit [\f\n\r\t\v].
\S	Entspricht einem beliebigen Nicht-Leerraumzeichen. Entspricht den Unicode-Zeichenkategorien [^\f\n\r\t\v\x85\p{Z}]. Wenn mit der ECMAScript-Option ECMAScript-konformes Verhalten angegeben wurde, ist \S gleichbedeutend mit [^\f\n\r\t\v].
\d	Entspricht einer beliebigen Dezimalziffer. Gleichbedeutend mit \p{Nd} für Unicode und [0-9] für Nicht-Unicode mit ECMAScript-Verhalten.
\D	Entspricht einer beliebigen Nichtziffer. Gleichbedeutend mit \P{Nd} für Unicode und [^0-9] für Nicht-Unicode mit ECMAScript-Verhalten.
^	Bestimmt, dass der Vergleich am Anfang der Zeichenfolge oder der Zeile erfolgen muss.
\$	Bestimmt, dass der Vergleich am Ende der Zeichenfolge, vor einem \n am Ende der Zeichenfolge oder am Ende der Zeile erfolgen muss.
\A	Bestimmt, dass der Vergleich am Anfang der Zeichenfolge erfolgen muss (die Multiline-Option wird ignoriert).
\Z	Bestimmt, dass der Vergleich am Ende der Zeichenfolge oder vor einem \n am Ende der Zeichenfolge erfolgen muss (die Multiline-Option wird ignoriert).
\z	Bestimmt, dass der Vergleich am Ende der Zeichenfolge erfolgen muss (die Multiline-Option wird ignoriert).
\G	Bestimmt, dass der Vergleich an dem Punkt erfolgen muss, an dem der vorherige Vergleich beendet wurde. Beim Verwenden mit Match.NextMatch() wird sichergestellt, dass alle Übereinstimmungen aneinander grenzend sind.
\b	Bestimmt, dass der Vergleich an einer Begrenzung zwischen \w (alphanumerischen) und \W (nicht alphanumerischen) Zeichen erfolgen muss. Der Vergleich muss bei Wortbegrenzungen erfolgen, d. h. beim ersten oder letzten Zeichen von Wörtern, die durch beliebige nicht alphanumerische Zeichen voneinander getrennt sind.
\B	Bestimmt, dass der Vergleich nicht bei einer \b-Begrenzung erfolgen darf.

**Tabelle 21.2:** Steuer- bzw. Befehlszeichen für Reguläre Ausdrücke

Angenommen, Sie möchten alle Begriffe finden, die mit der Zeichenfolge »Me.« beginnen und anschließend vier Buchstaben aufweisen, dann wäre, wenn Sie nach der oben stehenden Tabelle logisch vorgingen, die Suchzeichenfolge

Me.\.\w\w\w\w

auf den ersten Blick die richtige Vorgehensweise. Aber die Ergebnisliste entspricht wahrscheinlich mit dem Ergebnis, wie es auch in Abbildung 21.5 zu sehen ist, nicht dem, was Sie im Hinterkopf hatten.



**Abbildung 21.5:** Beim ersten Versuch entspricht die Ergebnisliste manchmal nicht dem, was Sie sich vorgestellt haben

Ihre Vorstellung ist es eher gewesen, dass das Wort nach den vier Buchstaben auch zu Ende ist. Aber auch diese Vorstellung müssen Sie dem Computer mitteilen.

Ergänzen Sie die Suchabfrage um das Steuerzeichen »\s«, und verwenden damit den Suchbegriff

Me.\.\w\w\w\w\s

entspricht das Ergebnis vermutlich schon eher Ihren Vorstellungen.

## Verwendung von Quantifizierern

Nun ist es vermutlich nicht gerade praxisnah, nach Begriffen zu suchen, deren Zeichenanzahl Sie vorher schon kennen. Oder, um beim vorherigen Beispiel zu bleiben: Sie möchten schon eher nach einem Wort suchen, dass mit »Me.« anfängt, dessen Buchstabenzahl Sie aber nicht wissen. Sie möchten also der Such-Engine mitteilen, dass sie im Anschluss an »Me.« nach mindestens einem beliebigen weiteren Zeichen suchen soll, das unter die Kategorie »\w« fällt. Die Lösung zu diesem Problem sind die so genannten Quantifizierer, die Sie in der folgenden Tabelle aufgelistet finden:

Quantifizierer	Funktion
*	Setzt keine oder mehr Übereinstimmungen voraus. Beispiel: Entsprechendes Vorhandensein vorausgesetzt, findet »Me.\w*« sowohl die Zeichenfolge »Me.« als auch »Me.Close«. Dieser Quantifizierer ist gleichbedeutend mit {0,}.
+	Setzt eine oder mehr Übereinstimmungen voraus. Beispiel: »Me.\w+« findet sowohl die Zeichenfolge »Me.Close« als auch »Me.Panel1« aber nicht »Me.« oder »Mehl«.
?	Setzt keine oder eine Übereinstimmung voraus.
{n}	Setzt exakt n Übereinstimmungen voraus. Beispiel: »(\w+\.){2}« findet in dem String Me.components = New System.ComponentModel.Container Me.Splitter2 = New System.Windows.Forms.Splitter die Begriffe »System.ComponentModel.« und »System.Windows.«
{n,}	Setzt mindestens n Übereinstimmungen voraus.
{n,m}	Setzt mindestens n, jedoch höchstens m Übereinstimmungen voraus.
*?	Setzt die erste Übereinstimmung voraus, die so wenige Wiederholungen wie möglich verwendet. Dieser Befehl wird auch »faules *« (lazy *) genannt.
+?	»Faules +«: Setzt so wenige Wiederholungen wie möglich voraus, jedoch mindestens eine.
??	»Faules ?«: Setzt keine Wiederholungen voraus, falls möglich, oder eine Wiederholung.
{n}?{n}	Gleichbedeutend mit {n}.
{n,}?{n,}	Setzt so wenige Wiederholungen wie möglich voraus, jedoch mindestens n Wiederholungen.
{n,m}?{n,m}	Setzt so wenige Wiederholungen wie möglich zwischen n und m voraus.

**Tabelle 21.3:** Mit diesen Quantifizierern steuern Sie Anweisungen für Zeichenwiederholungen

Sie können als Suchbegriff beispielsweise

Me.\w+

eingeben, um zum gewünschten Ziel zu gelangen. Warum? Analysieren wir den Suchbegriff Zeichen für Zeichen. »Me« bestimmt zunächst die ersten beiden Zeichen des Präfixes der gesuchten Begriffe. Den Punkt können wir nicht im Klartext schreiben, da er selbst ein Steuerzeichen darstellt. Damit wird der vorangestellte Backslash nötig, um die Such-Engine den Punkt als bloßes Suchzeichen betrachten zu lassen. Mit »\w« teilen wir der Engine anschließend mit, dass wir nach einem weiteren Buchstabenzeichen suchen. Das Plus schließlich erweitert die Anweisung: Die Engine sucht damit nicht nach einem Buchstabenzeichen sondern nach mindestens einem Buchstabenzeichen. Das Ergebnis lässt nicht lange auf sich warten. Klicken Sie nach Eingabe dieses Suchstrings auf die Schaltfläche *Suchen*, sehen Sie ein Ergebnis, etwa wie in Abbildung 21.6 zu sehen.



**Abbildung 21.6:** Dieser Suchbegriff ist geeignet, mit den Quantifizierern herumzuexperimentieren

Verändern Sie den Suchbegriff beispielsweise in

Me\.\w\*

dann zählt auch »kein Zeichen« als Kriterium für die Begriffserkennung. Das mag zunächst verwirrend sein, denn wie kann »kein Zeichen« als Kriterium gelten?

Wenn Sie alle Zeichenfolgen finden wollen, die mit »Me.« beginnen und weitere Zeichen haben, die aber auch nur aus »Me.« selbst bestehen dürfen, müssen Sie der Such-Engine mitteilen, dass nach dem Suchbegriff Zeichen folgen dürfen, aber nicht müssen. Und das Mitteilen des »nicht müssen« entspricht dem Kriterium »kein Zeichen«.

## Gruppen

Gruppen bilden Sie, wenn Sie einen großen Suchbegriff in mehrere kleine Gruppen unterteilen möchten und die Ergebnisse der kleinen Gruppen auch einzeln abrufen wollen. Sie verwenden zur Gruppenbildung runde Klammern. Dazu ein Beispiel:

Angenommen, Sie möchten den Quelltext eines Programms nach allen Prozeduren durchsuchen lassen, ganz gleich ob es sich um *Properties*, *Subs* oder *Functions* handelt. Sie suchen aber nach bestimmten, nämlich solchen die entweder als *Private*, *Public* oder *Protected* deklariert sind. Und: Sie möchten ohne große Umschweife die Ergebnisse der beiden Gruppen wissen, nämlich um welchen Gültigkeitsbereich es sich handelt und was Sie deklariert haben.

Dazu entwickeln Sie einen Suchbegriff, der die Entscheidungskriterien für den Gültigkeitsbereich in der ersten Gruppe

(Private|Public|Protected)

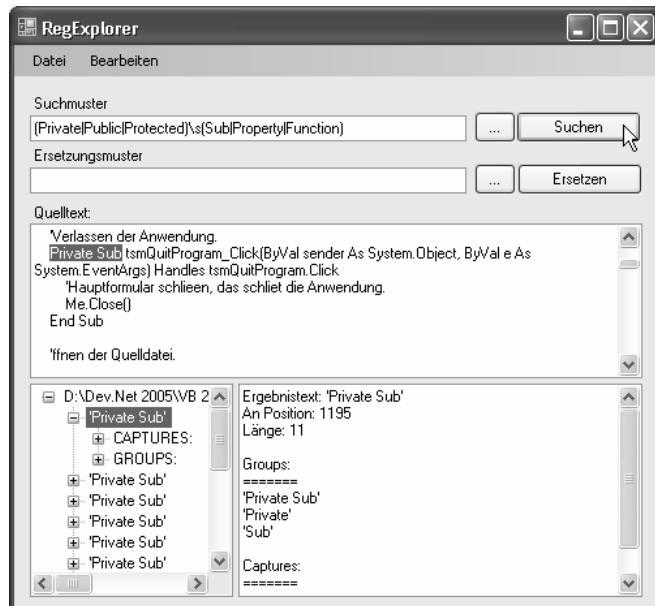
ein Trennzeichen dazwischen und die Prozedurart mit

(Sub|Property|Function)

in der zweiten Gruppe enthält. Fügen Sie diese einzelnen Komponenten zu einem Gesamtsuchbegriff zusammen, etwa mit

(Private|Public|Protected)\s(Sub|Property|Function)

dann erhalten Sie, angewendet auf den Beispiel-Quellcode, ein Ergebnis, etwa wie in Abbildung 21.7 zu sehen.



**Abbildung 21.7:** Durch das Gruppieren von Suchbegriffen können Sie auch programmtechnisch elegant auf Teilergebnisse zugreifen

Das Tolle daran: Sie haben mit Gruppen Zugriff auf – in diesem Beispiel – alle drei Gruppen. Dreit? Genau. Mit einer (der ersten) Gruppe arbeiten Sie grundsätzlich – denn sobald Sie einen Suchbegriff eingeben, erstellen Sie bereits die erste Gruppe. Das gesamte Suchergebnis entspricht deswegen auch immer dem Ergebnis der 1. Gruppe. In diesem Beispiel haben wir explizit zwei weitere Gruppen definiert – die entsprechenden Ergebnisse spiegeln sich in Gruppe 2 und 3 wider, was Sie in der Abbildung auch sehr schön erkennen können.

Gruppen eignen sich nicht nur dazu, beim Programmieren mit Regulären Ausdrücken elegant auf Teilergebnisse zugreifen zu können. Insbesondere beim Ersetzen von Begriffen leisten sie hervorragende Dienste. Durch Gruppen, die Sie im Übrigen auch benennen können, lassen sich Ersetzungen individualisieren. Dazu benötigen Sie allerdings weitere Informationen über Steuerzeichen, die Sie im *Ersetzen*-Ausdruck für Reguläre Ausdrücke verwenden können.

## Suchen und Ersetzen

Diese speziellen Steuerzeichen können Sie ausschließlich beim Ersetzen einsetzen. Die nachstehende Tabelle zeigt, welche Steuerzeichen die Regular-Expression-Engine für das Ersetzen von Ausdrücken versteht:

Zeichen	Beschreibung
\$number	Ersetzt die letzte untergeordnete Zeichenfolge, die der Gruppennummer number (dezimal) entspricht.
\${name}	Ersetzt die letzte untergeordnete Zeichenfolge, die einer (?<name>) -Gruppe entspricht.
\$\$	Ersetzt ein einzelnes "\$"-Literal.
\$&	Ersetzt eine Kopie der gesamten Entsprechung.
\$`	Ersetzt den gesamten Text der Eingabezeichenfolge vor der Entsprechung.
\$'	Ersetzt den gesamten Text der Eingabezeichenfolge nach der Entsprechung.
\$+	Ersetzt die zuletzt erfasste Gruppe.
\$_	Ersetzt die gesamte Eingabezeichenfolge.

**Tabelle 21.4:** Ersetzungs-Steuerzeichen für Reguläre Ausdrücke

Um beim vorhandenen Beispiel zu bleiben: Wenn Sie alle Prozeduren, ganz gleich, wie Sie sie zuvor definiert haben, durch den Gültigkeitsbereichsbezeichner *Private* ersetzen wollen, verwenden Sie als Suchbegriff den bereits bekannten

(Private|Public|Protected)\s(Sub|Property|Function)

und als Ersetzungsbegriff folgenden:

Private \$2

Mit »\$2« greifen Sie auf das Ergebnis der zweiten Gruppe zu – in diesem Beispiel die Prozedurenart, denn sie ist an zweiter Stelle definiert worden. Das Ergebnis der ersten Gruppe interessiert Sie nicht, denn Sie ersetzen es ohnehin durch die Zeichenfolge »Private«.

Das gleiche Beispiel mit benannten Gruppen sähe folgendermaßen aus:

(?<GBereich>Private|Public|Protected)\s(?<Prozedur>Sub|Property|Function)

würden Sie hierbei als Suchstring und

Private \${Prozedur}

als Ersetzungszeichenfolge verwenden.

Nun könnten wir dieses Beispiel noch weiter spinnen und eine Ersetzungsroutine entwickeln, die den vormals vorhandenen Gültigkeitsbereich als Kommentar hinter die Definition setzt. Dazu müssen wir den Suchbegriff um eine weitere Gruppe erweitern, die den Rest der Zeile als insgesamt zu findende Zeichenfolge mit einschließt, etwa folgendermaßen:

Die Suchzeichenfolge:

(?<GBereich> Public|Protected)\s(?<Prozedur>Sub|Property|Function) (?<Rest>[^\\r\\n]\*)

Die Ersetzenzeichenfolge:

```
Private ${Prozedur}${Rest} ' Vormals: ${GBereich}
```

Und das Ergebnis sehen Sie in Abbildung 21.8.

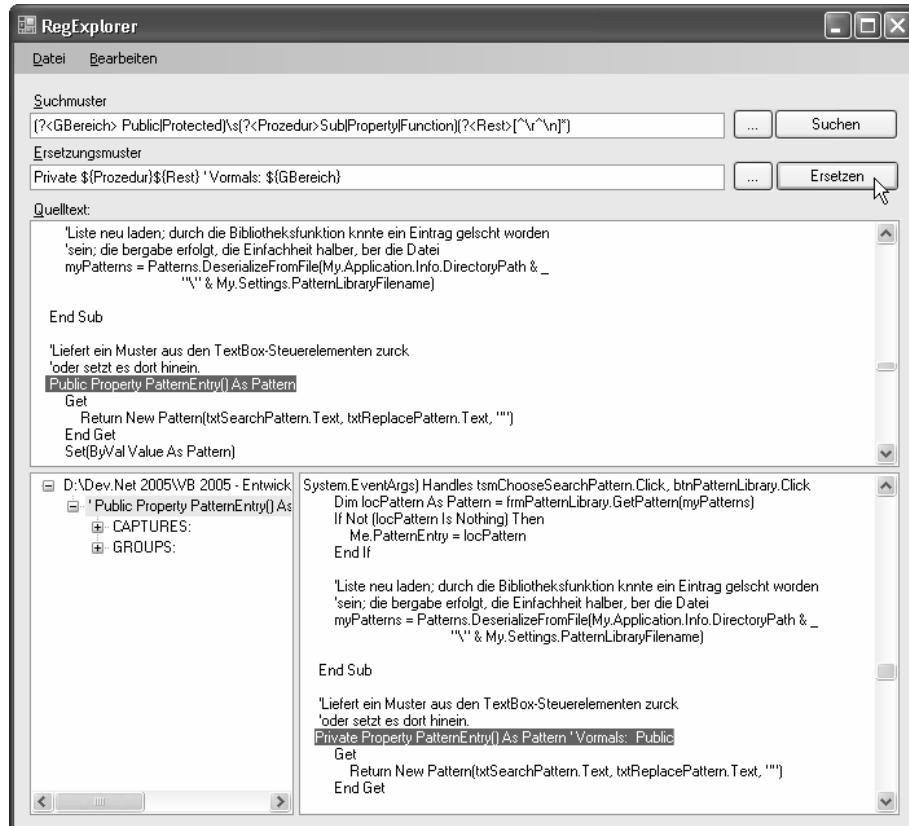


Abbildung 21.8: Das Ersetzen von Text mit Regulären Ausdrücken macht Programmieren fast überflüssig!

## Captures

*Captures* sind dann interessant, wenn Quantifizierer bei Gruppenoperationen ins Spiel kommen. Um auch hier wieder beim Beispiel zu bleiben: Sie möchten wissen, aus welchen Variablen – so vorhanden – die einzelnen Prozedurenaufrufe bestehen. In diesem Fall legen Sie eine Regel an, die die einzelnen Variablen erfassen kann – und zwar so, dass ein Quantifizierer auf den gesamten Ausdruck anwendbar wird. Zum Beispiel:

```
\(((\w|\x20|\.)+[\,|\\])*)+
```

Schauen wir uns den Ausdruck einmal genauer an. Erste Regel: `(\w|\x20|\.)` – er muss mit einer Klammer beginnen. Dann beginnt der eigentliche zu wiederholende Ausdruck:

```
((\w|\x20|\.)+[\,|\\])*)+
```

Dieser besteht wiederum aus zwei Ausdrücken, nämlich

([\w|\x20|\.])+

und

[\\,|\\])

Ausdruck Nummer eins legt alle nach der Klammer vorkommenden Zeichen so fest, dass sie aus Buchstaben, Leerzeichen oder dem Punkt bestehen dürfen. Das anschließende Quantifizierungszeichen »+« definiert, dass diese Zeichen sich beliebig wiederholen dürfen, aber mindestens einmal vorhanden sein müssen.

Der zweite Ausdruck regelt das Ende eines Parameterblocks, der mit einem Komma oder einer schließenden Klammer enden kann. Beide Ausdrücke zusammengefügt ergeben die komplette Parameterregel innerhalb der Klammer. Und jetzt kommt der Trick: Da wir nicht wissen, wie viele Parameter innerhalb eines Prozedurenprototypen definiert sind, setzen wir den »+-Quantifizierer ans Ende, und schon kann uns egal sein, wie viele Parameter folgen – der Quantifizierer sorgt dafür, dass entsprechend viele gefunden werden.

Anschließend schnappen wir uns den Suchstring aus dem vorherigen Beispiel und modifizieren ihn so, dass eine weitere Gruppe für den Funktionsnamen gefunden werden kann. Das ist vergleichsweise einfach: Lediglich der Ausdruck

\s([\w+)

muss noch eingefügt werden. Das »\s« regelt die Trennung zwischen Prozedurentypnamen und Funktionsnamen; die Gruppe »([\w+)« deckt den Funktionsnamen ab. Packen wir alles zusammen, erhalten wir folgenden Gesamtsuchstring:

(?<GBereich>Private|Public|Protected)\s(?<Prozedur>Sub|Property|Function)\s([\w+]\((([\w|\x20|\.])+[\\,|\\])\)+

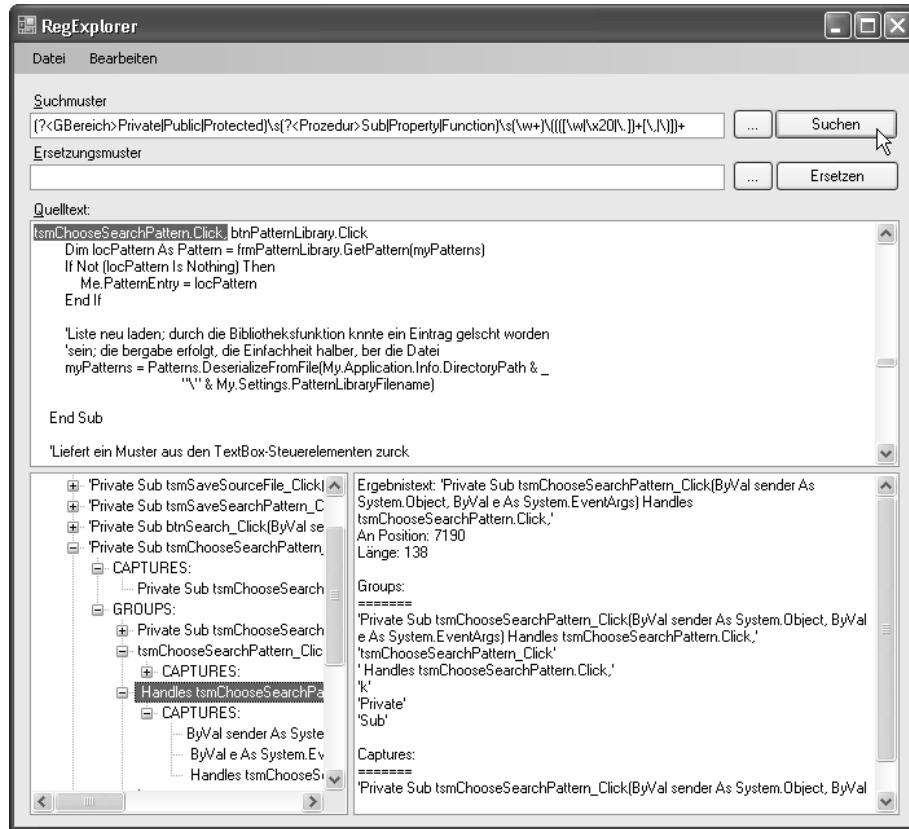
Eindrucksvoll, oder nicht? Ganz ehrlich: So einfach, wie ich diese Beschreibung hier herunter geschrieben habe, war das Austüfteln dieses Strings nicht – es hat mich immerhin drei Stunden gekostet. Nichtsdestotrotz hat sich der Aufwand gelohnt, denn: Spätestens an dieser Stelle werden Sie den Nutzen der *Captures* kennen lernen.

Um die *Captures* der einzelnen Gruppen auch sichtbar zu machen, wählen Sie aus dem Menü *Datei* die Option *GroupCaptures anzeigen*.

Wenn Sie diesen String auf die zuvor geladene Visual Basic-Datei anwenden (keine Angst, Sie können diese Mammutkonstruktion aus der Bibliothek holen), können Sie sich die *Captures* der einzelnen Gruppen ebenfalls in der Ergebnisliste betrachten – etwa wie in Abbildung 21.9 zu sehen.

Da wir die Parameterfindung so allgemeingültig gehalten haben, dass wir einen Quantifizierer einsetzen konnten, kommen wir über die Captures an jede einzelne Zeichenfolge, die durch die Quantifizierer in Kombination mit dem eigentlichen Gruppensuchstring gefunden wurde. Und das sind bei der 3. Gruppe eben die einzelnen Parameter.

Sie sehen, dass wir nur durch die Anwendung von Regulären Ausdrücken schon auf dem besten Wege sind, einen kompletten Cross-Referenzer zu kreieren – und bislang haben wir noch nicht eine einzige Zeile Code dafür schreiben müssen!



**Abbildung 21.9:** Richtig angewendet entlocken Sie der *Regular Expression Engine* mit *Captures* die durch die Quantifizierer entstandenen Suchergebnisse

Doch das wird sich gleich ändern. Im ► Abschnitt »Programmieren von Regulären Ausdrücken« ab Seite 630 erfahren Sie mehr darüber, wie Sie Reguläre Ausdrücke in Ihren eigenen Programmen einsetzen können.

Um die Tabellen zum Nachschlagen komplett zu halten, habe ich zuvor aber noch zwei für Sie.

## Optionen bei der Suche

Optionen für die Suche können Sie später, bei der Programmierung von Regulären Ausdrücken, bequem über die Klassenparameter einstellen. Allerdings können Sie diese Parameter auch mit Steuerzeichen direkt in den Suchstring einbauen. Die folgende Tabelle gibt Ihnen Auskunft darüber, mit welchem Steuerzeichen Sie welchen Parameter aktivieren bzw. deaktivieren können.

RegexOption-Member	Inline-Zeichen	Beschreibung
None	Nicht vorhanden	Gibt an, dass keine Optionen festgelegt wurden.
IgnoreCase	i	Gibt an, dass bei Übereinstimmungen die Groß-/Kleinschreibung berücksichtigt werden soll.
Multiline	m	Bestimmt den Mehrzeilenmodus. Das ändert die Bedeutung von ^ und \$, sodass sie jeweils dem Anfang und dem Ende einer beliebigen Zeile innerhalb des zu durchsuchenden Strings und nicht nur dem Anfang und dem Ende der gesamten Zeichenfolge entsprechen.
ExplicitCapture	n	Gibt an, dass die einzigen gültigen Aufzeichnungen ausdrücklich benannte oder nummerierte Gruppen in der Form (?<name>...) sind. Dadurch können Klammern als nicht aufzeichnende Gruppen eingesetzt werden, ohne dass die umständliche Syntax des Ausdrucks (?....) benötigt wird.
Compiled	c	Gibt an, dass der Reguläre Ausdruck in eine Assembly kompiliert wird. Generiert MSIL (Microsoft Intermediate Language)-Code für den Regulären Ausdruck und ermöglicht eine schnellere Ausführung, jedoch auf Kosten der kurzen Startdauer.
Singleline	s	Gibt den Einzeilenmodus an. Ändert die Bedeutung des Punktes (.), sodass dieser jedem Zeichen entspricht (und nicht jedem Zeichen mit Ausnahme von \n).
IgnorePatternWhitespace	x	Gibt an, dass Leerraum ohne Escape-Zeichen aus dem Muster ausgeschlossen wird und ermöglicht Kommentare hinter einem Nummernzeichen (#). Beachten Sie, dass niemals Leerraum aus einer Zeichenklasse eliminiert wird.
RightToLeft	r	Gibt an, dass die Suche von rechts nach links und nicht, wie standardmäßig, von links nach rechts durchgeführt wird. Ein Regulärer Ausdruck mit dieser Option steht links von der Anfangsposition und nicht rechts davon. (Daher sollte die Anfangsposition als das Ende der Zeichenfolge angegeben werden.) Diese Option kann nicht mitten in der Suchmusterzeichenfolge angegeben werden, um zu verhindern, dass Reguläre Ausdrücke mit Endlosschleifen auftreten. Die (?<)-Lookbehind-Konstrukte bieten jedoch eine ähnliche Funktionalität, die als Teilausdruck verwendet werden können. <i>RightToLeft</i> ändert lediglich die Suchrichtung. Die gesuchte untergeordnete Zeichenfolge an sich wird nicht umgekehrt.
ECMAScript	Nicht vorhanden	Gibt an, dass für den Ausdruck so genanntes ECMAScript-konformes Verhalten aktiviert ist (bestimmtes, standardisiertes <i>RegEx</i> -Verhalten). Diese Option kann nur in Verbindung mit dem <i>IgnoreCase</i> - und dem <i>Multiline</i> -Flag verwendet werden. Bei Verwendung dieser Option mit anderen Flags wird eine Ausnahme ausgelöst.
CultureInvariant	Nicht vorhanden	Gibt an, dass kulturelle Unterschiede bei der Sprache ignoriert werden.

**Tabelle 21.5:** Options-Steuerzeichen für Reguläre Ausdrücke

## Steuerzeichen zu Gruppendefinitionen

Auch bei der Definition von Gruppen gibt es weitere Kombinationsmöglichkeiten. Die folgende Tabelle verrät Ihnen, welche es gibt:

Gruppenkonstruktur	Beschreibung
( )	Zeichnet die übereinstimmende Teilzeichenfolge auf (oder die nicht aufzeichnende Gruppe). Aufzeichnungen mit () werden gemäß der Reihenfolge der öffnenden Klammern automatisch nummeriert, beginnend mit 1. Die erste Aufzeichnung, Aufzeichnungselement Nummer 0, ist der Text, dem das gesamte Muster für den Regulären Ausdruck entspricht. Auch wenn Sie also keine Gruppe mit einer Klammer gebildet haben, gibt es immer mindestens Gruppe 1.
(?<name> )	Zeichnet die übereinstimmende Teilzeichenfolge in einem Gruppennamen oder einem Nummernnamen auf. Die Zeichenfolge für »name« darf keine Satzzeichen enthalten und nicht mit einer Zahl beginnen. Sie können anstelle von spitzen Klammern einfache Anführungszeichen verwenden. Beispiel: »(?'name')«.
(?<name1-name2> )	Ausgleichsgruppdefinition. Löscht die Definition der zuvor definierten Gruppe »name2« und speichert in Gruppe »name1« das Intervall zwischen der zuvor definierten Gruppe »name2« und der aktuellen Gruppe. Wenn keine Gruppe »name2« definiert ist, wird die Übereinstimmung rückwärts verarbeitet. Da durch Löschen der letzten Definition von »name2« die vorherige Definition von »name2« angezeigt wird, kann mithilfe dieses Konstrukt der Aufzeichnungstapel für die Gruppe »name2« als Zähler für die Aufzeichnung von geschachtelten Konstrukten, z. B. Klammern, verwendet werden. In diesem Konstrukt ist »name1« optional. Sie können anstelle von spitzen Klammern einfache Anführungszeichen verwenden. Beispiel: »(?'name1-name2')«.
(?: )	Nicht aufzeichnende Gruppe.
(?imnsx-imnsx: )	Aktiviert oder deaktiviert die angegebenen Optionen innerhalb des Teilausdrucks. Beispielsweise aktiviert »(?-s:)« die Einstellung, dass Groß-/Kleinschreibung nicht beachtet wird, und deaktiviert den Einzeilenmodus. Weitere Informationen hierzu finden Sie unter Optionen für Reguläre Ausdrücke.
(?= )	Positive Lookahead-Anweisung mit einer Breite von Null. Der Vergleich wird nur dann fortgesetzt, wenn der Teilausdruck rechts von dieser Position übereinstimmt. Beispiel: »\w+(?=\\d)« entspricht einem Wort, gefolgt von einer Ziffer, wobei für die Ziffer keine Übereinstimmung gesucht wird. Dieses Konstrukt wird nicht rückwärts verarbeitet.
(?! )	Negative Lookahead-Anweisung mit einer Breite von Null. Der Vergleich wird nur dann fortgesetzt, wenn der Teilausdruck rechts von dieser Position nicht übereinstimmt. Beispiel: »\b(?!un)\\w+\\b« entspricht Wörtern, die nicht mit »un« beginnen.
(?<= )	Positive Lookbehind-Anweisung mit einer Breite von Null. Der Vergleich wird nur dann fortgesetzt, wenn der Teilausdruck links von dieser Position übereinstimmt. Beispiel: »(?<=19)99« entspricht Instanzen von 99, die auf 19 folgen. Dieses Konstrukt wird nicht rückwärts verarbeitet.
(?<! )	Negative Lookbehind-Anweisung mit einer Breite von Null. Der Vergleich wird nur dann fortgesetzt, wenn der Teilausdruck links von dieser Position nicht übereinstimmt.
(?> )	Nicht zurückverfolgende Teilausdrücke (so genannte »gierige« Teilausdrücke). Für den Teilausdruck wird einmal eine volle Übereinstimmung gesucht, dann wird der Teilausdruck nicht stückweise in der Rückwärtsverarbeitung einbezogen. (D. h. der Teilausdruck entspricht nur Zeichenfolgen, für die durch den Teilausdruck allein eine Übereinstimmung gesucht werden würde.)

**Tabelle 21.6:** Spezielle Gruppensteuerzeichen bei Reguläre Ausdrücken

# Programmieren von Regulären Ausdrücken

Alle Grundlagen zu Regulären Ausdrücken sind jetzt an vielen Beispielen geklärt, und damit liegt die aufwändigste Lernarbeit bereits hinter Ihnen. Sich das Programmieren selbst anzueignen ist jetzt nur noch ein Klacks, und die Beispiele, die nun folgen, sind direkt aus dem Programm entnommen, mit dem Sie die ganze Zeit gearbeitet haben.

Die Klasse Regex bildet den Schlüssel zu den Funktionen von Regulären Ausdrücken. Sie können Sie auf zwei verschiedene Weisen verwenden: Entweder Sie instanzieren sie und nutzen ihre Member-Funktionen. Oder Sie nutzen ausschließlich ihre statischen Funktionen.

---

**HINWEIS:** Aufgepasst jedoch, wenn Sie die statischen Funktionen der Regex-Klasse verwenden. Fehler, die zum Beispiel in Ausdrücken vorkommen, führen nicht zu Ausnahmen (*Exceptions*)! Ich habe keine Ahnung, was sich die Entwickler der Klassen dabei gedacht haben, aber ob Sie es glauben oder nicht: Wenn Sie die statischen Funktionen verwenden und Fehler dabei auftreten, zeigen die entsprechenden Funktionen lediglich eine MessageBox (!!) – Ihr Programm bekommt davon aber nichts mit, es wird nicht durch eine Ausnahme unterbrochen und läuft weiter, als wäre nichts passiert. Deswegen gilt der Grundsatz: Instanzieren Sie grundsätzlich die Regex-Klasse, und vermeiden Sie die Nutzung der statischen Funktionen, wo Sie können, denn Sie können sonst auf mögliche Fehler keinen Einfluss nehmen!

---

Zu Demonstrationszwecken (die statischen Funktionen wollen ja dennoch gezeigt werden) habe ich mich an diesen Grundsatz in den folgenden Beispielen übrigens ein paar Mal selbst nicht gehalten.

## Ergebnisse im Match-Objekt

Beginnen wir direkt mit einer statischen Funktion: Sie möchten nach einem Regulären Ausdruck in einem String suchen. Nichts leichter als das, Sie schreiben einfach:

```
Dim match As Match = Regex.Match("Dieser wird durchsucht", "hiernach")
```

Das Match-Objekt selber wird durch die Match-Funktion zurückgeliefert. Möchten Sie auf die nicht statischen Member-Funktionen der Regex-Klasse zugreifen, müssen Sie das Regex-Objekt zunächst – wie jede andere Klasse auch – instanzieren. Das gleiche Ergebnis würden Sie folgendermaßen erzielen:

```
Dim RegexInstanz As New Regex("Hiernach")
Dim match As Match = RegexInstanz.Match("Dieser wird durchsucht")
```

Da das Suchmuster bereits bei der Klasseninstanzierung bestimmt wird, geben Sie es im Unterschied zur Verwendung der statischen Version nicht mehr als Parameter an, wenn Sie nach Übereinstimmungen mit der Match-Methode suchen. Nun liegt es in der Natur von Regulären Ausdrücken, dass ein String höchstens ausreichen würde, den ersten Treffer im Text widerzuspiegeln. Sie benötigen allerdings mehr Informationen, um wirklich Brauchbares mit dem Treffer – oder vielmehr: den Treffern – anstellen zu können. Deswegen hält das Match-Objekt einige Eigenschaften parat, die diese Informationen liefern:

Eigenschaft des Match-Objektes	Funktion
NextMatch	Liefert ein neues Match-Objekt zurück, das Informationen über den nächsten Treffer enthält.
Value	Gibt den String zurück, der den Treffer darstellt.
Index	Gibt die Position innerhalb des Suchstrings an, an dem der Treffer aufgetreten ist. Die Positionszählung beginnt dabei, wie bei allen Strings, an der Position 0.
Length	Gibt an, aus wie vielen Zeichen der Treffer-String besteht.
Success	Ermittelt, ob der Treffer erfolgreich war.
Groups	Liefert eine Collection aus Group-Objekten zurück, die die Teilergebnisse der einzelnen Gruppen enthalten.
Captures	Liefert eine Collection aus Capture-Objekten zurück, die Captures enthalten, etwa wie die in den vorangegangenen Suchbegriff-Beispielen beschriebenen.

**Tabelle 21.7:** Die wichtigsten Eigenschaften des *Match*-Objektes

## Die Matches-Auflistung

Nun wäre es nicht des schönen Programmierstils würdig, den .NET ermöglicht, müsste man den jeweils nächsten Treffer einer Regex.Match-Abfrage in einer Do/Loop-Schleife ermitteln, bis NextMatch schließlich Nothing zurückliefert, weil es keine weiteren Treffer mehr gibt. Aus diesem Grund bietet die Regex-Klasse die so genannte Matches-Collection an, die alle Treffer als Array von Match-Objekten erhält, und durch die sich vor allen Dingen elegant mit For/Each iterieren lässt.

Der Programmcode bis zur äußeren Schleife, die anschließend die TreeView im RegExplorer aufbaut, sieht aus diesem Grund folgendermaßen aus:

```
Private Sub btnSearch_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
Handles btnSearch.Click

    Dim locRegEx As Regex

    Try
        locRegEx = New Regex(txtSearchPattern.Text)
    Catch ex As Exception
        MessageBox.Show("Fehler beim Anlegen des RegEx-Objektes!" + ex.Message, _
            "Fehler in Ausdruck:", MessageBoxButtons.OK, MessageBoxIcon.Exclamation)
    End Try

    Dim locRootNode As TreeNode

    'Die Ergebnisstruktur in der TreeView anzeigen.
    tvwResults.Nodes.Clear()
    With tvwResults.Nodes
```

```

'Dateiname ist Wurzelknoten der TreeView.
locRootNode = .Add(myFilename)
'Alle Match-Objekte (Treffer) durchlaufen...
For Each locMatch As Match In locRegEx.Matches(txtSourceText.Text).

.

.

Next

```

Wenn der Anwender auf die *Suchen*-Schaltfläche klickt, verzweigt das Programm in Sub `btnSearch_Click`. Die Variable `locRegEx` wird anschließend als Typ `Regex` definiert. Da wir die `Regex`-Klasse instanzieren, ergibt es Sinn, die Instanzierungsanweisung in einen Try/Catch-Block zu packen, da genau hier der Zeitpunkt in Frage kommt, zu dem die *Regular Expression Engine* eine Ausnahme auslösen kann, die ein anwenderfreundliches Programm natürlich abfangen muss. Die eigentliche Auswertung erfolgt mit dem Abruf von `locRegEx.Matches(txtSourceText.Text)` in der Schleifendefinition. Die Textbox `txtSourceText` enthält den Quelltext; den Suchbegriff, der aus der Textbox `txtSearchPattern` stammt, haben wir beim Instanzieren des `Regex`-Objektes Zeilen zuvor schon definiert. Die `Matches`-Eigenschaft liefert nun alle Treffer als `Match`-Auflistung zurück, durch die mit `For/Each` wunderbar iteriert werden kann. `locMatch` enthält dabei wieder alle Informationen über einen Treffer.

## Abrufen von Captures und Gruppen eines Match-Objektes

Das Abrufen von vorhandenen *Captures* und Gruppen eines `Match`-Objektes gestaltet sich ebenfalls recht simpel: Für *Captures* verwenden Sie die `Captures`-Eigenschaft, die eine `Captures`-Auflistung zurückliefert, die wiederum aus einzelnen `Capture`-Objekten aufgebaut ist.

Für Gruppen verwenden Sie die `Groups`-Eigenschaft, die eine `Groups`-Auflistung aus `Group`-Objekten zurückliefert.

Das Iterieren durch diese Elemente ist ebenfalls recht einfach, wie der erste Teil der inneren Schleife des Programmteils zum Aufbau der `TreeView` zeigt:

```

'Alle Match-Objekte (Treffer) durchlaufen...
For Each locMatch As Match In locRegEx.Matches(txtSourceText.Text)

    'und in der TreeView darstellen.
    Dim locMainNode As New TreeNode(""" + locMatch.Value + """)
    locMainNode.Tag = locMatch
    locRootNode.Nodes.Add(locMainNode)

    'Falls es zu einem Match Captures gab...
    If locMatch.Captures.Count > 0 Then
        Dim locCaptureNode As TreeNode = locMainNode.Nodes.Add("CAPTURES:")
        For Each locCC As Capture In locMatch.Captures
            '...auch diese unter jedem Match darstellen.
            Dim locNode As TreeNode = locCaptureNode.Nodes.Add(locCC.Value)
            locNode.Tag = locCC
        Next
    End If

```

```

'Das Gleiche gilt für Groups.
If locMatch.Groups.Count > 0 Then
    Dim locGroupNode As TreeNode = locMainNode.Nodes.Add("GROUPS:")
    For Each locGroup As Group In locMatch.Groups
        Dim locNode As TreeNode = locGroupNode.Nodes.Add(locGroup.Value)
        locNode.Tag = locGroup

    'Captures der einzelnen Gruppen nur im Bedarfsfall zeigen
    If myGroupCaptures Then
        If locGroup.Captures.Count > 0 Then
            Dim locCaptureNode As TreeNode = locNode.Nodes.Add("CAPTURES:")
            For Each locCC As Capture In locGroup.Captures
                Dim locGCNode As TreeNode = locCaptureNode.Nodes.Add(locCC.Value)
                locGCNode.Tag = locCC
            Next
        End If
    End If
    Next
End If
Next

```

Damit die *Captures* der einzelnen Gruppen nur im Bedarfsfall angezeigt werden, gibt es ein Flag, das die Anzeige steuert. Dieses Flag wird gesetzt in Abhängigkeit der Einstellung, die der Anwender des Programms mit der Option *Group Captures anzeigen* im Menü *Datei* festlegt:

```

'An- und abschalten der Anzeige der Group Captures:
Private Sub tsmShowGroupCaptures_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles tsmShowGroupCaptures.Click
    'Xor vertauscht Bits.
    myGroupCaptures = myGroupCaptures Xor True
    'Durch ein Häkchen widerspiegeln lassen:
    tsmShowGroupCaptures.Checked = myGroupCaptures
End Sub

```

Um die *Group Captures* letzten Endes darzustellen, wenn der Anwender diese Option gewählt hat, verfährt das Programm so wie im vorherigen Listing (fett markiert) zu sehen.

Sowohl die *Group-* als auch die *Capture*-Objekte, die dabei verwendet werden, sind dem *Match*-Objekt in ihrer Anwendung sehr ähnlich. Das *Capture*-Objekt verfügt über die folgenden wichtige Eigenschaften:

Wichtige Eigenschaften des Capture-Objektes	Funktion
Value	Gibt den String zurück, der das Capture darstellt.
Index	Gibt die Position innerhalb des Suchstrings an, an dem das Capture aufgetreten ist. Die Positionszählung beginnt dabei, wie bei allen Strings, an der Position 0.
Length	Gibt an, aus wie vielen Zeichen der Capture-String besteht.

**Tabelle 21.8:** Die wichtigsten Eigenschaften des Capture-Objektes

Noch ähnlicher zum Match-Objekt ist das Group-Objekt, das quasi die gleiche Funktionalität wie das Match-Objekt aufweist, nur dass es – logischerweise – keine Groups-Eigenschaft besitzt und auch die NextMatch-Eigenschaft vermissen lässt.

Eigenschaft des Group-Objektes	Funktion
Value	Gibt den String zurück, der den Treffer der Gruppe darstellt.
Index	Gibt die Position innerhalb des Suchstrings an, an dem der Treffer der Gruppe aufgetreten ist. Die Positionszählung beginnt dabei, wie bei allen Strings, an der Position 0.
Length	Gibt an, aus wie vielen Zeichen der Gruppentreffer-String besteht.
Success	Ermittelt, ob der Gruppentreffer erfolgreich war.
Captures	Liefert eine <i>Collection</i> aus <i>Capture</i> -Objekten zurück, die <i>Captures</i> enthalten, etwa wie die in den vorangegangenen Suchbegriff-Beispielen beschriebenen.

**Tabelle 21.9:** Die wichtigsten Eigenschaften der Match-Klasse

Mit diesen Infos sind Sie für die Programmierung von Regulären Ausdrücken bestens gerüstet. Sie sehen selbst, dass sich die eigentliche Problemlösung mit Regulären Ausdrücken nicht in der Anwendung der Klassen verbirgt – das ist der weitaus weniger aufwändige Teil. Das eigentliche Problem – oder besser: die eigentliche Übung, die Sie benötigen – liegt in der Anwendung der Regulären Ausdrücke selbst, also: Wie müssen Sie Ihre Such- bzw. Ersetzungsstrings gestalten, damit Sie die nötigen Informationen innerhalb einer Zeichenkette finden, um beispielsweise Benutzereingaben zu strukturieren und entsprechend auszuwerten?

Ein gutes Beispiel für den Einsatz von Regulären Ausdrücken zeigt das folgende Beispiel.

## Regex am Beispiel: Berechnen beliebiger mathematischer Ausdrücke

Es gibt hunderte von denkbaren Anwendungen, bei denen Sie mathematische Ausdrücke innerhalb eines Programms auswerten müssen. Denken Sie beispielsweise an Aufmaßprogramme, die die Bausubstanz eines Hauses berechnen. Oder einen Funktionsplotter, der die Graphen beliebiger Funktionen darstellen kann. Überhaupt kann es immer nur von Vorteil sein, wenn Sie dem Anwender an geeigneten Stellen die Möglichkeit geben, eine beliebige Formel zu berechnen. Er braucht dann nämlich nicht für jede Kleinigkeit seinen Taschenrechner zu bemühen.

Leider ist das Parsen – also das Analysieren und Berechnen – eines mathematischen Ausdrucks keine wirklich triviale Sache, denn es gibt einige Dinge zu berücksichtigen:

Da sind zunächst mal Klammern, die die Priorität der Berechnungsreihenfolge ändern. Der Ausdruck

$2*2+2$

ergibt natürlich was völlig anderes als

$2*(2+2)$

Im ersten Fall lautet das Ergebnis 6, im zweiten 8. Noch komplizierter wird es, wenn man die Hierarchie der Operatoren berücksichtigt. Sie können eine Formel nicht einfach von links nach rechts auseinander nehmen, sondern müssen die Operatorprioritäten berücksichtigen:

Der Ausdruck

$2+2*2^2$

ist dafür ein gutes Beispiel. Hier werden erst die Potenz, anschließend das Produkt und schließlich die Summe berechnet, nicht umgekehrt.

Ebenfalls nicht trivial sind Funktionen, am besten mit unterschiedlich langen Funktionsnamen und beliebig vielen Parametern. Denken Sie beispielsweise an einen Ausdruck wie

$2+2*(2+\text{Max}(123;234;345;456))$

In dieser Formel müssen nicht nur der Funktionsname, sondern auch die Anzahl der Parameter bestimmt werden, die innerhalb des Funktionsnamens auftreten.

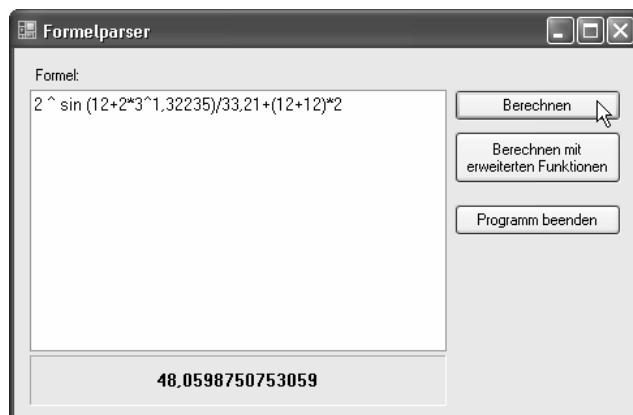
Und dann gibt es zu guter Letzt auch noch das Problem der negativen Vorzeichen. Der Parser muss so clever sein, dass er den Ausdruck

$-2*-1^{(-2-1)}$

in den Ausdruck

$((0-1)*2)*((0-1)*1)^(((0-1)*2)-((0-1)*1))$

umwandeln kann, wenn er nicht eine komplizierte Funktionalität zum Erkennen von negativen Zahlen bereitstellen will.



**Abbildung 21.10:** Mit dem Formelparser können Sie in Ihren eigenen Programmen beliebige Formeln errechnen

---

**BEGLEITDATEIEN:** Im Verzeichnis `.\VB 2005 - Entwicklerbuch\E - Datentypen\Kap21\FormelParser\` finden Sie das Projekt `FormelParser`.

---

Dieses Programm enthält die Klasse `ADFormularParser`, die in der Lage ist, genau diese Auswertungen durchzuführen. Das einzige Formular des Projektes ist nur das Drumherum, damit Sie die Klasse ohne umständliche Kommandozeilenparameter austesten können.

## Der Formelparser

Wenn Sie dieses Programm starten, sehen Sie einen Dialog, wie er auch in Abbildung 21.10 zu sehen ist. Eine Testformel ist bereits vorgegeben. Sie können unter *Formel* einen beliebigen Ausdruck eingeben und durch Mausklick auf *Berechnen* berechnen lassen. Das Ergebnis zeigt das Programm anschließend im Beschriftungsfeld unterhalb des Eingabefensters an.

Der Programmcode selber dazu ist denkbar gering und lautet wie folgt:

```
Private Sub btnCalculate_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
Handles btnCalculate.Click

    Dim locFormPars As New ADFormularParser(txtFormular.Text)
    lblResult.Text = locFormPars.Result.ToString

End Sub
```

Sobald der Anwender die Schaltfläche betätigt, instanziert das Programm die Klasse `ADFormularParser` in `locFormPars`. Der eigentliche Ausdruck liegt dabei als String aus dem Textfeld `txtFormular` vor und wird dem Konstruktor dieser Klasse übergeben. Für das Berechnen dieses Ausdrucks ist nur ein einziger Methodenaufruf erforderlich: `Result`. Sie liefert das Ergebnis als `Double` zurück; damit das Ergebnis im Label angezeigt werden kann, wird es mit `Tostring` in eine Zeichenkette umgewandelt.

## Die Klasse `ADFormularParser`

Interessant zu sehen ist es, wie die Klasse an sich arbeitet. Obwohl sie einen recht großen Funktionsumfang besitzt, ist die eigentliche Auswertungslogik durch die konsequente Anwendung von Regulären Ausdrücken recht klein gehalten.

Bevor Sie sich das dokumentierte Listing dieser Klasse anschauen, vielleicht noch ein paar Worte zur generellen Funktionsweise:

Bei der Entwicklung dieser Klasse stand ihre beliebige Erweiterbarkeit im Vordergrund. Das heißt im Klartext: Ein Entwickler, der diese Klasse benutzt, sollte durch Vererbung die Möglichkeit haben, auf einfache Weise den Funktionsvorrat zu erweitern. Diese Möglichkeit sollte aber nicht nur für normale Funktionen, sondern auch für Operatoren gelten. Aus diesem Grund besteht die Klasse eigentlich aus zwei wichtigen Klassen. Die erste Klasse speichert Operatoren in einem bestimmten Format; und die zweite Klasse ist schließlich für die eigentliche Funktionsauswertung verantwortlich.

Die Klasse zur Speicherung der Operatoren und Funktionen finden Sie im Folgenden abgedruckt. Sie befindet sich im Projekt in der Codedatei `ADFunction.vb`.

```
''' <summary>
''' Speichert die Parameter für eine Funktion, die vom FormelParser berücksichtigt werden kann.
''' </summary>
''' <remarks></remarks>
Public Class ADFunction
    Implements IComparable

    Public Delegate Function ADFunctionDelegate(ByVal parArray As Double()) As Double
```

```

Protected myFunctionname As String
Protected myParameters As Integer
Protected myFunctionProc As ADFunctionDelegate
Protected myConsts As ArrayList
Protected myIsOperator As Boolean
Protected myPriority As Byte

''' <summary>
''' Erstellt eine neue Instanz dieser Klasse.
''' Verwenden Sie diese Überladungsversion, um Operatoren zu erstellen, die aus einem Zeichen bestehen,
''' </summary>
''' <param name="OperatorChar">Das Zeichen, das den Operator darstellt.</param>
''' <param name="FunctionProc">Der ADFunctionDelegate für die Berechnung durch diesen Operator.</param>
''' <param name="Priority">Die Operatorpriorität (3= Potenz, 2=Punkt, 1=Strich).</param>
''' <remarks></remarks>
Sub New(ByVal OperatorChar As Char, ByVal FunctionProc As ADFunctionDelegate, ByVal Priority As Byte)

    If Priority < 1 Then
        Dim Up As New ArgumentException("Priority kann für Operatoren nicht kleiner 1 sein.")
        Throw Up
    End If

    myFunctionname = OperatorChar.ToString
    myParameters = 2
    myFunctionProc = FunctionProc
    myIsOperator = True
    myPriority = Priority
End Sub

''' <summary>
''' Erstellt eine neue Instanz dieser Klasse. Verwenden Sie
''' diese Überladungsversion, um Funktionen zu erstellen, die aus mehreren Zeichen bestehen.
''' </summary>
''' <param name="FunctionName">Die Zeichenfolge, die den Funktionsnamen darstellt.</param>
''' <param name="FunctionProc">Der ADFunctionDelegate für die Berechnung durch diese Funktion.</param>
''' <param name="Parameters">Die Anzahl der Parameter, die diese Funktion entgegen nimmt.</param>
''' <remarks></remarks>
Sub New(ByVal FunctionName As String, ByVal FunctionProc As ADFunctionDelegate, _
        ByVal Parameters As Integer)
    myFunctionname = FunctionName
    myFunctionProc = FunctionProc
    myParameters = Parameters
    myIsOperator = False
    myPriority = 0
End Sub

''' <summary>
''' Liefert den Funktionsnamen bzw. das Operatorenzeichen zurück.
''' </summary>
''' <value></value>
''' <returns></returns>
''' <remarks></remarks>

```

```

Public ReadOnly Property FunctionName() As String
    Get
        Return myFunctionname
    End Get
End Property

''' <summary>
''' Liefert die Anzahl der zur Anwendung kommenden Parameter für diese Funktion zurück.
''' </summary>
''' <value></value>
''' <returns></returns>
''' <remarks></remarks>
Public ReadOnly Property Parameters() As Integer
    Get
        Return myParameters
    End Get
End Property

''' <summary>
''' Zeigt an, ob es sich bei dieser Instanz um einen Operator handelt.
''' </summary>
''' <value></value>
''' <returns></returns>
''' <remarks></remarks>
Public ReadOnly Property IsOperator() As Boolean
    Get
        Return myIsOperator
    End Get
End Property

''' <summary>
''' Ermittelt die Priorität, die dieser Operator hat. (3=Potenz, 2=Punkt, 1=Strich)
''' </summary>
''' <value></value>
''' <returns></returns>
''' <remarks></remarks>
Public ReadOnly Property Priority() As Byte
    Get
        Return myPriority
    End Get
End Property

''' <summary>
''' Ermittelt den Delegaten, der diese Funktion oder diesen Operator berechnet.
''' </summary>
''' <value></value>
''' <returns></returns>
''' <remarks></remarks>
Public ReadOnly Property FunctionProc() As ADFunctionDelegate
    Get
        Return myFunctionProc
    End Get
End Property

```

```

'<summary>
'<summary> Ruft den Delegaten auf, der diese Funktion (diesen Operator) berechnet.
'</summary>
'<param name="parArray">Das Array, dass die Argumente der Funktion enthält.</param>
'<returns></returns>
'<remarks></remarks>
Public Function Operate(ByVal parArray As Double()) As Double
    If Parameters > -1 Then
        If parArray.Length <> Parameters Then
            Dim Up As New ArgumentException _
                ("Anzahl Parameter entspricht nicht der Vorschrift der Funktion " & FunctionName)
            Throw Up
        End If
    End If
    Return myFunctionProc(parArray)
End Function

'<summary>
'<summary> Vergleicht zwei Instanzen dieser Klasse anhand ihres Prioritätswertes.
'</summary>
'<param name="obj">Eine ADFunction-Instanz, die mit dieser Instanz verglichen werden soll.</param>
'<returns></returns>
'<remarks></remarks>
Public Function CompareTo(ByVal obj As Object) As Integer Implements System.IComparable.CompareTo
    If obj.GetType Is GetType(ADFunction) Then
        Return myPriority.CompareTo(DirectCast(obj, ADFunction).Priority) * -1
    Else
        Dim up As _
            New ArgumentException("Nur ActiveDev.Function-Objekte können verglichen/sortiert werden")
        Throw up
    End If
End Function
End Class

```

Wichtig zu wissen: Da Funktionen sich von Entwicklern, die die Klasse verwenden möchten, zu späterer Zeit hinzufügen lassen sollen, ohne dabei den Quellcode verändern zu müssen, arbeitet die Klasse ADFormularParser nicht mit fest »verdrahteten« Funktionen. Sie benutzt vielmehr Delegaten, um die Funktionsaufrufe durchzuführen. Zum besseren Verständnis des Beispielcodes sollten Sie daher den Abschnitt über Delegaten in ► Kapitel 15 studiert haben.

Durch die Verwendung von Delegaten werden Funktionen nicht als feste Ziele durch Funktionsnamen angegeben. Funktionen selbst können in Variablen gespeichert werden (natürlich nicht die Funktionen selbst – intern werden lediglich die Zeiger auf die Adressen der Funktionen gespeichert). Sie definieren einen Delegaten mit einer Deklarationsanweisung ähnlich wie die Prozedur einer Klasse. Im Gegensatz zu einer solchen Prozedur erreichen Sie durch das Schlüsselwort `Delegate`, dass sich eine Prozedur, die über die gleiche Signatur wie der Delegat verfügt, in einer Delegaten-Variable speichern lässt. Im Beispiel wird der Delegat namens `ADFunctionDelegate` mit der folgenden Anweisung deklariert:

```
Public Delegate Function ADFunctionDelegate(ByVal parArray As Double()) As Double
```

Sie können nun eine Objektvariable vom Typ `ADFunctionDelegate` erstellen und ihr eine Funktion zuweisen. Der Zeiger auf die Funktion wird dabei in dieser Objektvariablen gespeichert. Angenommen, es gibt, wie im Beispiel, irgendwo die folgende Prozedur:

```
Public Shared Function Addition(ByVal Args() As Double) As Double
    Return Args(0) + Args(1)
End Function
```

Dann können Sie sie, da sie über die gleiche Signatur wie der Delegat verfügt, in einer Delegatenvariablen speichern – etwa wie im folgenden Beispiel:

```
Dim locDelegate As ADFunctionDelegate
locDelegate = New ADFunctionDelegate(AddressOf Addition)
```

Die Verwendung von `AddressOf` (etwa: *Adresse von*) macht deutlich, dass hier die Adresse (also ein Zeiger) der Funktion in der Delegatenvariable gespeichert wird.

Es gibt anschließend zwei Möglichkeiten, die Funktion `Addition` aufzurufen: einmal direkt über den Funktionsnamen und über den Delegaten. Der direkte Aufruf mit

```
Dim locErgebnis As Double = Addition(New Double() {10, 10})
```

bewirkt also das Gleiche wie der Umweg über den Delegaten mit

```
Dim locErgebnis As Double = locDelegate(New Double() {10, 10})
```

bloß mit einem zusätzlichen Vorteil: Sie können erst zur Laufzeit durch entsprechende Variablenzuweisung bestimmen, welche Funktion aufgerufen werden soll, wenn es mehrere Funktionen gleicher Signatur gibt.

Um beim Beispiel zu bleiben: Genau das ist der Grund, wieso den Funktionen, die das Programm berechnen kann, die Argumente als `Double`-Array und als fortlaufende Parameter übergeben werden. So kann gewährleistet werden, dass alle Funktionen die gleichen Signaturen haben. Damit kann jede Funktion in einer Delegatenvariablen gespeichert werden (vordefiniert in einer generischen `List(Of ADFunction)`-Auflistung, im nachfolgenden Listing fett markiert). Die Auswertungsroutine kann dann mithilfe der Auflistung nicht nur den Funktionsnamen finden, sondern die zugeordnete mathematische Funktion auch direkt über ihren Delegaten aufrufen.

Ansonsten hat die Klasse lediglich die Aufgabe, die Rahmendaten einer Funktion (oder eines Operators) zu speichern. Für Operatoren gibt es eine besondere Eigenschaft namens `Priority`, die es erlaubt, die Stellung eines Operators in der Hierarchieliste aller Operatoren zu bestimmen. Diese Priorität ermöglicht die Regel »Potenz-vor-Klammer-vor-Punkt-vor-Strich« einzuhalten. Bestimmte Operatoren (wie beispielsweise »`^`« für die Potenz) haben eine höhere Priorität als andere (wie beispielsweise »`+`« für die Addition).

```
Imports System.Text.RegularExpressions

Public Class ADFormularParser

    Protected myFormular As String
    Protected myFunctions As List(Of ADFunction)
    Protected myPriorizedOperators As ADPrioritizedOperators
    Protected Shared myPredefinedFunctions As List(Of ADFunction)
    Protected myResult As Double
    Protected myIsCalculated As Boolean
    Protected myConsts As ArrayList
```

```

Private myConstEnumCounter As Integer

Protected Shared myXVariable As Double
Protected Shared myYVariable As Double
Protected Shared myZVariable As Double

'Definiert die Standardfunktionen statisch bei der ersten Verwendung dieser Klasse.
Shared Sub New()

    myPredefinedFunctions = New List(Of ADFunction)

    With myPredefinedFunctions
        .Add(New ADFunction("+"c, AddressOf Addition, CByte(1)))
        .Add(New ADFunction("-"c, AddressOf Subtraction, CByte(1)))
        .Add(New ADFunction("*"c, AddressOf Multiplication, CByte(2)))
        .Add(New ADFunction("/"c, AddressOf Division, CByte(2)))
        .Add(New ADFunction("\\"c, AddressOf Remainder, CByte(2)))
        .Add(New ADFunction("^"c, AddressOf Power, CByte(3)))
        .Add(New ADFunction("PI", AddressOf PI, 1))
        .Add(New ADFunction("Sin", AddressOf Sin, 1))
        .Add(New ADFunction("Cos", AddressOf Cos, 1))
        .Add(New ADFunction("Tan", AddressOf Tan, 1))
        .Add(New ADFunction("Max", AddressOf Max, -1))
        .Add(New ADFunction("Min", AddressOf Min, -1))
        .Add(New ADFunction("Sqrt", AddressOf Sqrt, 1))
        .Add(New ADFunction("Tanh", AddressOf Tanh, 1))
        .Add(New ADFunction("LogDec", AddressOf LogDec, 1))
        .Add(New ADFunction("XVar", AddressOf XVar, 1))
        .Add(New ADFunction("YVar", AddressOf YVar, 1))
        .Add(New ADFunction("ZVar", AddressOf ZVar, 1))
    End With
End Sub

''' <summary>
''' Erstellt eine neue Instanz dieser Klasse.
''' </summary>
''' <param name="Formular">Die auszuwertende Formel, die als Zeichenkette vorliegen muss.</param>
''' <remarks></remarks>
Sub New(ByVal Formular As String)

    'Vordefinierte Funktionen übertragen
    myFunctions = myPredefinedFunctions
    myFormular = Formular
    OnAddFunctions()

End Sub

```

Die Parser-Klasse verfügt über zwei Konstruktoren – über einen statischen und einen nicht statischen. Im statischen Konstruktor, der dann aufgerufen wird, wenn die Klasse innerhalb einer Assembly das erste Mal zur Anwendung kommt, werden die Grundfunktionen der Klasse definiert, deren Code sich in der Projektdatei *ADFormularParser\_Math.vb* befindet.

Bei der eigentlichen Instanzierung der Klasse werden die so schon vorhandenen Funktionen der Klasseninstanz zugewiesen. Indem der Anwender die Klasse vererbt und die Funktion OnAddFunction überschreibt, kann er im gleichen Stil weitere Funktionen der Klasse hinzufügen. Ein Beispiel dafür folgt am Ende der Codebeschreibung.

```
'Mit dem Überschreiben dieser Funktion kann der Entwickler eigene Funktionen hinzufügen
Public Overridable Sub OnAddFunctions()
    'Nichts zu tun in der Basisversion
    Return
End Sub

'Interne Funktion, die das Berechnen startet.
Private Sub Calculate()

    Dim locFormular As String = myFormular
    Dim locOpStr As String = ""

    'Operatorenliste anlegen
    myPriorizedOperators = New ADPrioritizedOperators
    For Each adf As ADFunction In myFunctions
        If adf.IsOperator Then
            myPriorizedOperators.AddFunction(adf)
        End If
    Next

    'Operatoren Zeichenkette zusammenbauen

    For Each ops As ADFunction In myFunctions
        If ops.IsOperator Then
            locOpStr += "\\" + ops.FunctionName
        End If
    Next

    'White-Spaces entfernen
    'Syntax-Check für Klammern
    'Negativ-Vorzeichen verarbeiten
    locFormular = PrepareFormular(locFormular, locOpStr)

    'Konstanten 'rausparsen
    locFormular = GetConsts(locFormular)

    myResult = ParseSimpleTerm(Parse(locFormular, locOpStr))
    IsCalculated = True
End Sub
```

Diese Routine ist die »Zentrale« der Parser-Klasse. Hier werden alle weiteren Funktionen aufgerufen, die benötigt werden, um einen Ausdruck korrekt auszuwerten.

```
'Überschreibbare Funktion, die die Formelauswertung steuert.
Protected Overridable Function Parse(ByVal Formular As String, ByVal OperatorRegEx As String) As String

    Dim locTemp As String
    Dim locTerm As Match
```

```

Dim locFuncName As Match
Dim locMoreInnerTerms As MatchCollection
Dim locPreliminaryResult As New ArrayList
Dim locFuncFound As Boolean
Dim locOperatorRegEx As String = "\{[\d\; " + OperatorRegEx + "]*\}\"

Dim adf As ADFunction

locTerm = Regex.Match(Formular, locOperatorRegEx)
If locTerm.Value <> "" Then
    locTemp = Formular.Substring(0, locTerm.Index)

    'Befindet sich ein Funktionsname davor?
    locFuncName = Regex.Match(locTemp, "[a-zA-Z]*", RegexOptions.RightToLeft)

    'Gibt es mehrere, durch ; getrennte Parameter?
    locMoreInnerTerms = Regex.Matches(locTerm.Value, "[\d" + OperatorRegEx + "]*[;|\)]")

    'Jeden Parameterterm auswerten und zum Parameter-Array hinzufügen
    For Each locMatch As Match In locMoreInnerTerms
        locTemp = locMatch.Value
        locTemp = locTemp.Replace(";", "").Replace(")", "")
        locPreliminaryResult.Add(ParseSimpleTerm(locTemp))
    Next

    'Möglicher Syntaxfehler: Mehrere Parameter, aber keine Funktion
    If locFuncName.Value = "" And locMoreInnerTerms.Count > 1 Then
        Dim up As New SyntaxErrorException _
            ("Mehrere Klammerparameter aber kein Funktionsname angegeben!")
        Throw up
    End If

    If locFuncName.Value <> "" Then
        'Funktionsnamen suchen
        locFuncFound = False
        For Each adf In myFunctions
            If adf.FunctionName.ToUpper = locFuncName.Value.ToUpper Then
                locFuncFound = True
                Exit For
            End If
        Next

        If locFuncFound = False Then
            Dim up As New SyntaxErrorException("Der Funktionsname wurde nicht gefunden")
            Throw up
        Else
            Formular = Formular.Replace(locFuncName.Value + locTerm.Value, _
                myConstEnumCounter.ToString("000"))
            Dim locArgs(locPreliminaryResult.Count - 1) As Double
            locPreliminaryResult.CopyTo(locArgs)
            'Diese Warnung bezieht sich auf einen hypothetischen Fall,
            'der aber nie eintreten kann!
            myConsts.Add(adf.Operate(locArgs))
    End If
End If

```

```

        myConstEnumCounter += 1
    End If
Else
    Formular = Formular.Replace(locTerm.Value, myConstEnumCounter.ToString("000"))
    myConsts.Add(CDb1(locPreliminaryResult(0)))
    myConstEnumCounter += 1
End If
Else
    Return Formular
End If
Formular = Parse(Formular, OperatorRegEx)
Return Formular

End Function

'Überschreibbare Funktion, die einen einfachen Term
'(ohne Funktionen, nur Operatoren) auswertet.
Protected Overrides Function ParseSimpleTerm(ByVal Formular As String) As Double

    Dim locPos As Integer
    Dim locResult As Double

    'Klammern entfernen
    If Formular.IndexOfAny(New Char() {"(", ")"}) > -1 Then
        Formular = Formular.Remove(0, 1)
        Formular = Formular.Remove(Formular.Length - 1, 1)
    End If

    'Die Prioritäten der verschiedenen Operatoren von oben nach unten durchlaufen
    For locPrioCount As Integer = myPriorizedOperators.HighestPriority To _
        myPriorizedOperators.LowestPriority Step -1
        Do
            'Schauen, ob *nur* ein Wert
            If Formular.Length = 3 Then
                Return CDb1(myConsts(Integer.Parse(Formular)))
            End If

            'Die Operatorenzeichen einer Ebene ermitteln
            Dim locCharArray As Char() = myPriorizedOperators.OperatorChars(CByte(locPrioCount))
            If locCharArray Is Nothing Then
                'Gibt keinen Operator dieser Ebene, dann nächste Hierarchie.
                Exit Do
            End If

            'Nach einem der Operatoren dieser Hierarchieebene suchen
            locPos = Formular.IndexOfAny(locCharArray)
            If locPos = -1 Then
                'Kein Operator dieser Ebene mehr in der Formel vorhanden - nächste Hierarchie.
                Exit Do
            Else
                Dim locDblArr(1) As Double
                'Operator gefunden - Teilterm ausrechnen
                locDblArr(0) = CDb1(myConsts(Integer.Parse(Formular.Substring(locPos - 3, 3))))
            End If
        Loop
    Next
End Function

```

```

locDb1Arr(1) = CDbl(myConsts(Integer.Parse(Formular.Substring(locPos + 1, 3)))

'Die entsprechende Funktion aufrufen, die durch die Hilfsklassen
'anhand Priorität und Operatorzeichen ermittelt werden kann.
Dim locOpChar As Char = Convert.ToChar(Formular.Substring(locPos, 1))
locResult = myPriorizedOperators.OperatorByChar(
    CByte(locPrioCount), locOpChar).Operate(locDb1Arr)

'Und den kompletten Ausdruck durch eine neue Konstante ersetzen
myConsts.Add(locResult)
Formular = Formular.Remove(locPos - 3, 7)
Formular = Formular.Insert(locPos - 3, myConstEnumCounter.ToString("000"))
myConstEnumCounter += 1
End If
Loop
Next
End Function

```

Parse und ParseSimpleTerm sind die eigentlichen Arbeitspferde der Klasse. Die Funktionsweise von Parse ergibt sich aus den Kommentaren – weitere Erklärungen dazu sind deswegen überflüssig. Interessant wird es bei ParseSimpleTerm, vor allen Dingen, wenn Sie zu den Lesern gehören, die den Vorgänger dieses Buches gelesen haben: In der Vorgängerversion dieses Beispiels zu *Visual Basic .NET – Das Entwicklerbuch* hat sich nämlich ein gemeiner Fehler eingeschlichen:<sup>1</sup> ParseSimpleTerm dient dazu, dass Teilterme des gesamten Ausdrucks, die nur aus Operatoren bestehen, gemäß der Operatorenhierarchie ausgewertet werden. Dabei müssen Operatoren gleicher Priorität von links nach rechts ausgewertet werden. In der Vorgängerversion dieses Beispiels kamen Operatoren gleicher Hierarchie aber auch *nacheinander* zur Anwendung (also erst wurden Additionen, dann wurden Subtraktionen berechnet, nicht, wie es sein sollte, Additionen und Subtraktionen in einem Zug). Bei bestimmten Konstellationen hatte das falsche Ergebnisse zur Folge (beispielsweise bei dem Ausdruck  $-2+1$  der  $-3$  ergab, da zunächst alle Additionen ( $2+1$ ) und anschließend alle Subtraktionen ( $*-1$ ) durchgeführt wurden).

Mit zwei neu geschaffenen Hilfsklassen, die Sie in der Codedatei *AuxilliaryClasses.vb* finden, arbeitet ParseSimpleTerm nun so, dass Operatoren gleicher Ebene so berechnet werden, wie sie von links nach rechts betrachtet auftreten.

Die folgende Routine nutzt die Text-Analyse-Fähigkeit von Regulären Ausdrücken, um einen konstanten Ausdruck in der zu analysierenden Formel zu ermitteln. An diesem Beispiel wird einmal mehr deutlich, wie das geschickte Verwenden von Regulären Ausdrücken eine ganze Menge Programmieraufwand sparen kann.

```

'Überschreibbare Funktion, die die konstanten Zahlenwerte in der Formel ermittelt.
Protected Overrides Function GetConsts(ByVal Formular As String) As String
    Dim locRegEx As New Regex("[\d,]+[\$]*")
    'Alle Ziffern mit Komma oder Punkt aber keine Whitespaces
    myConstEnumCounter = 0
    myConsts = New ArrayList
    Return locRegEx.Replace(Formular, AddressOf EnumConstsProc)
End Function

```

---

<sup>1</sup> Mein Dank gilt deswegen an dieser Stelle Andreas Schlegel, der mich auf diesen Fehler aufmerksam gemacht hat!

```

'Rückruffunktion für das Auswerten der einzelnen Konstanten (siehe vorherige Zeile).
Protected Overrides Function EnumConstsProc(ByVal m As Match) As String
    Try
        myConsts.Add(Double.Parse(m.Value))
        Dim locString As String = myConstEnumCounter.ToString("000")
        myConstEnumCounter += 1
        Return locString
    Catch ex As Exception
        myConsts.Add(Double.NaN)
        Return "ERR"
    End Try
End Function

'Hier werden vorbereitende Arbeiten durchgeführt.
Protected Overrides Function PrepareFormular(ByVal Formular As String, _
                                              ByVal OperatorRegEx As String) As String
    Dim locBracketCounter As Integer
    'Klammern überprüfen
    For Each locChar As Char In Formular.ToCharArray
        If locChar = "("c Then
            locBracketCounter += 1
        End If
        If locChar = ")"c Then
            locBracketCounter -= 1
            If locBracketCounter < 0 Then
                Dim up As New SyntaxErrorException _
                    ("Zu viele Klammer-Zu-Zeichen.")
                Throw up
            End If
        End If
    Next
    If locBracketCounter > 0 Then
        Dim up As New SyntaxErrorException _
            ("Eine offene Klammer wurde nicht ordnungsgemäß geschlossen.")
        Throw up
    End If

    'White-Spaces entfernen
    Formular = Regex.Replace(Formular, "\s", "")

    'Vorzeichen verarbeiten
    If Formular.StartsWith("-") Or Formular.StartsWith("+") Then
        Formular = Formular.Insert(0, "0")
    End If

    'Sonderfall negative Klammer
    Formular = Regex.Replace(Formular, "\(-\(", "(0-(")

    Return Regex.Replace(Formular, _
        "(?<operator>" + OperatorRegEx + "\()-(?<zah1>[\d\.\.,]*)", _
        "${operator}((0-1)*${zah1})")
End Function

```

Und auch für die Auswertung von Vorzeichen verwendet das Programm wieder die Hilfe von Regulären Ausdrücken (im oben stehenden Listingauszug fett dargestellt).

Dabei wird, wie schon eingangs erwähnt, beispielsweise der Ausdruck

**-2\*-1^(-2-1)**

in den Ausdruck

**((0-1)\*2)\*((0-1)\*1)^(((0-1)\*2)-((0-1)\*1))**

umgewandelt – mit diesem kleinen Trick sind negative Vorzeichen berücksichtigt. Die Routine nutzt dazu die Suchen-und-Ersetzen-Funktion Replace der Regex-Klasse.

```
Public Property Formular() As String
    Get
        Return myFormular
    End Get
    Set(ByVal Value As String)
        IsCalculated = False
        myFormular = Value
    End Set
End Property

Public ReadOnly Property Result() As Double
    Get
        If Not IsCalculated Then
            Calculate()
        End If
        Return myResult
    End Get
End Property

Public Property IsCalculated() As Boolean
    Get
        Return myIsCalculated
    End Get
    Set(ByVal Value As Boolean)
        myIsCalculated = Value
    End Set
End Property

Public Property Functions() As ArrayList
    Get
        Return myFunctions
    End Get
    Set(ByVal Value As ArrayList)
        myFunctions = Value
    End Set
End Property
```

Alle fest implementierten mathematischen Operatoren und Funktionen folgen ab diesem Punkt. Beachten Sie, dass auch die hier implementierten Funktionen die Signatur des Delegaten ADFunctionDelegate voll erfüllen. Falls Sie den Formel Parser um eigene Operatoren oder Funktionen ergänzen wollen, können Sie diese prinzipiell hier entlehnen.

Wie Sie genau vorgehen, um den Formel Parser um eigene Funktionen zu erweitern, erfahren Sie im letzten Abschnitt dieses Kapitels. Die mathematischen Funktionen finden Sie übrigens ausgelagert in der Codedatei *ADFormularParser\_Math.vb*.

---

**HINWEIS:** Zwar gibt es zwei Codedateien – *ADFormularParser.vb* sowie *ADFormularParser\_Math.vb* – aber beide Codedateien enthalten dennoch Code für nur *eine* Klasse. Möglich wird das durch das Schlüsselwort Partial, mit dem Sie Klassen mit umfangreichen Code auf mehrere Codedateien verteilen können. Mehr zu partiellen Klassen finden Sie auch in ► Kapitel 6.

---

```
Partial Public Class ADFormularParser
    Public Shared Function Addition(ByVal Args() As Double) As Double
        Return Args(0) + Args(1)
    End Function

    Public Shared Function Subtraction(ByVal Args() As Double) As Double
        Return Args(0) - Args(1)
    End Function

    Public Shared Function Multiplication(ByVal Args() As Double) As Double
        Return Args(0) * Args(1)
    End Function

    Public Shared Function Division(ByVal Args() As Double) As Double
        Return Args(0) / Args(1)
    End Function

    Public Shared Function Remainder(ByVal Args() As Double) As Double
        Return Decimal.Remainder(New Decimal(Args(0)), New Decimal(Args(1)))
    End Function

    Public Shared Function Power(ByVal Args() As Double) As Double
        Return Args(0) ^ Args(1)
    End Function

    Public Shared Function Sin(ByVal Args() As Double) As Double
        Return Math.Sin(Args(0))
    End Function

    Public Shared Function Cos(ByVal Args() As Double) As Double
        Return Math.Cos(Args(0))
    End Function

    Public Shared Function Tan(ByVal Args() As Double) As Double
        Return Math.Tan(Args(0))
    End Function

    Public Shared Function Sqrt(ByVal Args() As Double) As Double
        Return Math.Sqrt(Args(0))
    End Function

    Public Shared Function PI(ByVal Args() As Double) As Double
        Return Math.PI
    End Function
```

```

Public Shared Function Tanh(ByVal Args() As Double) As Double
    Return Math.Tanh(Args(0))
End Function

Public Shared Function LogDec(ByVal Args() As Double) As Double
    Return Math.Log10(Args(0))
End Function

Public Shared Function XVar(ByVal Args() As Double) As Double
    Return XVariable
End Function

Public Shared Function YVar(ByVal Args() As Double) As Double
    Return YVariable
End Function

Public Shared Function ZVar(ByVal Args() As Double) As Double
    Return ZVariable
End Function

Public Shared Function Max(ByVal Args() As Double) As Double
    Dim retDouble As Double

    If Args.Length = 0 Then
        Return 0
    Else
        retDouble = Args(0)
        For Each locDouble As Double In Args
            If retDouble < locDouble Then
                retDouble = locDouble
            End If
        Next
    End If
    Return retDouble
End Function

Public Shared Function Min(ByVal Args() As Double) As Double
    Dim retDouble As Double

    If Args.Length = 0 Then
        Return 0
    Else
        retDouble = Args(0)
        For Each locDouble As Double In Args
            If retDouble > locDouble Then
                retDouble = locDouble
            End If
        Next
    End If
    Return retDouble
End Function

```

```

Public Shared Property XVariable() As Double
    Get
        Return myXVariable
    End Get
    Set(ByVal Value As Double)
        myXVariable = Value
    End Set
End Property

Public Shared Property YVariable() As Double
    Get
        Return myYVariable
    End Get
    Set(ByVal Value As Double)
        myYVariable = Value
    End Set
End Property

Public Shared Property ZVariable() As Double
    Get
        Return myZVariable
    End Get
    Set(ByVal Value As Double)
        myZVariable = Value
    End Set
End Property

End Class

```

---

**TIPP:** Ein paar Worte zu den Funktionen XVariable, YVariable und ZVariable möchte ich an dieser Stelle verlieren: Sie dienen dazu, mit Variablen innerhalb einer Formel zu arbeiten, die Sie wiederum vom Programm aus steuern können. Wenn Sie innerhalb der zu verarbeitenden Formel die Ausdrücke *XVar*, *YVar* und *ZVar* verwenden, werden deren Funktionsergebnisse aus den hier zu sehenden Funktionen »entnommen«. Auf diese Weise haben Sie die Möglichkeit, beispielsweise einen Funktionsplotter mit frei bestimmbaren Formeln zu realisieren oder einfach nur ein simples Programm, das Ihnen Wertetabellen von Funktionen erstellt. Sie müssen dabei lediglich die entsprechenden Werte für XVariable, YVariable und ZVariable innerhalb Ihres Programms setzen.

---

Wichtig für das korrekte Funktionieren der Auswertung für einfache Teilterme sind die folgenden beiden Hilfsklassen, die Sie in der Codedatei *AuxilliaryClasses.vb* finden.

```

Imports System.Collections.ObjectModel

''' <summary>
''' Auflistung, in der alle Operatoren gleicher Priorität gesammelt werden, damit
''' es die Möglichkeit gibt, sie von links nach rechts (in einem Rutsch) zu verarbeiten.
''' </summary>
''' <remarks></remarks>
Public Class ADOperatorsOfSamePriority
    Inherits Collection(Of ADFunction)
    Private myPriority As Byte

    Sub New()
        MyBase.New()
    End Sub

    Protected Overrides Sub InsertItem(ByVal index As Integer, ByVal item As ADFunction)
        If Not item.IsOperator Then
            Dim locEx As New _
                ArgumentException("Nur Operatoren (keine Funktionen) können dieser Auflistung hinzugefügt werden!")
            Throw locEx
        End If
        If Me.Count = 0 Then
            myPriority = item.Priority
        Else
            'Überprüfen, ob es dieselbe Priorität ist, sonst Ausnahme!
            If item.Priority <> myPriority Then
                Dim locEx As New _
                    ArgumentException("Nur Operatoren der Priorität " & myPriority & " können dieser Auflistung hinzugefügt werden!")
                Throw locEx
            End If
        End If
        MyBase.InsertItem(index, item)
    End Sub

    Protected Overrides Sub SetItem(ByVal index As Integer, ByVal item As ADFunction)
        Dim locEx As New _
            ArgumentException("Elemente können in dieser Auflistung nicht ausgetauscht werden!")
        Throw locEx
    End Sub

    ''' <summary>
    ''' Liefert die Priorität dieser Operatorenauflistung zurück.
    ''' </summary>
    ''' <value></value>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Public ReadOnly Property Priority() As Byte
        Get
            Return myPriority
        End Get
    End Property
End Class

```

```

''' <summary>
''' Fasst alle Operatorenlisten nach Priorität kategorisiert in einer übergeordneten Auflistung zusammen.
''' </summary>
''' <remarks></remarks>
Public Class ADPrioritizedOperators
    Inherits KeyedCollection(Of Byte, ADOperatorsOfSamePriority)
    Private myHighestPriority As Byte
    Private myLowestPriority As Byte

    ''' <summary>
    ''' Fügt einer der untergeordneten Auflistungen einen neuen Operator hinzu,
    ''' in Abhängigkeit von seiner Priorität.
    ''' </summary>
    ''' <param name="Function"></param>
    ''' <remarks></remarks>
    Public Sub AddFunction(ByVal [Function] As ADFunction)
        'Feststellen, ob es schon eine Auflistung für diese Operator-Priorität gibt
        If Me.Contains([Function].Priority) Then
            'Ja - dieser hinzufügen,
            Me([Function].Priority).Add([Function])
        Else
            'Nein - anlegen und hinzufügen.
            Dim locOperatorsOfSamePriority As New ADOperatorsOfSamePriority()
            locOperatorsOfSamePriority.Add([Function])
            Me.Add(locOperatorsOfSamePriority)
        End If
    End Sub

    Protected Overrides Function GetKeyForItem(ByVal item As ADOperatorsOfSamePriority) As Byte
        Return item.Priority
    End Function

    Protected Overrides Sub InsertItem(ByVal index As Integer, ByVal item As ADOperatorsOfSamePriority)

        If Me.Count = 0 Then
            myHighestPriority = item.Priority
            myLowestPriority = item.Priority
            MyBase.InsertItem(index, item)
            Return
        End If

        MyBase.InsertItem(index, item)

        If myHighestPriority < item.Priority Then
            myHighestPriority = item.Priority
        End If

        If myLowestPriority > item.Priority Then
            myLowestPriority = item.Priority
        End If
    End Sub

```

```

'<summary>
'<summary> Liefert alle Operatorzeichen einer bestimmten Priorität als Char-Array zurück.
'</summary>
'<param name="Priority">Die Priorität, deren Operatoren zusammengestellt werden sollen.</param>
'<returns></returns>
'<remarks></remarks>
Public Function OperatorChars(ByVal Priority As Byte) As Char()
    If Me.Contains(Priority) Then
        Dim locChars As New List(Of Char)
        For Each locFunction As ADFunction In Me(Priority)
            locChars.Add(Convert.ToChar(locFunction.FunctionName))
        Next
        Return locChars.ToArray
    End If
    Return Nothing
End Function

'<summary>
'<summary> Liefert die Funktion zurück, die sich durch ein Operator-Zeichen einer bestimmten Priorität ergibt.
'</summary>
'<param name="Priority">Die Priorität, die den Operatoren entspricht, die ...</param>
'<param name="OperatorChar">Das Operatorzeichen mit der angegebenen ...</param>
'<returns></returns>
'<remarks></remarks>
Public Function OperatorByChar(ByVal Priority As Byte, ByVal OperatorChar As Char) As ADFunction
    If Me.Contains(Priority) Then
        For Each locFunction As ADFunction In Me(Priority)
            If OperatorChar = Convert.ToChar(locFunction.FunctionName) Then
                Return locFunction
            End If
        Next
    End If
    Return Nothing
End Function

'<summary>
'<summary> Liefert die höchste Prioritätsnummer zurück.
'</summary>
'<value></value>
'<returns></returns>
'<remarks></remarks>
Public ReadOnly Property HighestPriority() As Byte
    Get
        Return myHighestPriority
    End Get
End Property

'<summary>
'<summary> Liefert die kleinste Prioritätsnummer zurück.
'</summary>
'<value></value>
'<returns></returns>
'<remarks></remarks>

```

```

Public ReadOnly Property LowestPriority() As Byte
    Get
        Return myLowestPriority
    End Get
End Property
End Class

```

Diese beiden Klassen dienen dazu, Operatoren nach Priorität einzuordnen. Gleichzeitig stellen sie Funktionen bereit, mit deren Hilfe man einerseits alle Operatorzeichen einer Hierarchieebene anhand der Prioritätennummer, andererseits die Funktionsklasse (ADFunction) einer Funktion mithilfe von Priorität und Operatorzeichen ermitteln kann.

Verwendet werden beide Funktionen anschließend, um folgende Konstrukte bei der Formelauswertung in den Griff zu bekommen:

$5-2*3+7/2+1$

Im ersten Schritt werden die Ausdrücke  $2*3$  und  $7/2$  verarbeitet – und zwar von links nach rechts innerhalb eines Durchlaufs. Um die Suche nach beiden Operatorzeichen dieser Ebene zu finden, verfügt die Klasse über die Methode `OperatorChars`, die ein Char-Array für die entsprechende Hierarchieebene zurückliefert – im Beispiel also die Zeichen »\*« und »/«. Mit der String-Funktion `IndexOfAny` können also beide Operatoren tatsächlich auf gleicher Ebene gefunden werden.

Um nun aber herauszufinden, welche der beiden Operatoren bei der Auswertung des Teilterms tatsächlich zur Anwendung kommen müssen, dient dann die Funktion `OperatorByChar`, die – gekapselt in `ADFunction` – den eigentlichen Delegaten zur Durchführung der Berechnung zurückliefert. Übergeben wird dieser Funktion das Operatorzeichen und die Priorität, und sie liefert anschließend die entsprechende `ADFunction`-Instanz zurück, mit der die Berechnung dann durchgeführt werden kann.

Auf diese Weise ist einerseits die beliebige Erweiterung durch zusätzliche Operatoren und andererseits die Einhaltung der Hierarchien garantiert.

## Vererben der Klasse ADFormularParser, um eigene Funktionen hinzuzufügen

Sie können die Klasse vererben, um auf einfache Weise eigene Funktionen zum Auswerten in Formeln hinzuzufügen. Das umgebende Beispielprogramm demonstriert das anhand einer einfachen Funktion, die Sie dann im Ausdruck verwenden können, wenn Sie die zweite Schaltfläche zum Auswerten verwenden:

```

Private Sub btnCalculateEx_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnCalculateEx.Click
    Dim locFormPars As New ModifiedFormularParser(txtFormular.Text)
    lblResult.Text = locFormPars.Result.ToString
End Sub
End Class

Public Class ModifiedFormularParser
    Inherits ADFormularParser

```

```

Sub New(ByVal Formular As String)
    MyBase.New(Formular)
End Sub

Public Overrides Sub OnAddFunctions()
    'Benutzerdefinierte Funktion hinzufügen.
    Functions.Add(New ADFunction("Double", AddressOf [Double], 1))
End Sub

Public Shared Function [Double](ByVal Args() As Double) As Double
    Return Args(0) * 2
End Function

End Class

```

Die Klasse ModifiedFormularParser ist, wie hier im Code gezeigt, aus der ADFormularParser-Klasse hervorgegangen. Durch Überschreiben der Funktion OnAddFunctions können Sie eigene Funktionen der Klasse hinzufügen.

Beim Überschreiben der Klasse müssen Sie nur zwei Punkte beachten:

- Funktionen, die die Klasse zusätzlich behandeln sollen, müssen, wie hier im Beispiel zu sehen, als statische Funktionen eingebunden werden (sie müssen also mit dem Attribut Shared definiert werden).

In der Methode OnAddFunctions, die Sie überschreiben müssen, können Sie die Funktionsvorschriften durch das Instanzieren zusätzlicher ADFunction-Klasse der Functions-Auflistung hinzufügen.



# 22 Serialisierung von Objekten

---

- 
- 658 Einführung in Serialisierungstechniken**
  - 666 Flaches und tiefes Klonen von Objekten**
  - 671 Serialisieren von Objekten mit Zirkelverweisen**
  - 674 XML-Serialisierung**
- 

Wenn Sie Anwendungen entwickeln, kommen Sie irgendwann zwangsläufig an den Punkt, an dem Sie die Objekte, mit deren Hilfe Sie die Daten Ihrer Anwendung verwalten, für den späteren Gebrauch sichern oder zur Weiterverarbeitung an eine andere Instanz übertragen müssen. Vom aktuellen »Zustand« eines Objektes muss in diesem Fall eine Art Momentaufnahme gemacht werden; der Speicherinhalt aller Eigenschaften und aller untergeordneten Objekte muss dabei gesichert werden. Dabei spielt es natürlich erst einmal keine Rolle, auf welche Weise die Daten gesichert werden: Sie können byteweise – so, wie sie im Arbeitsspeicher stehen – direkt dort ausgelesen und in eine Datei geschrieben werden; denkbar wäre auch, dass sie zuvor durch entsprechende Algorithmen komprimiert und erst dann in einer Datei gespeichert werden. Die Daten eines Objektes, wie Zahlen oder Datumswerte, könnten auch zuvor in ein vom Anwender lesbares Format umgewandelt und speziell formatiert werden, sodass ein Transfervorgang sie auch beispielsweise im *XML*- oder *Soap*-Format direkt über das Internet zu einem anderen Server transportieren könnte.

Ganz gleich wie die Daten aus einem Objekt »geholt« und anschließend an eine andere Stelle verfrachtet werden – die Prozedur, die diese Aufgabe erfüllt, kann nur nach einem bestimmten Schema vorgehen: Sie muss *der Reihe nach* alle Eigenschaften und Variablen des Objektes abfragen, sie aufbereiten und anschließend mit den aufbereiteten Daten irgendetwas anstellen. Bei diesem Vorgang spricht man vom Serialisieren von Objekten.

Auch der umgekehrte Weg ist notwendig: Wenn Sie beispielsweise eine Reihe von Adressobjekten in eine Textdatei serialisiert haben, damit die Daten nach Beenden des Adressprogramms und Ausschalten des Computers erhalten bleiben, muss der umgekehrte Prozess stattfinden, sobald der Anwender den Computer einschaltet, die Adressverwaltung startet und mit den zuvor erfassten Adressen weiterarbeiten möchte. In diesem Fall müssen die Objekte wieder den gleichen Zustand annehmen, den sie vor dem Serialisieren hatten. Sie müssen jetzt aus der Textdatei *deserialisiert* werden.

Nun wäre das Framework nicht das Framework, wenn es Sie bei diesen Vorgängen, die natürlich in jeder Anwendung vorkommen, nicht unterstützen würde. Sie brauchen also nicht selbst Hand anzulegen und jede einzelne Eigenschaft eines Objektes auszulesen und in eine Textdatei zu schreiben. Umgekehrt müssen Sie Adressobjekte auch nicht selbst in Ihrer Anwendung komplett neu instanzie-

ren, während Sie sie aus einer Textdatei wieder einlesen. Das Framework hält für diesen Zweck einige geniale Werkzeuge bereit, die Sie in den folgenden Abschnitten kennen lernen sollten, da sie Ihnen die Arbeit erheblich erleichtern können – nicht nur, wenn Sie die Adressen Ihrer Adressverwaltung auf der Festplatte Ihres Computers speichern wollen.

## Einführung in Serialisierungstechniken

Bevor Sie sich anschauen, was das Framework an Serialisierungstechniken zu bieten hat, lassen Sie uns zunächst einen Blick auf ein ganz simples Beispiel werfen. Dieses Beispielprogramm macht nichts weiter, als den Anwender eine Adresse eingeben zu lassen und diese durch Klick auf eine Schaltfläche in eine Datei zu serialisieren (sprich: zu sichern).

---

**BEGLEITDATEIEN:** Sie finden dieses Beispiel im Verzeichnis `.\VB 2005 - Entwicklerbuch\E - Datentypen\Kap22\Serialization01`.

---



**Abbildung 22.1:** Die Nur-Lesen-Eigenschaften werden beim Serialisieren nicht übernommen

Das Beispiel nutzt dabei zunächst noch keine Serialisierungstechniken des Frameworks, sondern liest die einzelnen Eigenschaften sozusagen manuell aus und speichert sie als Text in einer Datei. Beim Klicken auf die Schaltfläche *Serialisieren* generiert die Prozedur, die das Click-Ereignis behandelt, ein Adresse-Objekt aus den Feldinhalten des Dialoges. Dieses Objekt übergibt die Routine einer weiteren Prozedur, die eine Datei zum Schreiben öffnet, die Eigenschaften nacheinander ausliest und sie als String in die Datei schreibt:

```
Private Sub btnSerialisieren_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnSerialisieren.Click
    Dim locAdresse As New Adresse(txtVorname.Text, _
        txtNachname.Text, _
        txtStraße.Text, _
        txtPLZOrt.Text)
    'Einen "unmöglichen" Dateinamen verwenden, damit, wie es der Zufall will,
    'nicht eine andere wichtige Datei gleichen Namens überschrieben wird.
    Adresse.SerializeToFile(locAdresse, "C:\serializedemo_f4e3w21.txt")

    'Info über den Datensatz anzeigen.
    txtErstelltAm.Text = locAdresse.ErfasstAm.ToString("dd.MM.yyyy HH:mm:ss")
    txtErstelltVon.Text = locAdresse.ErfasstVon
End Sub
```

Die Adresse-Klasse hält die notwendigen Elemente zum Speichern der Adressdaten und die statische Prozedur zum Serialisieren der Objektdaten in eine Datei bereit:

```
Public Class Adresse
    Private myName As String
    Private myVorname As String
    Private myStraße As String
    Private myPLZOrt As String
    Private myErfasstAm As DateTime
    Private myErfasstVon As String

    Sub New(ByVal Vorname As String, ByVal Name As String, ByVal Straße As String, ByVal PLZOrt As String)
        'Konstruktor legt alle Member-Daten an.
        myName = Name
        myVorname = Vorname
        myStraße = Straße
        myPLZOrt = PLZOrt
        myErfasstAm = DateTime.Now
        myErfasstVon = Environment.UserName
    End Sub

    'Alle öffentlichen Felder in die Datei schreiben.
    Public Shared Sub SerializeToFile(ByVal adr As Adresse, ByVal Filename As String)
        Dim locStreamWriter As New StreamWriter(Filename, False, System.Text.Encoding.Default)
        With locStreamWriter
            .WriteLine(adr.Vorname)
            .WriteLine(adr.Name)
            .WriteLine(adr.Straße)
            .WriteLine(adr.PLZOrt)
            .Flush()
            .Close()
        End With
    End Sub

    'Aus der Datei lesen und daraus ein neues Adressobjekt erstellen.
    Public Shared Function SerializeFromFile(ByVal Filename As String) As Adresse
        'Interessiert an dieser Stelle nicht, deswegen ausgelassen.
    End Function

    Public Property Name() As String
        Get
            Return myName
        End Get
        Set(ByVal Value As String)
            myName = Value
        End Set
    End Property
```

```

'Die beiden folgenden Eigenschaften haben "Nur-Lesen-Status", da auch
'der Entwickler das Anlegen-Datum nicht manipulieren darf!
Public ReadOnly Property ErfasstAm() As DateTime
    Get
        Return myErfasstAm
    End Get
End Property

Public ReadOnly Property ErfasstVon() As String
    Get
        Return myErfasstVon
    End Get
End Property

#Region "Die anderen Eigenschaften"
'Aus Platzgründen ausgelassen.
#End Region
End Class

```

Sie sehen anhand dieses Codelistings, dass es zwei Eigenschaften gibt, die zwar gelesen, aber nicht geschrieben werden dürfen: Diese Eigenschaften, die Auskunft darüber geben, wer das Adresse-Objekt zu welchem Zeitpunkt angelegt hat, dürfen ausschließlich bei der Objekterstellung definiert werden.

Bei Serialisieren auf die herkömmliche Art und Weise, wie hier im Beispiel zu sehen, ergibt sich hier schon ein Problem: Wenn Sie das Programm nämlich beenden, anschließend erneut starten und die Deserialisieren-Schaltfläche betätigen, dann wird zwar aus der Datei ein neues Adresse-Objekt erzeugt. Dieses Objekt entspricht dem ursprünglichen aber nicht in allen Details, denn: Das Erstellungsdatum und der »Ersteller« des Objektes selber hätten zwar noch mitgesichert werden können; der Deserialisierungs-Algorithmus hat aber auf Grund der Beschaffenheit dieser zusätzlichen Eigenschaften keine Möglichkeit, die Originalzustände dieser Eigenschaft wieder einzulesen. Er könnte diese Zusatzinformationen dem Objekt schlachtrweg nicht zuordnen.

Diese Tatsache spiegelt sich im Programm wider: Wann immer Sie das ursprünglich gespeicherte Objekt durch Klick auf die entsprechende Schaltfläche deserialisieren, steht in den unteren Infofeldern nicht das ursprüngliche Erstellungsdatum, sondern das Erstellungsdatum des Objektes zum Zeitpunkt des Deserialisierens.

## Serialisieren mit SoapFormatter und BinaryFormatter

Anders sieht es aus, wenn Sie das Serialisieren und das Deserialisieren mit bestimmten Hilfsmitteln aus dem Framework durchführen. Sie brauchen sich in diesem Fall nämlich nicht um das Auslesen der Eigenschaften der Objekte selbst zu kümmern – das Framework macht das automatisch für Sie. Und noch mehr: Einige Klassen des Frameworks, die für das Serialisieren von Objektdaten zuständig sind, erlauben auch das Serialisieren und Deserialisieren von privaten Eigenschaften einer Klasse.

---

**WICHTIG:** Für alle Objekte, die serialisiert werden sollen, gilt: Sie müssen mit einem besonderen Attribut namens `Serializable` gekennzeichnet werden. Wenn diese Voraussetzungen erfüllt sind, können Sie zwei der Serializer-Klassen für die Serialisierung und Deserialisierung einer so gekennzeichneten Objekt-Instanz verwenden, die auch private Member-Variablen einer Klasse berücksichtigen.

---

- **SoapFormatter-Klasse:** Stellt Serialisierungs- und Deserialisierungsfunktionen zur Verfügung, die das SOAP-Format verwenden. Die Dateninhalte eines Objektes werden dabei in reinen Text umgewandelt, der auf der einen Seite auch für den Anwender verständlich mit einem Texteditor gelesen und angezeigt und auf der anderen Seite – da er das SOAP-Format einhält – auch problemlos über das Internet transportiert werden kann. Der Nachteil: Diese Art der Datenspeicherung ist nicht sonderlich effizient, eben durch die Konvertierung nativer Daten in lesbaren Text.
- **BinaryFormatter-Klasse:** Stellt Serialisierungs- und Deserialisierungsfunktionen zur Verfügung, die das Binärformat verwenden. Die Dateninhalte eines Objektes werden dabei so verwendet, wie sie im Arbeitsspeicher vorliegen. Werden die Daten eines Objektes unter Verwendung dieses *Serializers* beispielsweise in einer Datei gespeichert, erfolgt die Datenspeicherung sehr kompakt. Eine auf diese Weise generierte Datei ist aber für den Anwender direkt nicht lesbar, da sie die Daten des Objektes im binären Format enthält.

---

**BEGLEITDATEIEN:** Das folgende Beispiel demonstriert den Einsatz mit diesen beiden Klassen. Sie finden dieses Beispiel unter *.\VB 2005 - Entwicklerbuch\E - Datentypen\Kap22\Serialization02*.

---



**Abbildung 22.2:** In dieser Version können Sie zwischen Soap- und Binärserialisierung wählen. Die Nur-Lesen-Eigenschaften bleiben jetzt beim Deserialisieren erhalten

Wenn Sie dieses Programm starten, wählen Sie für das Serialisieren bzw. Deserialisieren das gewünschte Format aus. Wenn Sie sich für das Soap-Format entscheiden, generiert die SoapFormatter-Klasse eine Datei, die der folgenden entspricht (vorausgesetzt natürlich, Sie haben die gleichen Daten eingegeben, wie in Abbildung 22.2 zu sehen):

```

<SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsds="http://www.w3.org/2001/XMLSchema" xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:clr="http://schemas.microsoft.com/soap/encoding/clr/1.0" SOAP-
  ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <a1:Adresse id="ref-1">
      xmlns:a1="http://schemas.microsoft.com/clr/nsassem/Serialization02/Serialization02%20Version%3D1.0.0.0%20
      %20Culture%3Dneutral%20%20PublicKeyToken%3Dnull">
        <myName id="ref-3">Thiemann</myName>
        <myVorname id="ref-4">Uwe</myVorname>
        <myStraße id="ref-5">Autorenstraße 34</myStraße>
        <myPLZOrt id="ref-6">99999 Buchhausen</myPLZOrt>
        <myErfasstAm>2006-02-15T11:29:24.1295355+01:00</myErfasstAm>
        <myErfasstVon id="ref-7">ACTIVEDEVELOP\loeffel</myErfasstVon>
    </a1:Adresse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Sie werden überrascht sein, wenn Sie sehen, mit wie wenig Aufwand die Serialisierung im Programm realisiert wurde. Wenn Sie einen Blick in den Projektmappen-Explorer werfen, finden Sie dort im Gegensatz zum vorherigen Beispiel zwei weitere Klassendateien. Sie enthalten die Kapselungen von SoapFormatter und BinaryFormatter zur Serialisierung (und Deserialisierung) beliebiger, serialisierbarer Objekte in Dateien, die folgendermaßen aussehen.

### Universeller Soap-Datei-De-/Serializer

```
Imports System.Runtime.Serialization
Imports System.Runtime.Serialization.Formatters.Soap
Imports System.IO

Public Class ADSoapSerializer

    Shared Sub SerializeToFile(ByVal FileInfo As FileInfo, ByVal [Object] As Object)

        Dim locFs As FileStream = New FileStream(FileInfo.FullName, FileMode.Create)
        Dim locSoapFormatter As New SoapFormatter(Nothing,
            New StreamingContext(StreamingContextStates.File))
        locSoapFormatter.Serialize(locFs, [Object])
        locFs.Flush()
        locFs.Close()

    End Sub

    Shared Function DeserializeFromFile(ByVal FileInfo As FileInfo) As Object

        Dim locObject As Object

        Dim locFs As FileStream = New FileStream(FileInfo.FullName, FileMode.Open)
        Dim locSoapFormatter As New SoapFormatter(Nothing,
            New StreamingContext(StreamingContextStates.File))
        locObject = locSoapFormatter.Deserialize(locFs)
        locFs.Close()
        Return locObject

    End Function

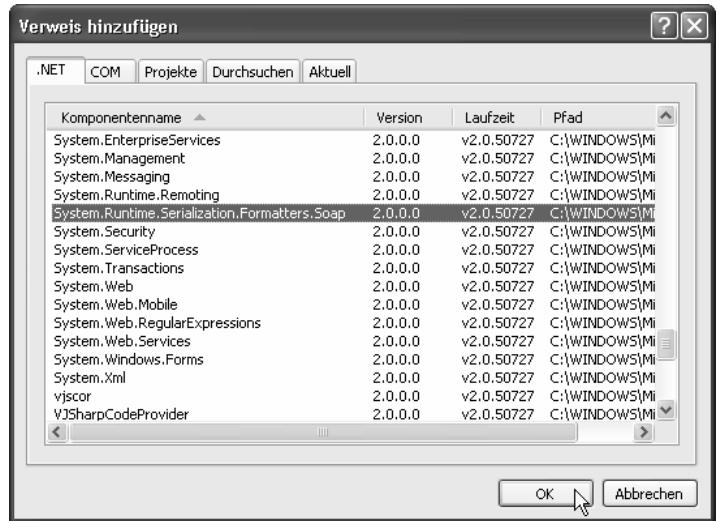
End Class
```

Wenn Sie SoapFormatter verwenden wollen, müssen Sie einen Verweis in das entsprechende Projekt einbauen. Dazu öffnen Sie im Projektmappen-Explorer über dem Projektnamen oder dem Ordner *Verweise* (wenn das Symbol *alle Dateien anzeigen aktiviert* ist) mit der rechten Maustaste das Kontextmenü und wählen dort den Eintrag *Verweise*. Im Dialog, der sich jetzt öffnet, wählen Sie den Eintrag *System.Runtime.Serialization.Formatters.Soap* per Doppelklick aus (etwa wie in Abbildung 22.3 zu sehen).

Verlassen Sie den Dialog mit *OK*. Achten Sie ebenfalls darauf, die Anweisungen

```
Imports System.Runtime.Serialization
Imports System.Runtime.Serialization.Formatters.Soap
```

als erste Zeilen in der Codedatei zu platzieren, in der Sie SoapFormatter verwenden möchten.



**Abbildung 22.3:** Wenn Sie mit der *SoapFormatter*-Klasse Objektdaten im Soap-Format serialisieren wollen, binden Sie diesen Verweis in Ihr Projekt ein

## Universeller Binary-Datei-De-/Serializer

```

Imports System.Runtime.Serialization
Imports System.Runtime.Serialization.Formatters.Binary
Imports System.IO

Public Class ADBinarySerializer

    Shared Sub SerializeToFile(ByVal FileInfo As FileInfo, ByVal [Object] As Object)

        Dim locFs As FileStream = New FileStream(FileInfo.FullName, FileMode.Create)
        Dim locBinaryFormatter As New BinaryFormatter(Nothing, _
            New StreamingContext(StreamingContextStates.File))
        locBinaryFormatter.Serialize(locFs, [Object])
        locFs.Flush()
        locFs.Close()

    End Sub

    Shared Function DeserializeFromFile(ByVal FileInfo As FileInfo) As Object

        Dim locObject As Object

        Dim locFs As FileStream = New FileStream(FileInfo.FullName, FileMode.Open)
        Dim locBinaryFormatter As New BinaryFormatter(Nothing, _
            New StreamingContext(StreamingContextStates.File))
        locObject = locBinaryFormatter.Deserialize(locFs)
        locFs.Close()
        Return locObject

    End Function

End Class

```

Achten Sie darauf, die Anweisungen

```
Imports System.Runtime.Serialization  
Imports System.Runtime.Serialization.Formatters.Binary
```

als erste Zeilen in der Codedatei zu platzieren, in der Sie BinaryFormatter verwenden möchten. Einen speziellen Verweis brauchen Sie, anders als bei SoapFormatter, nicht in das Projekt einzubinden.

### Funktionsweise der Datei-Serializer-Klassen

Beide Klassen funktionieren exakt nach dem gleichen Prinzip, sie verwenden lediglich unterschiedliche Formatter (also Klassen, die Datenaufbereitungslogiken zur Verfügung stellen), um unterschiedliche Formate zu erzeugen bzw. für die Rekreation des Ursprungobjektes zu verwenden.

Sie öffnen beim Serialisieren zunächst einen Dateistrom, über den die Daten über den jeweiligen Formatter aus dem Objekt in die Datei gelangen. In der anschließenden Instanzierung des Formatters wird dieser durch die Übergabe einer StreamingContext-Instanz darauf vorbereitet, welches Ziel die Objektserialisierung haben wird (in diesem Fall werden die Objektdaten in eine Datei serialisiert). Die eigentliche Serialisierung des Objektes geschieht dann durch die einzige Zeile:

```
locSoapFormatter.Serialize(locFs, [Object])
```

Der Rest der Prozedur ist obligatorisch: Alle Stromdaten werden mit Flush aus dem internen Puffer geleert, und die Datei wird anschließend geschlossen. Das Ergebnis: Die Objektdaten wurden einschließlich ihrer privaten Member in die Datei geschrieben. Der umgekehrte Weg beim Deserialisieren erfolgt äquivalent.

---

**TIPP:** Wenn Sie diese Art der Serialisierung in Dateien in Ihren eigenen Programmen verwenden wollen, kopieren Sie die Codedateien ADBinarySerializer.vb bzw. ADSoapSerializer.vb einfach in Ihr Projektverzeichnis und fügen sie anschließend Ihrem Projekt hinzu. Sie können sie dann auf genauso einfache Weise verwenden, wie sie im Beispielprojekt vom Adressobjekt verwendet wurden (siehe in Fettschrift gesetzte Zeilen am Ende des Codelistings):

---

```
<Serializable()>  
_  
Public Class Adresse  
  
    Private myName As String  
    Private myVorname As String  
    Private myStraße As String  
    Private myPLZOrt As String  
    Private myErfasstAm As DateTime  
    Private myErfasstVon As String  
  
    Sub New(ByVal Vorname As String, ByVal Name As String, ByVal Straße As String, ByVal PLZOrt As String)  
        'Konstruktor legt alle Member-Daten an.  
        myName = Name  
        myVorname = Vorname  
        myStraße = Straße  
        myPLZOrt = PLZOrt  
        myErfasstAm = DateTime.Now  
        myErfasstVon = Environment.UserName  
    End Sub
```

```

'Alle öffentlichen Felder in die Datei schreiben - Soap-Format
Public Shared Sub SerializeSoapToFile(ByVal adr As Adresse, ByVal Filename As String)
    ADSoapSerializer.SerializeToFile(New FileInfo(Filename), adr)
End Sub

'Aus der Datei lesen und daraus ein neues Adressobjekt erstellen - Soap-Format.
Public Shared Function SerializeSoapFromFile(ByVal Filename As String) As Adresse
    Return CType(ADSoapSerializer.DeserializeFromFile(New FileInfo(Filename)), Adresse)
End Function

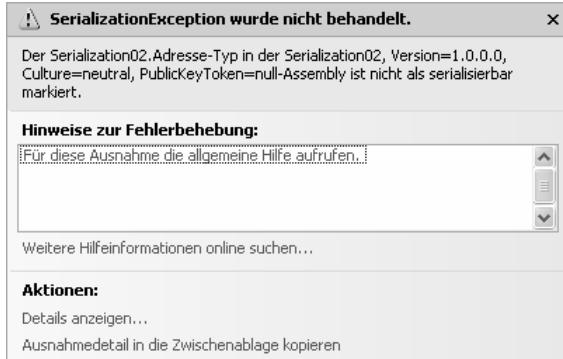
'Alle öffentlichen Felder in die Datei schreiben - Binary-Format.
Public Shared Sub SerializeBinToFile(ByVal adr As Adresse, ByVal Filename As String)
    ADBinarySerializer.SerializeToFile(New FileInfo(Filename), adr)
End Sub

'Aus der Datei lesen und daraus ein neues Adressobjekt erstellen - Binary-Format.
Public Shared Function SerializeBinFromFile(ByVal Filename As String) As Adresse
    Return CType(ADBinarySerializer.DeserializeFromFile(New FileInfo(Filename)), Adresse)
End Function

.
.
.

```

**WICHTIG:** Wenn Sie die Objektserialisierung anwenden, achten Sie darauf, dass alle Objekte, die Sie serialisieren wollen, das `Serializable`-Attribut tragen – so wie auch im vorherigen Codelisting zu sehen (siehe erste beiden Zeilen). Wenn die Klasse, die Sie serialisieren möchten, oder eine Objektinstanz, die diese Klasse einbindet, dieses Attribut nicht trägt, löst das Framework beim Serialisierungsversuch durch eines der zu serialisierenden Objekte eine Ausnahme aus, etwa wie in Abbildung 22.4 zu sehen.



**Abbildung 22.4:** Wenn eine zu serialisierende Klasse oder ein Objekt, das sie einbindet, nicht mit dem Attribut `Serializable` gekennzeichnet ist, löst das Framework beim Serialisierungsversuch diese Ausnahme aus

# Flaches und tiefes Klonen von Objekten

Der Vorteil beim Serialisieren über Funktionen des Frameworks ist, dass die Serialisierungsalgorithmen in der Lage sind, automatisch so genannte »tiefe Klons« (vollständige Kopien) eines Objektes zu erstellen.

Dazu folgender Hintergrund: Angenommen, Sie haben ein Objekt, das die Daten Ihrer Applikation speichert. Dieses Objekt verfügt dann beispielsweise über eine Eigenschaft, die eine ArrayList zur Verfügung stellt, in der weitere Elemente gespeichert sind. Um eine komplette Kopie dieses Objektes zu erstellen, würde es nicht ausreichen, die Elemente, die diese ArrayList-Eigenschaft beinhaltet, zu kopieren, wie das folgende Beispiel zeigt:

---

**BEGLEITDATEIEN:** Sie finden dieses Beispiel unter *.\VB 2005 - Entwicklerbuch\E - Datentypen\Kap22\DeepCloning*.

---

```
Module mdlMain
    Sub Main()
        Dim locAdrOriginal As New Adresse("Hans", "Mustermann", "Musterstraße 22", "59555 Lippstadt")
        Dim locAdrKopie As Adresse
        With locAdrOriginal.BefreundetMit
            .Add(New Adresse("Uwe", "Thiemann", "Autorstr. 33", "49595 Buchhausen"))
            .Add(New Adresse("Gaby", "Halek", "Autorstr. 34", "49595 Buchhausen"))
            .Add(New Adresse("Gabriele", "Löffelmann", "Autorstr. 35", "49595 Buchhausen"))
        End With
        'Originaladresse ausgeben.
        Console.WriteLine("Original:")
        Console.WriteLine("=====")
        Console.WriteLine(locAdrOriginal)

        'Kopie anlegen.
        With locAdrOriginal
            locAdrKopie = New Adresse(.Vorname, .Name, .Straße, .PLZOrt)
            locAdrKopie.Name += " (Kopie)"
            locAdrKopie.BefreundetMit = .BefreundetMit
        End With

        'Kopie ausgeben.
        Console.WriteLine("Kopie:")
        Console.WriteLine("=====")
        Console.WriteLine(locAdrKopie)

        'Änderungen im Original:
        CType(locAdrOriginal.BefreundetMit(1), Adresse).Name = "Löffelmann-Halek"
        CType(locAdrOriginal.BefreundetMit(2), Adresse).Name = "Löffelmann-Halek"

        'Kopie nach Änderungen im Original:
        Console.WriteLine("Kopie nach Änderung im Original:")
        Console.WriteLine("=====")
        Console.WriteLine(locAdrKopie)
        Console.ReadLine()
    End Sub
End Module
```

Dieses Beispiel nutzt die leicht abgewandelte Adresse-Klasse, die Sie schon aus dem vorherigen Beispiel kennen. Sie verfügt über eine zusätzliche Eigenschaft namens BefreundetMit. Diese Eigenschaft entspricht einer ArrayList mit der Aufgabe, andere Adressen aufzunehmen, die bestimmen, mit wem dieser Kontakt befreundet ist.

```
<Serializable()> _
Public Class Adresse

    Private myName As String
    Private myVorname As String
    Private myStraße As String
    Private myPLZOrt As String
    Private myErfasstAm As DateTime
    Private myErfasstVon As String
    Private myBefreundetMit As ArrayList

    Sub New(ByVal Vorname As String, ByVal Name As String, ByVal Straße As String, ByVal PLZOrt As String)
        'Konstruktor legt alle Member-Daten an.
        myName = Name
        myVorname = Vorname
        myStraße = Straße
        myPLZOrt = PLZOrt
        myErfasstAm = DateTime.Now
        myErfasstVon = Environment.UserName
        myBefreundetMit = New ArrayList
    End Sub

    Public Property BefreundetMit() As ArrayList
        Get
            Return myBefreundetMit
        End Get
        Set(ByVal Value As ArrayList)
            myBefreundetMit = Value
        End Set
    End Property

    #Region "Die anderen Eigenschaften"
    'Aus Platzgründen ausgelassen.
    #End Region

    Public Overrides Function ToString() As String
        Dim locTemp As String
        locTemp = Name + ", " + Vorname + ", " + Straße + ", " + PLZOrt + vbNewLine
        locTemp += "---- Befreundet mit: ---" + vbNewLine
        For Each locAdr As Adresse In BefreundetMit
            locTemp += "    * " + locAdr.ToStringShort() + vbNewLine
        Next
        locTemp += vbNewLine
        Return locTemp
    End Function
```

```

Public Function ToStringShort() As String
    Return Name + ", " + Vorname
End Function
End Class

```

Zusätzlich gibt es in dieser Klasse zwei Funktionen – `ToString` und `ToStringShort` – die eine Adresse-Instanz in einen String umwandeln, damit die Ausgabe einfacher wird.

Das Programm macht nun Folgendes: Es legt eine Originaladresse an und fügt ihr weitere Adress-objekte hinzu, die in der `ArrayList`, die durch `BefreundetMit` offen gelegt wird, gespeichert werden. Anschließend erzeugt es eine identische Kopie der Originaladresse im Objekt `locAdrKopie`.

Das Problem: An dieser Stelle wird eine so genannte flache Kopie des Objektes erstellt. Nur die Eigenschaften der oberen Ebene werden in die Kopie übernommen. Das wird auch deutlich, wenn Sie das Beispielprogramm starten:

Original:

```

=====
Mustermann, Hans, Musterstraße 22, 59555 Lippstadt
--- Befreundet mit: ---
* Thiemann, Uwe
* Halek, Gaby
* Löffelmann, Gabriele

```

Kopie:

```

=====
Mustermann (Kopie), Hans, Musterstraße 22, 59555 Lippstadt
--- Befreundet mit: ---
* Thiemann, Uwe
* Halek, Gaby
* Löffelmann, Gabriele

```

Kopie nach Änderung im Original:

```

=====
Mustermann (Kopie), Hans, Musterstraße 22, 59555 Lippstadt
--- Befreundet mit: ---
* Thiemann, Uwe
* Löffelmann-Halek, Gaby
* Löffelmann-Halek, Gabriele

```

Beim Ändern der Elemente der `BefreundetMit`-`ArrayList` werden auch die Elemente der Kopie verändert. Und das muss auch so sein, denn: Die Eigenschaft `BefreundetMit` stellt ja nicht wirklich selbst eine `ArrayList` dar, sondern verweist lediglich auf sie. Tatsächlich gibt es die Elemente `ArrayList` nur ein einziges Mal. In diesem Beispiel ist also nur eine flache Kopie (»shallow clone« bzw. »shallow copy« auf englisch) des Adresse-Objektes erstellt worden.

Anders wird das, wenn Sie eine tiefe Kopie (»deep clone« bzw. »deep copy« auf englisch) erstellen. Hier werden die Elemente der `ArrayList`, um beim Beispiel zu bleiben, wirklich kopiert – es gibt nach Abschluss der Kopierstellung wirklich zwei völlig unabhängige Objekte, mit ebenso unabhängigen Elementen.

Was bei diesem Beispiel mit noch vergleichsweise wenig Aufwand durchführbar wäre, sieht bei wirklich komplexen Objekten schon anders aus. Wollten Sie eine DeepCopy-Routine selber implementieren, bedeutete dies einen enormen Aufwand. Obendrein könnten Sie eine solche Routine mit normalen Mitteln nicht universal gestalten; sie würde sich nur auf das aktuelle Objekt beziehen. Bei einer Klassenableitung, die die Basisklasse um weitere Eigenschaften ergänzen würde, müssten Sie die DeepCopy-Routine schon wieder modifizieren.

Doch diesen Aufwand müssen Sie auch gar nicht betreiben, denn mit den Serialierungsfunktionen nimmt Ihnen das Framework diesen kompletten Aufwand ab. Beim Serialisieren erstellt das Framework nämlich eine tiefe Kopie.<sup>1</sup> Und da Serialisierung natürlich nicht notwendigerweise bedeutet, das Objekt in einer Datei zwischenzuspeichern, können Sie mit einem kleinen Trick eine universelle DeepCopy-Routine kreieren, die mit jedem serialisierbaren Objekt funktioniert, wie das folgende Beispiel zeigt.

## Universelle DeepClone-Methode

Das folgende Beispiel verwendet ebenfalls den BinarySerializer für das Serialisieren und das Deserialisieren eines Objektes, doch nutzt er einen anderen Träger im Vergleich zum letzten Beispiel. Ein Objekt, das es zu serialisieren gilt, wird nicht in eine Datei, sondern in ein MemoryStream-Objekt geschrieben, das schließlich in ein Byte-Array konvertiert wird. Im umgekehrten Fall erzeugt der BinarySerializer aus einem Byte-Array, das in ein MemoryStream-Objekt umgewandelt wird, wieder das ursprüngliche Objekt. Da der BinarySerializer grundsätzlich auch verschachtelte Eigenschaften verarbeitet, lässt sich daraus auf einfache Art und Weise eine Klasse erstellen, die von jedem beliebigen Objekt, das selbst als serialisierbar gekennzeichnet ist und auch nur selbst serialisierbare Objekte verwendet, eine tiefe Kopie anfertigen kann.

---

**BEGLEITDATEIEN:** Sie finden dieses Beispiel unter .\VB 2005 - Entwicklerbuch\E - Datentypen\Kap22\UniversalDeepCloning\.

---

```
Imports System.Runtime.Serialization
Imports System.Runtime.Serialization.Formatters.Binary
Imports System.IO

Public Class ADObjectCloner

    Public Shared Function DeepCopy(ByVal [Object] As Object) As Object
        Return DeserializeFromByteArray(SerializeToByteArray([Object]))
    End Function

    Shared Function SerializeToByteArray(ByVal [Object] As Object) As Byte()
        Dim retByte() As Byte
        Dim locMs As MemoryStream = New MemoryStream

```

---

<sup>1</sup> Wobei diese Aussage nicht hundertprozentig richtig ist, denn eigentlich erstellt das Framework ja keine Kopie des Objektes, sondern liest beim Serialisieren zunächst nur seine Daten aus. Das macht es aber bis in die unterste Ebene, sodass man den kombinierten Vorgang von Serialisieren eines Objektes in einen Datenstrom und Deserialisieren dieses Datenstroms in ein neues Objekt durchaus »tiefer Kopieren« des Objektes nennen kann.

```

Dim locBinaryFormatter As New BinaryFormatter(Nothing,
    New StreamingContext(StreamingContextStates.Clone))
locBinaryFormatter.Serialize(locMs, [Object])
locMs.Flush()
locMs.Close()
retByte = locMs.ToArray()
Return retByte

End Function

Shared Function DeserializeFromByteArray(ByVal by As Byte()) As Object

    Dim locObject As Object

    Dim locFs As MemoryStream = New MemoryStream(by)
    Dim locBinaryFormatter As New BinaryFormatter(Nothing,
        New StreamingContext(StreamingContextStates.File))
    locObject = locBinaryFormatter.Deserialize(locFs)
    locFs.Close()
    Return locObject

End Function

```

End Class

Das Hauptprogramm verwendet im folgenden Beispiel nun nicht mehr eigenen Code zum Kopieren des Objektes, sondern benutzt die DeepCopy-Funktion der Klasse (die geänderte Passage ist fett hervorgehoben):

```

Module mdlMain

    Sub Main()
        Dim locAdrOriginal As New Adresse("Hans", "Mustermann", "Musterstraße 22", "59555 Lippstadt")
        Dim locAdrKopie As Adresse
        With locAdrOriginal.BefreundetMit
            .Add(New Adresse("Uwe", "Thiemann", "Autorstr. 33", "49595 Buchhausen"))
            .Add(New Adresse("Gaby", "Halek", "Autorstr. 34", "49595 Buchhausen"))
            .Add(New Adresse("Gabriele", "Löffelmann", "Autorstr. 35", "49595 Buchhausen"))
        End With

        'Originaladresse ausgeben.
        Console.WriteLine("Original:")
        Console.WriteLine("=====")
        Console.WriteLine(locAdrOriginal)

        'Kopie anlegen.
        locAdrKopie = CType(ADObjectCloner.DeepCopy(locAdrOriginal), Adresse)

        'Kopie ausgeben.
        Console.WriteLine("Kopie:")
        Console.WriteLine("=====")
        Console.WriteLine(locAdrKopie)
    End Sub

```

```

'Änderungen im Original:
(CType(locAdrOriginal.BefreundetMit(1), Adresse).Name = "Löffelmann-Halek"
(CType(locAdrOriginal.BefreundetMit(2), Adresse).Name = "Löffelmann-Halek"

'Kopie nach Änderungen im Original:
Console.WriteLine("Kopie nach Änderung im Original:")
Console.WriteLine("====")
Console.WriteLine(locAdrKopie)
Console.ReadLine()

End Sub

```

End Module

Wenn Sie das Programm anschließend starten, erkennen Sie den Unterschied zum vorherigen Beispiel. Die beiden erzeugten Objekte sind wirklich komplett unabhängig voneinander. Änderungen an der ArrayList des Ausgangsobjektes beeinflussen das Ergebnis der Ausgabe der Objektkopie in keiner Weise:

Original:

```

=====
Mustermann, Hans, Musterstraße 22, 59555 Lippstadt
--- Befreundet mit: ---
* Thiemann, Uwe
* Halek, Gaby
* Löffelmann, Gabriele

```

Kopie:

```

=====
Mustermann, Hans, Musterstraße 22, 59555 Lippstadt
--- Befreundet mit: ---
* Thiemann, Uwe
* Halek, Gaby
* Löffelmann, Gabriele

```

Kopie nach Änderung im Original:

```

=====
Mustermann, Hans, Musterstraße 22, 59555 Lippstadt
--- Befreundet mit: ---
* Thiemann, Uwe
* Halek, Gaby
* Löffelmann, Gabriele

```

## Serialisieren von Objekten mit Zirkelverweisen

So genannte Zirkelverweise bereiten Speicheralgorithmen erfahrungsgemäß die größten Schwierigkeiten. Zirkelverweise kennen Sie vielleicht aus der Tabellenkalkulation. Sie treten dann auf, wenn beispielsweise Zelle A auf Zelle B, diese auf Zelle C, und diese wieder auf Zelle A verweisen soll. Was bei Tabellenkalkulationen grundsätzlich verboten ist, gestattet das Framework mit Objektreferenzen

dagegen sehr wohl. Und – um bei unserem bisherigen Beispiel zu bleiben – im Szenario des Programms aus dem letzten Abschnitt könnte das sogar recht schnell passieren, denn: Es ist nicht nur denkbar, sondern wahrscheinlich, dass Person A Person B zu seinem Freundeskreis zählt und umgekehrt.

Welche Auswirkungen Zirkelverweise normalerweise haben, zeigt das folgende Beispiel.

---

**BEGLEITDATEIEN:** Sie finden dieses Beispiel unter [.\\VB 2005 - Entwicklerbuch\\E - Datentypen\\Kap22\\Zirkelverweise](#).

---

In diesem Beispielprogramm sind im Gegensatz zum vorherigen die folgenden Änderungen vorgenommen worden, was die Adresse-Klasse betrifft. Die `Tostring`-Funktion benutzt selbst nicht mehr die Kurzform der Adressen für die Ausgabe der `BefreundetMit`-Eigenschaft, sondern erzeugt den Ausgabe-String ebenfalls mit der `Tostring`-Funktion, die allerdings eine kleine Änderung erfahren hat. Welche das ist, sehen Sie, wenn Sie das Programm starten und sich anschließend das Ergebnis anschauen:

Erste Adresse:

=====

Halek, Gaby, Musterstraße 24, 32132 Buchhausen

--- Befreundet mit: ---

Raubein, Petra, Autorenstr. 12, 32132 Buchhausen

Thiemann, Uwe, Autorenstr. 22, 32132 Buchhausen

--- Befreundet mit: ---

Koch, Manuela, Autorenstr. 22, 32132 Buchhausen

Zweite Adresse:

=====

Thiemann, Uwe, Autorenstr. 22, 32132 Buchhausen

--- Befreundet mit: ---

Koch, Manuela, Autorenstr. 22, 32132 Buchhausen

Das Programm, das diese Ausgabe hervorgebracht hat, sieht folgendermaßen aus:

```
Module mdlMain
    Sub Main()
        Dim locErsteAdr As New Adresse("Gaby", "Halek", "Musterstraße 24", "32132 Buchhausen")
        Dim locZweiteAdr As New Adresse("Uwe", "Thiemann", "Autorenstr. 22", "32132 Buchhausen")
        locErsteAdr.BefreundetMit.Add(New Adresse("Petra", "Raubein", "Autorenstr. 12", "32132 Buchhausen"))
        locErsteAdr.BefreundetMit.Add(locZweiteAdr)
        locZweiteAdr.BefreundetMit.Add(New Adresse("Manuela", "Koch", "Autorenstr. 22", "32132 Buchhausen"))
        'Wenn Sie diese Zeile einbauen, erstellen Sie einen Zirkelverweis.
        ''locZweiteAdr.BefreundetMit.Add(locErsteAdr)

        'Originaladresse ausgeben.
        Console.WriteLine("Erste Adresse:")
        Console.WriteLine("=====")
        Console.WriteLine(locErsteAdr)

        Console.WriteLine("Zweite Adresse:")
        Console.WriteLine("=====")
```

```

Console.WriteLine(locZweiteAdr)

'Kopie anlegen.
Dim locAdrKopie As Adresse = CType(ADObjectCloner.DeepCopy(locErsteAdr), Adresse)
Console.ReadLine()

End Sub
End Module

```

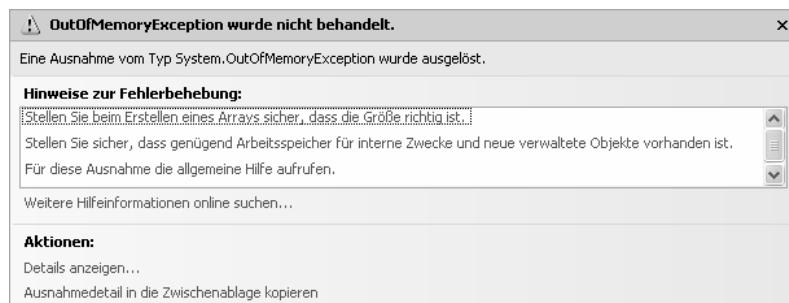
Sie erkennen am Ergebnis und dem Programmtext, dass das Programm auch verschachtelte Daten in der Eigenschaft `BefreundetMit` (und der dahinter steckenden `ArrayList`) bei der Ausgabe berücksichtigt. Wenn es die Liste der befreundeten Personen erstellt, und eine Person dieser Liste wieder Personeneinträge unter `BefreundetMit` führt, werden diese bei der Ausgabe ebenfalls berücksichtigt.

Eine solche Vorgehensweise kann für ein Programm allerdings wirklich fatale Folgen haben. Denn wenn eine der Personen der `BefreundetMit`-Liste in ihrer Liste die Person führt, die ihr ebenfalls zugeordnet ist (Person A ist befreundet mit Person B, und Person B ist ebenfalls befreundet mit Person A – was ja durchaus nichts Ungewöhnliches ist), dann haben Sie es mit einem Zirkelverweis zu tun.

Der `ToString`-Algorithmus versagt in diesem Fall, weil er sich durch den Zirkelverweis immer wieder selbst aufruft. Sie können dieses Verhalten testen, indem Sie die Auskommentierung der im Listing fett gesetzten Zeile aufheben und den Auflösungsversuch eines Zirkelverweises initiieren. Wenn Sie das Programm anschließend starten, erscheint nach einer Weile im Ausgabefenster die Meldung

Eine Ausnahme (erste Chance) des Typs "System.OutOfMemoryException" ist in `mscorlib.dll` aufgetreten.

Das Programm »hängt« eine Weile, und je nachdem, wie gut oder schlecht Ihr System mit der nur noch ungenügend zur Verfügung stehenden Menge an Speicher klarkommt, sehen Sie kurz darauf folgende Ausnahme:



**Abbildung 22.5:** Durch den Zirkelverweis versagt `ToString` der Beispielklasse, da es sich immer wieder selbst aufruft und dabei allen verfügbaren Speicher verbraucht

Viele Serialisierungsalgorithmen versagen ebenfalls durch eine solche Konstellation der Datenklassen. Dass die `DeepCopy`-Methode jedoch funktioniert und der .NET-Serializer keine Fehlermeldung auslöst, können Sie ganz einfach testen, indem Sie alle `Console.WriteLine`-Befehle auskommentieren. Das Programm passiert nach einem erneuten Start die Zeile

```

Dim locAdrKopie As Adresse = CType(ADObjectCloner.DeepCopy(locErsteAdr), Adresse)

```

anstandslos, beendet anschließend ordnungsgemäß und beweist so, dass der Serialisierungsalgorithmus des Frameworks über Zirkelverweise erhaben ist.

## **Serialisierung von Objekten unterschiedlicher Versionen beim Einsatz von BinaryFormatter oder SoapFormatter-Klassen**

Einer sehr wichtigen Sache sollten Sie sich bewusst sein: Wenn Sie zur Serialisierung BinaryFormatter oder SoapFormatter verwenden, um Objekte in Dateien zu serialisieren, dann können Sie bei einem Update eines Programms leicht Gefahr laufen, in Versionskonflikte zu geraten.

Angenommen, Sie verfügen über eine Datenebene in Ihrer Applikation, die durch ein Objekt repräsentiert wird. Dieses Objekt kapselt alle Daten Ihrer Anwendung. Da Sie dem Anwender natürlich die Möglichkeit geben müssen, die Daten als Datei auf einem Datenträger zu speichern, nutzen Sie dafür die Serialisierungsfunktionen von .NET.

Nun erweitern Sie durch ein Update Ihr Programm, und es kommen einige Eigenschaften zur Datenklasse hinzu. Da sich die Deserialisierungsfunktionen beider Serialisierungsklassen am Objekt orientieren, erwartet es die Daten zu einigen Eigenschaften, die es im alten Objektmodell natürlich nicht gegeben hat. Der Deserialisierungsversuch schlägt in diesem Fall mit einer Ausnahme fehl.

Eine Möglichkeit, das zu verhindern, besteht durch den Einsatz der so genannten **Serialization-Binder-Klasse**, die für die Lösung des Versionsproblems konzipiert wurde. Mehr über dieses Objekt erfahren Sie in der Visual Studio-Online-Hilfe.

Eine andere Möglichkeit, das Versionsproblem in den Griff zu bekommen, ist die Anwendung der XML-Serialisierung, die im folgenden Abschnitt besprochen wird.

## **XML-Serialisierung**

Die XML-Serialisierung erfolgt generell nach dem gleichen Prinzip, wie Sie es beim Einsatz von BinaryFormatter und SoapFormatter kennen gelernt haben. Die XML-Serialisierung hat im Gegensatz dazu entscheidende Vorteile:

- Das Problem mit der Serialisierung von Klassen unterschiedlicher Versionen betrifft sie nicht, denn es werden nur die Daten deserialisiert, die einerseits vorhanden sind, andererseits auch in den zu deserialisierenden Klassen zur Verfügung stehen.
- XML-Code ist nicht nur leichter lesbar, sondern kann auch von den verschiedensten Programmen importiert und weiterverarbeitet werden.

Allerdings müssen Sie beim Erstellen von Klassen, die Sie im XML-Format serialisieren wollen, auch einige Kompromisse eingehen:

- Wie bei anderen Klassen, die Sie mit .NET-Serialisierungstechniken serialisieren, müssen die Klassen mit dem **Serializable-Attribut** gekennzeichnet werden.
- Eine Klasse, die im XML-Format serialisiert werden soll, muss über einen parameterlosen Konstruktur (`Sub New()`) verfügen.
- Nur Member-Variablen bzw. Eigenschaften, die öffentlichen Zugriff haben, können serialisiert werden.

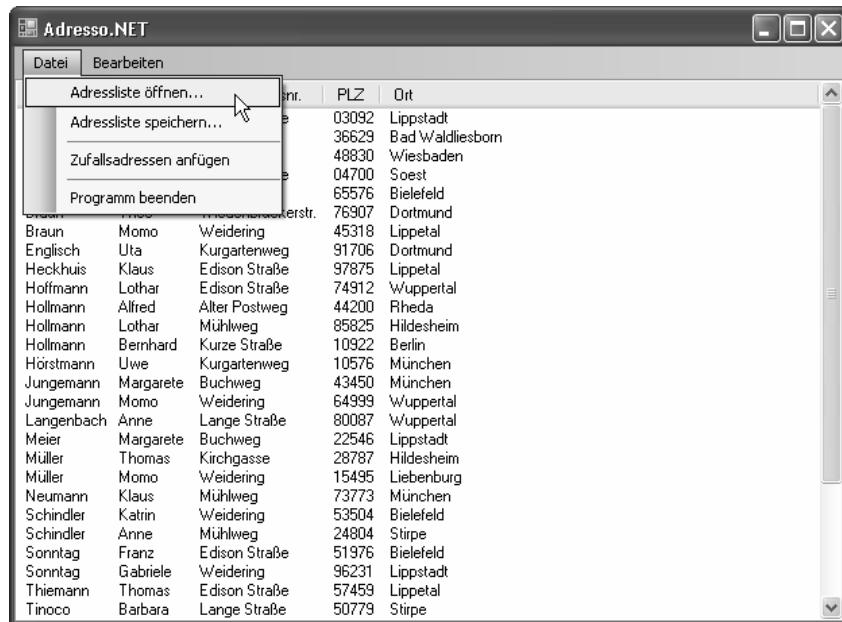
Wenn Ihre Klassen diese Voraussetzungen erfüllen, steht einer Serialisierung nichts mehr im Weg.

---

**BEGLEITDATEIEN:** Sie finden das folgende Beispielprojekt im Verzeichnis `.\VB 2005 - Entwicklerbuch\E – Datentypen\Kap22\XMLSerialization` unter dem Projektnamen `Adresso.sln`.

---

Dieses Beispielprojekt entspricht im Großen und Ganzen dem letzten Beispiel aus ► Kapitel 20. Es erlaubt das Generieren von Zufallsadressen und deren Darstellung in einer Liste. Im Gegensatz zum Beispiel aus Kapitel 20 verfügt es jedoch über eine richtige Menüstruktur und in dieser Ausbaustufe über ein paar zusätzliche Funktionen, die über dieses Menü erreichbar sind:



**Abbildung 22.6:** Die Adressenverwaltung »Adresso.Net« erlaubt es in dieser Ausbaustufe Zufallsadressen zu generieren und erstellte Listen in Dateien abzuspeichern und zu laden

Sie können eine beliebige Anzahl zufälliger Adressen erstellen, indem Sie den Menüpunkt *Zufallsadressen anfügen* wiederholt aufrufen. Mit *Adressliste speichern* erstellen Sie die so generierte Adressliste als Datei im XML-Format. Über den umgekehrten Weg können Sie eine Liste, die als XML-Datei vorliegt, über den Menüpunkt *Adressliste öffnen* in die entsprechenden Adresse-Objekte umwandeln, die dann wieder im ListView-Steuerelement des Programms angezeigt werden.

Neben der Darstellung der Liste und der Auswertung der Menüereignisse, die weitestgehend dem Beispiel aus ► Kapitel 20 entsprechen, und auf die ich aus diesem Grund an dieser Stelle nicht nochmals eingehen möchte, besteht der Code dieses Beispiels aus zwei zentralen Klassen.

Die Klasse Adresse kennen Sie aus vorherigen Beispielen bereits, und ihre Codebesonderheiten sind schnell erklärt:

```
<Serializable()
Public Class Adresse

    'Membervariablen, die die Daten halten:
    Protected myMatchcode As String
```

```

Protected myName As String
Protected myVorname As String
Protected myStraße As String
Protected myPLZ As String
Protected myOrt As String

''' <summary>
''' Erstellt eine neue Instanz dieser Klasse.
''' </summary>
''' <remarks>Ein parameterloser Konstruktor wird benötigt,
''' um XML-Serialisierung zu ermöglichen.</remarks>
Sub New()
    MyBase.New()
End Sub

'Konstruktor - legt eine neue Instanz an
Sub New(ByVal Matchcode As String, ByVal Name As String, ByVal Vorname As String, _
        ByVal Straße As String, ByVal Plz As String, ByVal Ort As String)
    myMatchcode = Matchcode
    myName = Name
    myVorname = Vorname
    myStraße = Straße
    myPLZ = Plz
    myOrt = Ort
End Sub

Public Overridable Property Matchcode() As String
    Get
        Return myMatchcode
    End Get
    Set(ByVal Value As String)
        myMatchcode = Value
    End Set
End Property

.
.   'Der Code der anderen Eigenschaften wurde aus Platzgründen ausgelassen.
.

End Class

```

Drei Dinge sind wichtig, damit die XML-Serialisierung mit Auflistungen von Elementen dieses Typs funktionieren:

- Das Attribut **Serializable** muss auf die Klasse angewendet werden (siehe oberste, fett hervorgehobene Listingzeile).
- Die Klasse muss über einen parameterlosen Konstruktor verfügen (darunter, ebenfalls in Festschrift).
- Alle Eigenschaften, die serialisiert werden sollen, müssen öffentlich zugänglich sein. Im Beispiellisting ist die **Matchcode**-Eigenschaft (die im Übrigen in der Formularliste nicht dargestellt wird) deswegen auch mit dem **Public**-Modifizierer ausgestattet.

Gespeichert werden die einzelnen Adresse-Elemente in einer Auflistung namens Adressen, die aus der generischen Klasse List(Of Adresse) abgeleitet wird (mehr zur generischen Auflistung erfahren Sie ebenfalls in ► Kapitel 20). Diese Klasse sieht folgendermaßen aus:

```
<Serializable()>
Public Class Adressen
    Inherits List(Of Adresse)

    ''' <summary>
    ''' Erstellt eine neue Instanz dieser Klasse.
    ''' </summary>
    ''' <remarks></remarks>
    Sub New()
        MyBase.New()
    End Sub

    ''' <summary>
    ''' Erstellt eine neue Instanz dieser Klasse
    ''' und ermöglicht das Übernehmen einer vorhandenen Auflistung in diese.
    ''' </summary>
    ''' <param name="collection">Die generische Adresse-Auflistung, deren Elemente
    ''' in diese Auflistungsinstanz übernommen werden sollen.</param>
    ''' <remarks></remarks>
    Sub New(ByVal collection As ICollection(Of Adresse))
        MyBase.New(collection)
    End Sub

    ''' <summary>
    ''' Serialisiert alle Elemente dieser Auflistung in eine XML-Datei.
    ''' </summary>
    ''' <param name="Dateiname">Der Dateiname der XML-Datei.</param>
    ''' <remarks></remarks>
    Sub XMLSerialize(ByVal Dateiname As String)
        Dim locXmlWriter As New XmlSerializer(GetType(Adressen))
        Dim locXmlDatei As New StreamWriter(Dateiname)
        locXmlWriter.Serialize(locXmlDatei, Me)
        locXmlDatei.Flush()
        locXmlDatei.Close()
    End Sub

    ''' <summary>
    ''' Generiert aus einer XML-Datei eine neue Instanz dieser Auflistungsklasse.
    ''' </summary>
    ''' <param name="Dateiname">Name der XML-Datei, aus der die Daten für
    ''' diese Auflistungsinstanz entnommen werden sollen.</param>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Public Shared Function XmlDeserialize(ByVal Dateiname As String) As Adressen
        Dim locXmlLeser As New XmlSerializer(GetType(Adressen))
        Dim locXmlDatei As New StreamReader(Dateiname)
        Return CType(locXmlLeser.Deserialize(locXmlDatei), Adressen)
    End Function
```

```

''' <summary>
''' Liefert eine Instanz dieser Klasse mit Zufallsadressen zurück.
''' </summary>
''' <param name="Anzahl">Anzahl der Zufallsadressen, die erzeugt werden sollen.</param>
''' <returns></returns>
''' <remarks></remarks>
Public Shared Function ZufallsAdressen(ByVal Anzahl As Integer) As List(Of Adresse)

    ' Ausgelassener Code; unwichtig an dieser Stelle
    ' End Function
End Class

```

Auch diese Klasse, die die Adresse-Elemente speichert, erfüllt die Voraussetzungen für die XML-Serialisierung – Sie verfügt über das notwendige Attribut, über einen parameterlosen Konstruktor und über öffentliche Eigenschaften. Die einzige öffentliche Eigenschaft, auf die es bei dieser Klasse ankommt, ist übrigens im Code nicht zu erkennen: Es ist Items, die durch die Basisklasse List(Of Adresse) definiert wurde und per Vererbung natürlich auch in der neuen Adressen-Klasse zur Verfügung steht. Sie erlaubt dem XML-Serialisierer Zugriff auf die einzelnen Adresse-Elemente und damit letzten Endes auf jedes einzelne Datenfeld von Adresse.

Beim ersten Blick auf den Code dieser Klasse mag es vielleicht verwirren, dass die Methode zum Serialisieren des Klasseninhalts eine nicht statische, die zum Deserialisieren hingegen eine statische ist. Doch das ist klar: Beim Deserialisieren gibt es noch keine Instanz der Adressen-Klasse; also gibt es auch keine Member-Methode die aufgerufen werden könnte, denn die Instanz mit den Adresse-Elementen geht ja erst durch den Deserialisierungsprozess hervor!

Anders ist es beim Serialisieren: Hier sind die einzelnen Elemente bereit in der Instanz von Adressen vorhanden, und aus diesem Grund kann es auch eine Member-Methode sein, die das Serialisieren vornimmt; sie greift auf ihre eigenen Elemente zu.

Das Anwenden dieser beiden Methoden ist schließlich eine Kleinigkeit. Die beiden folgenden Ereignisbehandlungsmethoden, die das Auswählen der entsprechenden Menüeinträge auswerten, sind dafür verantwortlich (relevante Codeteile sind wieder fett hervorgehoben):

```

'Wird aufgerufen, wenn Anwender Datei/Adresslist öffnen wählt.
Private Sub tsmAdresslisteLaden_Click(ByVal sender As System.Object,
                                         ByVal e As System.EventArgs) Handles tsmAdresslisteLaden.Click

    Dim dateiÖffnenDialog As New OpenFileDialog
    With dateiÖffnenDialog
        .CheckFileExists = True
        .CheckPathExists = True
        .DefaultExt = "*.xml"
        .Filter = "Adresse XML-Dateien" &
                  " (*.adrxm1 | *.adrxml | Alle Dateien (*.*)|.*"
    End With
    Dim dialogErgebnis As DialogResult = .ShowDialog
    If dialogErgebnis = Windows.Forms.DialogResult.Cancel Then
        Exit Sub
    End If

```

```

'Adressen deserialisieren
myAdressen = Adressen.XmlDeserialize(.FileName)

'Liste neu darstellen
ElementeDarstellen()
End With
End Sub

'Wird aufgerufen, wenn Anwender Datei/Adressliste speichern wählt.
Private Sub tsmAdresslisteSpeichern_Click(ByVal sender As System.Object,
                                         ByVal e As System.EventArgs) Handles tsmAdresslisteSpeichern.Click

    Dim dateiSpeichernDialog As New SaveFileDialog

    With dateiSpeichernDialog
        .CheckPathExists = True
        .DefaultExt = "*.xml"
        .Filter = "Adresse XML-Dateien" & " (" & "*.adxml" & ")" | " & _
                  "*.adxml" & "|Alle Dateien (*.*)|*.*"
        Dim dialogErgebnis As DialogResult = .ShowDialog
        If dialogErgebnis = Windows.Forms.DialogResult.Cancel Then
            Exit Sub
        End If

        'Adressen serialisieren
        myAdressen.XMLSerialize(.FileName)
    End With
End Sub
End Class

```

## Prüfen der Versionsunabhängigkeit der XML-Datei

Die XML-Datei, die dieses Programm generiert, sieht auszugsweise etwa wie in Abbildung 22.7 aus, und Sie können sich »Ihre« Version mit dem Notepad anschauen.

Wenn Sie ein Datenfeld aus dieser XML-Datei entfernen, etwa wie in Abbildung 22.7 zu sehen, und anschließend abspeichern, können Sie diese XML-Datei dennoch deserialisieren lassen – für die Ort-Eigenschaft des betroffenen Adresse-Objektes wird dann beim Deserialisieren eben keine Zuordnung vorgenommen. Der Deserialisierungsprozess schlägt aber nicht fehl.

```

<?xml version="1.0" encoding="utf-8"?>
<ArrayOfAdresse xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Adresse>
    <Matchcode>AdUt00000008</Matchcode>
    <Name>Ademmer</Name>
    <Vorname>Uta</Vorname>
    <Straße>Edison Straße</Straße>
    <PLZ>03092</PLZ>
    <Ort>Bad Waldliesborn</Ort>
  </Adresse>
  <Adresse>
    <Matchcode>AdMi00000011</Matchcode>
    <Name>Ademmer</Name>
    <Vorname>Michaela</Vorname>
    <Straße>Kurze Straße</Straße>
    <PLZ>36629</PLZ>
  </Adresse>
  <Adresse>
    <Matchcode>AlAx00000014</Matchcode>
    <Name>Albrecht</Name>
    <Vorname>Axel</Vorname>
    <Straße>Musterstraße</Straße>
    <PLZ>48830</PLZ>
  </Adresse>

```

**Abbildung 22.7:** Auch wenn Sie ein Datenfeld aus der XML-Datei entfernen, lässt sich die Datei anschließend noch deserialisieren

Genau so könnten Sie der Adresse-Klasse eine zusätzliche Eigenschaft hinzufügen. »Alte« Versionen der Serialisierungsdateien ließen sich dennoch weiterverwenden.

---

**TIPP:** Wenn Sie XML-Serialisierungslogik in eigenen Klassen implementieren wollen, nutzen Sie am besten die Codeausschnittsbibliothek (über die Sie in ► Kapitel 4 mehr erfahren). Sie finden die entsprechenden Codeausschnitte unter *XML/Klassendaten aus einer XML-Datei lesen* sowie *XML/Klassendatei in eine XML-Datei schreiben*.

---

## Serialisierungsfehler bei KeyedCollection

Die generische KeyedCollection-Klasse eignet sich als Auflistungsbasis für Geschäftsobjekte ideal, denn:

- Sie stellt, da sie generisch ist, eine typsichere Auflistung des jeweiligen Elementes dar.
- Objekte lassen sich wahlweise über einen Index *oder* über einen Schlüssel abrufen. Den letzteren muss das Objekt allerdings selber zur Verfügung stellen – ► Kapitel 20 liefert im entsprechenden Abschnitt mehr zu diesem Thema.
- Da die KeyedCollection auch über einen Indexer verfügt, stellt sie auch eine Enumerierungstechnik für For/Each zur Verfügung, obwohl Sie zusätzlich Wörterbuchfunktionalität implementiert.

An Flexibilität ist diese Klasse also in Sachen Auflistung kaum zu übertreffen.

Allerdings ist diese Kombination auch gerade dann eine Crux, wenn sich Schlüsseldatentyp und Indexer typtechnisch ins Gehege kommen, denn:

Stellen Sie sich vor, Sie haben eine Klasse wie die folgende, die den eindeutigen Schlüssel eines Geschäftsobjektes über eine ID zur Verfügung stellt (und dieser Anwendungsfall ist keinesfalls konstruiert, denn gerade in der Datenbankprogrammierung haben Tabellen oft den Datentyp Integer als primären Schlüssel und Hauptindex):

---

**BEGLEITDATEIEN:** Das Projekt zu diesem Beispiel finden Sie im Verzeichnis *.\VB 2005 - Entwicklerbuch\E - Datentypen\Kap22\KeyedCollection*.

---

```
Public Class Adresse

    Private myIDAdresse As Integer
    Private myNachname As String
    Private myVorname As String

    Sub New(ByVal IDAdresse As Integer, ByVal Nachname As String, ByVal Vorname As String)
        myIDAdresse = IDAdresse
        myNachname = Nachname
        myVorname = Vorname
    End Sub

    Public Property IDAdresse() As Integer
        Get
            Return myIDAdresse
        End Get
        Set(ByVal value As Integer)
            myIDAdresse = value
        End Set
    End Property

    Public Property Nachname() As String
        Get
            Return myNachname
        End Get
        Set(ByVal value As String)
            myNachname = value
        End Set
    End Property

    Public Property Vorname() As String
        Get
            Return myVorname
        End Get
        Set(ByVal value As String)
            myVorname = value
        End Set
    End Property
End Class
```

Dieses Geschäftsobjekt speichert also eine simple Adresse – wesentliche Eigenschaften sind hier aus Platzgründen ausgelassen.

Was viel wichtiger ist: Ein Geschäftsobjekt dieses Typs identifiziert sich eindeutig durch seine IDAdresse-Eigenschaft, und das Problem dabei ist: Diese ist vom Typ Integer. Warum das ein Problem darstellt, zeigt die Implementierung einer KeyedCollection-Auflistung auf Basis dieser Klasse, denn es bietet sich mehr als an, dass IDAdresse auch als Quelle für die Schaffung des eindeutigen Schlüssels verwendet wird:

```
Public Class Adressen
    Inherits System.Collections.ObjectModel.KeyedCollection(Of Integer, Adresse)

    Protected Overrides Function GetKeyForItem(ByVal item As Adresse) As Integer
        Return item.IDAdresse
    End Function
End Class
```

Während der Aufbau einer KeyedCollection in dieser Struktur noch ohne Probleme möglich ist,

```
Protected Overrides Sub OnLoad(ByVal e As System.EventArgs)
    MyBase.OnLoad(e)
    myAdressen.Add(New Adresse(2, "Löffelmann", "Klaus"))
    myAdressen.Add(New Adresse(4, "Heckhuis", "Jürgen"))
    myAdressen.Add(New Adresse(6, "Sonntag", "Miriam"))
    myAdressen.Add(New Adresse(8, "Heckhuis", "Jürgen"))
    myAdressen.Add(New Adresse(10, "Vielstedde", "Anja"))
End Sub
```

zeigt sich beim Abrufen der Elemente bereits eine Besonderheit, wenn es sich beim Schlüssel um den Datentyp Integer handelt:

```
Private Sub btnObjekteAusgeben_Click(ByVal sender As Object, ByVal e As EventArgs)
    Dim count As Integer = 0
    For count = 0 To 4
        Debug.Print(myAdressen(count))
    Next

```

Item (key As Integer) As KeyedCollection.Adresse  
key:  
Der Schlüssel des abzurufenden Elements.

**Abbildung 22.8:** Handelt es sich um einen Integer-Schlüssel, bietet die KeyedCollection-Auflistung nur eine Signatur zum Abrufen der Elemente an

Soweit macht das auch Sinn. Da Indexer und Schlüssel in diesem Fall vom gleichen Datentyp sind, könnten weder Compiler noch Framework unterscheiden, welche »Version« der Abruffunktion sie verwenden sollen – die durch den Indexer oder die über den Schlüssel. Einzig und allein die Methodensignatur (also welche Parametertypen übergeben werden) entscheidet nämlich darüber, und da es in diesem Fall die gleichen Typen sind, kann keine Signaturenunterscheidung stattfinden. Ergo: Es gibt nur eine Möglichkeit, an das Element heranzukommen.

Anders wäre das, wenn unser Schlüssel beispielsweise vom Datentyp Long wäre.<sup>2</sup> In diesem Fall würde der Indexer über einen Integer, der Schlüssel über den Datentyp Long angesprochen werden, und dementsprechend würde Ihnen schon IntelliSense zwei Möglichkeiten zum Abrufen der Elemente anbieten:

---

<sup>2</sup> Lesen Sie aber bitte diesen Abschnitt zunächst zu Ende, bevor Sie jetzt schon alle Ihre KeyedCollection-basierenden Auflistungsklassen auf den Datentyp Long als Schlüssel umstellen!

```

Private Sub btnObjekteAusgeben_Click(ByVal sender As
    For count As Integer = 0 To 4
        debug.Print(myAdressen())
    Next
End Sub

```

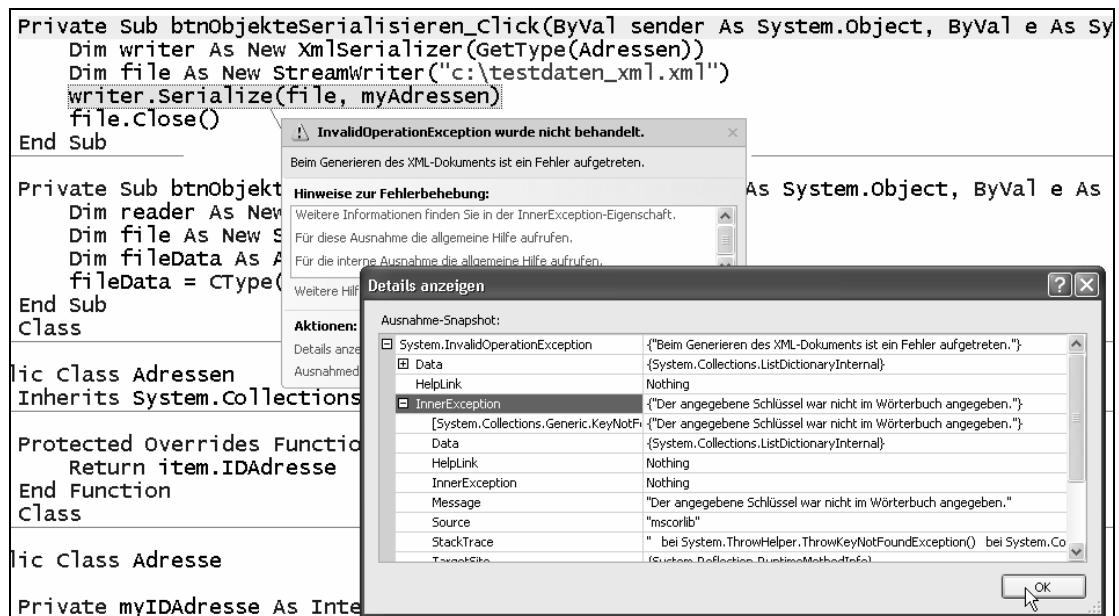
▲ 1 von 2 Item (**key As Long**) As KeyedCollection.Adresse  
**key:** Der Schlüssel des abzurufenden Elements.

▲ 2 von 2 Item (**index As Integer**) As KeyedCollection.Adresse  
**index:** Der nullbasierte Index des Elements, das abgerufen oder festgelegt werden soll.

**Abbildung 22.9:** Unterscheiden sich Schlüssel und Indexer beim Datentyp, gibt es bei der *KeyedCollection* tatsächlich zwei Möglichkeiten, an ein einzelnes Element heranzukommen

Diese Integer/Integer-Problematik wäre noch nicht weiter tragisch, denn durch den von *KeyedCollection* implementierten Enumerator gäbe es immer noch die Möglichkeit, die Objekte einerseits per Schlüssel, andererseits nacheinander abzurufen. Für letztere Methode würde Sie einfach eine For/Each-Schleife verwenden.

Richtig problematisch wird es aber, wenn Sie versuchen, die Daten zu serialisieren. Denn dann passiert Folgendes:



**Abbildung 22.10:** Beim Serialisieren stellen sich massive Probleme ein, wenn Schlüsseltyp und Indexer gleichermaßen vom Typ *Integer* sind!

Der Serialisierungsalgorithmus kommt hier offensichtlich nicht mehr mit der Doppelbelegung der Datentypen zurecht. Offensichtlich versucht er über eine Zählvariable der Reihe nach auf die Objekte zuzugreifen, die aber der Indexer von *KeyedCollection* fälschlicherweise als Schlüssel interpretiert. Da in unserem Beispiel die IDs nicht der Nummerierungsreihenfolge der Objekte entsprechen, schlägt dieser Versuch fehl und das Programm steigt mit einer Ausnahme, wie in der Abbildung zu sehen, aus.

## Workaround

Abhilfe können Sie bei diesem Szenario auf folgende Weise schaffen:

Sie schreiben sich eine Helper-Klasse, die nur zur Signaturenunterscheidung dient, die aber nichts weiter macht, als den Integer-Datentyp zu kapseln. Diese könnte beispielsweise folgendermaßen ausschauen:

```
<Serializable()>
Public Class IntKey

    Private myKey As Integer

    'Wichtig für die XML-Serialisierung: Parameterloser Konstruktor!
    Sub New()
        MyBase.new()
    End Sub

    Sub New(ByVal Key As Integer)
        myKey = Key
    End Sub

    Public ReadOnly Property Key() As Integer
        Get
            Return myKey
        End Get
    End Property
End Class
```

---

**HINWEIS:** Auf den ersten Blick könnte man meinen, dass auch die Verwendung des Long-Datentyps als Schlüssel ausreichend sein müsste, um das Problem in den Griff zu bekommen. Ich persönlich wäre dabei aber vorsichtig, da durch die implizite Konvertierungsmöglichkeit von Integer in Long unter Umständen dieselben Effekte zu Tage treten könnten. Mit dem hier vorgestellten Workaround funktioniert es jedoch einwandfrei.

---

Die geänderte Eigenschaft in der Adresse-Klasse sieht nun wie folgt aus:

```
Public Class Adresse

    Private myIDAdresse As IntKey
    Private myNachname As String
    Private myVorname As String

    'Den brauchen wir fürs Deserialisieren!
    Sub New()
        MyBase.new()
    End Sub

    Sub New(ByVal IDAdresse As IntKey, ByVal Nachname As String, ByVal Vorname As String)
        myIDAdresse = IDAdresse
        myNachname = Nachname
        myVorname = Vorname
    End Sub
```

```

Public Property IDAdresse() As IntKey
    Get
        Return myIDAdresse
    End Get
    Set(ByVal value As IntKey)
        myIDAdresse = value
    End Set
End Property
.
.
.
```

Und dementsprechend hat sich auch die KeyedCollection geändert:

```

Public Class Adressen
    Inherits System.Collections.ObjectModel.KeyedCollection(Of IntKey, Adresse)

    Protected Overrides Function GetKeyForItem(ByVal item As Adresse) As IntKey
        Return item.IDAdresse
    End Function
End Class
```

Das Hinzufügen von Elementen zur Auflistung bereitet zwar ein kleines bisschen mehr Aufwand:

```

Protected Overrides Sub OnLoad(ByVal e As System.EventArgs)
    MyBase.OnLoad(e)
    myAdressen.Add(New Adresse(New IntKey(2), "Löffelmann", "Klaus"))
    myAdressen.Add(New Adresse(New IntKey(4), "Heckhuis", "Jürgen"))
    myAdressen.Add(New Adresse(New IntKey(6), "Sonntag", "Miriam"))
    myAdressen.Add(New Adresse(New IntKey(8), "Heckhuis", "Jürgen"))
    myAdressen.Add(New Adresse(New IntKey(10), "Vielstedde", "Anja"))
End Sub
```

Aber dafür können Sie die einzelnen Elemente nunmehr zielsicher und ohne Zweideutigkeiten sowohl über den Indexer (siehe Grafik) als auch über den Schlüssel abrufen:

```

Private Sub btnObjekteAusgeben_Click(ByVal sender As System.Object,
    For count As Integer = 0 To 4
        Debug.Print(myAdressen(count).Nachname & ", " & myAdressen(
    Next
End Sub
```

▲ 2 von 2 ▾ Item (index As Integer) As KeyedCollection.Adresse  
**index:** Der nullbasierte Index des Elements, das abgerufen oder festgelegt werden soll.

Abbildung 22.11: Mit der Workaround-Lösung funktioniert nun der Abruf über Schlüssel und Indexer perfekt

---

**BEISPIELDATEIEN:** Eine Version des Programm, die dieses Verfahren nutzt, finden Sie im Verzeichnis .\VB 2005 - Entwicklerbuch\E - Datentypen\Kap22\KeyedCollection2\, und Sie werden sehen, dass es auch beim Serialisieren und Deserialisieren einwandfrei funktioniert.

---



# 23 Attribute und Reflection

---

- 
- |            |   |
|------------|---|
| <b>688</b> | <b>Genereller Umgang mit Attributen</b>   |
| <b>691</b> | <b>Einführung in Reflection</b>   |
| <b>697</b> | <b>Erstellung benutzerdefinierter Attribute und deren Erkennen zur Laufzeit</b> |
- 

Wenn Sie größere Anwendungen entwickeln, kommen Sie an Attributen nicht vorbei. Im Laufe dieses Buches konnten Sie diese Tatsache schon an zahlreichen Stellen feststellen. Attribute sind besondere Klassen, die keine spezielle Funktion ausführen, sondern nur ein Hinweis für eine andere Instanz sind, ein bestimmtes Element Ihrer Anwendung auf besondere Weise zu behandeln.

Sie können nicht nur auf die im Framework vorhandenen Attribute zurückgreifen, sondern auch eigene Attributklassen entwerfen. In diesem Fall müssen Sie allerdings auch Techniken beherrschen, die das Erkennen von Attributen zur Laufzeit ermöglichen – und an dieser Stelle kommt Reflection ins Spiel.

Die Reflection-Techniken im Framework stellen im Prinzip den Psychologen Ihres Programms dar, und sie helfen Ihnen, Informationen über bestimmte Assemblies, Klassen, Methoden, Eigenschaften und weitere Elemente zur Laufzeit Ihres Programms zu erhalten. Zu diesen Elementen gehören auch Attribute. Wenn Sie herausfinden wollen, ob eine bestimmte Methode einer Klasse oder Klasseninstanz beispielsweise mit einem bestimmten Attribut ausgestattet ist, verwenden Sie die Techniken der Reflection, um diese Attribute zu ermitteln. Gerade benutzerdefinierte Attribute und Reflection sind also zwei Themenbereiche, die eng miteinander verknüpft sind, und aus diesem Grund finden Sie diese beiden Themen auch als ein einziges Kapitel an dieser Stelle.

Damit klarer wird, welche enormen Möglichkeiten Ihnen Attribute und Reflection bieten können, finden Sie im Folgenden ein paar praktische Beispiele:

- Sie möchten, dass eine bestimmte Klasse Ihrer Datenbankanwendung automatisch die vorhandenen Datenbankfelder synchronisiert: Bestimmte Klassen sollen Tabellen darstellen, die Eigenschaften dieser Klassen die Datenfelder. Durch Attribute hätten Sie die Möglichkeit, diese Klassen zu kennzeichnen und die Datenbankdatei zur Laufzeit zu synchronisieren.
- Formulare könnten sich in Abhängigkeit bestimmter Klassen selber erstellen und die Eingabe bzw. Änderung einer Klasseninstanz übernehmen.
- Eine Ableitung des *ListView*-Steuerelements könnte ein Array mit Instanzen besonders gekennzeichneter Klassen automatisch anzeigen. Attribute würden bestimmen, welche Eigenschaften eines Array-Elementes als Spalte verwendet würden. Die Reihenfolge der Spalten ließe sich durch

weitere Attribute bestimmen. Dieses Beispiel finden Sie übrigens am Ende dieses Kapitels als Steuerelement realisiert.

Sie sehen: Beispiele gibt es viele, und vielleicht ahnen Sie schon, welch mächtiges Werkzeug Ihnen das Framework mit Attributen und Reflection in die Hände legt.

## Genereller Umgang mit Attributen

Wie in der Einführung schon kurz angerissen, und wie Sie es in den zahlreichen vergangenen Beispielprogrammen schon zigfach gesehen haben, werden Attribute ganz anders als herkömmliche Klassen verwendet (wenn sie auch auf dieselbe Weise erstellt werden). Attribute statthen Klassen oder Prozeduren mit besonderen Eigenschaften aus. Genauso, wie die Verwendung von **Fettschrift innerhalb eines Absatzes dieses Buches** selbst nicht den Sinn des Geschriebenen verändert, so erfüllt das Attribut Fettschrift dennoch seinen Zweck: Es ist der Hinweis für Sie, die so markierte Textstelle aufmerksamer zu lesen.

Attribute »markieren« eine Klasse oder eine Prozedur in Visual Basic, indem sie in Kleiner-/Größerzeichen eingeschlossen vor die Klassen- bzw. Prozedurdefinition gesetzt werden, etwa:

```
<MeinAttribute(MöglicherParameter)> Class MeineKlasse
```

```
End Class
```

Da Zeilen, in denen Attribute verwendet werden, auf diese Weise unnötig lang und damit schlecht lesbar sind, verwendet man in Visual Basic üblicherweise das *Underscore*-Zeichen (»\_«), um die Zeile zu umbrechen:

```
<MeinAttribute(MöglicherParameter)> _
Class MeineKlasse
```

```
End Class
```

---

**HINWEIS:** Achten Sie bei der Verwendung des Umbruchzeichens darauf, dass vor dem Umbruchzeichen ein Leerzeichen platziert wird!

---

Die Auswirkungen, die ein Attribut auf etwas hat, werden nicht durch die Attribut-Klasse gesteuert, sondern ausschließlich von den Instanzen, die die Elemente unter die Lupe nehmen, die mit Attributen versehen sind. Attributklassen sind in der Regel Klassen, die keine wirkliche Funktionalität zur Verfügung stellen; sie dienen lediglich als Container für Informationen. Behalten Sie diese wichtige Aussage im Hinterkopf, wenn Sie den Einsatz von Attributen planen.

---

**WICHTIG:** Attributklassen enden grundsätzlich auf den Namen ... *Attribute*. Dennoch müssen Sie den Namenszusatz *Attribute* nicht mit angeben, um eine *Attribute*-basierende Klasse für Kennzeichnungszwecke zu verwenden.

---

## Einsatz von Attributen am Beispiel von **ObsoleteAttribute**

Ein Beispiel soll das verdeutlichen: Die **ObsoleteAttribute**-Klasse dient beispielsweise dazu, eine Prozedur oder Klasse als »überholt« (versionstechnisch) zu kennzeichnen. Gesetzt den Fall, Sie haben vor einem Jahr eine Klassenbibliothek für .NET entwickelt, die Sie Ihren Kunden nun in

erweiterter und überarbeiteter Form zugänglich machen wollen. Im Rahmen der Umbauarbeiten haben Sie festgestellt, dass Sie bestimmte Funktionen nicht mehr benötigen, weil Ihre Klassenbibliothek entweder sehr viel automatisierter arbeiten kann oder es Sinn ergibt, bestimmte Funktionen durch neuere Funktionen zu ersetzen, da diese viel effizienter arbeiten.

Natürlich können Sie in der neuen Version die alten, überflüssigen Funktionen nicht einfach entfernen. Würde der Anwender Ihrer Klassenbibliothek nämlich anschließend sein Programm mit der neuen Version kompilieren, wären Fehlfunktionen vorprogrammiert (das Programm ließe sich wahrscheinlich gar nicht erst kompilieren). Mit Hilfe des Obsolete-Attributes können Sie den Entwickler aber gefahrlos darauf aufmerksam machen, eine bestimmte Funktion nicht mehr zu verwenden. Sie setzen in diesem Fall das ObsoleteAttribute vor die entsprechende Klasse/Prozedur und geben zusätzlich eine Hinweismeldung an.

---

**BEGLEITDATEIEN:** Sie finden das folgende Beispielprojekt unter .\VB 2005 - Entwicklerbuch\E - Datentypen\Kap23\AttributesDemo.

---

Bei diesem Beispielprogramm kommt es gar nicht darauf an, das Programm zu starten. Wenn Sie es geladen haben, betrachten Sie vielmehr den Quellcode und wie die Aufgabenliste auf die Verwendung einer bestimmten Eigenschaft reagiert:

```
Module mdlMain

    Sub Main()

        Dim locTestKlasse As New TestKlasse("Dies ist ein Test")
        Console.WriteLine(locTestKlasse.AlteEigenschaft)
        Console.WriteLine(locTestKlasse.NeueEigenschaft)
        Console.ReadLine()

    End Sub
End Module

Public Class TestKlasse
    Dim myEigenschaft As String

    Sub New(ByVal einString As String)
        myEigenschaft = einString
    End Sub

    'Alte Version. Diese Eigenschaft ist obsolet.
    <Obsolete("Sie sollten die AlteEigenschaft-Eigenschaft nicht mehr verwenden. " + _
              "Verwenden Sie stattdessen NeueEigenschaft")> _
    Property AlteEigenschaft() As String
        Get
            Return myEigenschaft
        End Get
        Set(ByVal Value As String)
            myEigenschaft = Value
        End Set
    End Property
```

```

Property NeueEigenschaft() As String
    Get
        Return myEigenschaft
    End Get
    Set(ByVal Value As String)
        myEigenschaft = Value
    End Set
End Property
End Class

```

Die Eigenschaft `AlteEigenschaft` der Klasse `TestKlasse` ist hier im Beispiel mit dem `Obsolete`-Attribut ausgestattet. Der Visual Basic Compiler erkennt dieses Attribut und zeigt in der Aufgabenliste eine Warnung, die den Entwickler auf diese Tatsache hinweist. Die `ObsoleteAttribute`-Klasse selbst hat aber nichts mit der Ausgabe des Textes zu tun, außer, dass sie den Text, der ausgegeben werden soll, zur Verfügung stellt. Die eigentliche Ausgabe des Textes erfolgt vom Visual Basic Compiler bzw. von der Entwicklungsumgebung.

---

**TIPP:** Obwohl die `ObsoleteAttribute`-Klasse im Beispiel zur Kennzeichnung der `AlteEigenschaft`-Eigenschaft verwendet worden ist, lässt sich die Eigenschaft verwenden und das Programm damit auch kompilieren und starten. Wenn Sie möchten, dass der Einsatz einer veralteten Eigenschaft zum Compiler-Fehler führt, der dafür sorgt, dass sich das Programm nicht mehr starten lässt, ändern Sie hinter dem Meldungsstring im `ObsoleteAttribute`-Konstruktor den zweiten (booleschen) Parameter in `True`, der die Kompilierung des Programms verhindert, das diese alte Version der Eigenschaft verwendet.

---

## Die speziell in Visual Basic verwendeten Attribute

Die wichtigsten Attribute haben Sie im Laufe der vergangenen Kapitel themenbezogen schon kennen gelernt. Es gibt allerdings einige spezielle Attribute für Visual Basic selbst,<sup>1</sup> die in der folgenden Tabelle zusammengefasst sind:

Attribut	Zweck
<code>COMClassAttribute</code> -Klasse	Weist den Compiler an, die Klasse als COM-Objekt anzuzeigen. Spezifisch für Visual Basic .NET.
<code>VBFixedStringAttribute</code> -Klasse	Gibt die Größe einer Zeichenfolge mit fester Länge in einer Struktur an, die mit Dateiein- und -ausgabefunktionen verwendet werden soll. Spezifisch für Visual Basic .NET.
<code>VBFixedArrayAttribute</code> -Klasse	Gibt die Größe eines festen Arrays in einer Struktur an, die mit Dateiein- und -ausgabefunktionen verwendet werden soll. Spezifisch für Visual Basic .NET.
<code>WebMethodAttribute</code> -Klasse	Ermöglicht das Aufrufen einer Methode mit dem SOAP-Protokoll. Wird in XML-Webdiensten verwendet.
<code>SerializableAttribute</code> -Klasse	Gibt an, dass eine Klasse serialisiert werden kann.

---

<sup>1</sup> Das bedeutet nicht, dass Sie nur diese Attribute in Visual Basic verwenden dürfen. Sie können natürlich alle Attribute des Frameworks auch in Ihren eigenen Visual-Basic-Anwendungen verwenden. Um eine Liste aller im Framework enthaltenen Attribute zu erhalten, rufen Sie die Online-Hilfe für die `Attribute`-Klasse auf und lassen sich anschließend alle abgeleiteten Klassen anzeigen.

Attribut	Zweck
MarshalAsAttribute-Klasse	Stellt fest, wie ein Parameter zwischen dem verwalteten Code von Visual Basic .NET und nicht verwaltetem Code z. B. einer Windows-API gemarshallt werden soll. Wird von der Common Language Runtime verwendet.
AttributeUsageAttribute-Klasse	Gibt die Verwendungsweise eines Attributs an.
DllImportAttribute-Klasse	Gibt an, dass die attributierte Methode als Export aus einer nicht verwalteten DLL implementiert ist.

**Tabelle 23.1:** Die speziellen Visual-Basic-Attribute

## Einführung in Reflection

Bevor wir im nächsten Schritt die beiden Themengebiete Reflection und Attribute miteinander verheiraten, lassen Sie uns zunächst einen Blick auf die Möglichkeiten der Reflection-Klassen selbst werfen.

Reflection selbst ist, wie am Anfang des Kapitels schon kurz zu erfahren war, der Oberbegriff für Techniken, die es einem Programm ermöglichen, etwas über Klassen zu erfahren, Klassen zu analysieren und auch Instanzen von Klassen programmtechnisch zu erstellen.

Natürlich ist es keine Kunst, eine Klasse programmtechnisch zu erstellen. Sie machen das immer, wenn Sie den Konstruktor einer Klasse verwenden. Aber darum geht es in diesem Fall auch gar nicht. Es geht darum, Klassen zu verwenden, von denen das Programm zum Zeitpunkt, an dem es gestartet wird, noch nichts weiß.

Angenommen, Sie möchten eine Funktion zur Verfügung stellen, die ein beliebiges Objekt als Parameter übernimmt und den Wert jeder einzelnen Eigenschaft auf dem Bildschirm ausgibt. Mit herkömmlichen Mitteln könnten Sie dieses Vorhaben nicht realisieren, denn: Da Sie im Vorfeld nicht wissen, welchen Objekttyp Sie zu erwarten haben, kennen Sie dessen Eigenschaftenamen auch nicht. Folglich können Sie die Werte dieser Eigenschaften auch nicht abrufen.

Sie müssen also Mittel und Wege finden, ein Objekt zu analysieren und zunächst zu ermitteln, über welche Eigenschaften es verfügt. Erst im zweiten Schritt können Sie, wenn Ihr Programm die Namen der Eigenschaften herausgefunden hat, die Inhalte der Eigenschaften herausfinden und schließlich auf dem Bildschirm ausgeben.

---

**BEGLEITDATEIEN:** Sie finden dieses Beispiel im Verzeichnis `.\VB 2005 - Entwicklerbuch\E - Datentypen\Kap23\Reflection01`.

---

Zentraler Bestandteil dieses Programms ist die `Adresse`-Klasse, die Sie aus vergangenen Kapiteln kennen. Die Main-Prozedur dieser Konsolenanwendung macht nichts weiter, als eine neue `Adresse`-Instanz zu erstellen und die Untersuchungsergebnisse im Konsolenfenster darzustellen, etwa wie im folgenden Bildschirmauszug zu sehen:

```
Attribute der Klasse:Reflection.Adresse
Standardattribute:
*AutoLayout, AnsiClass, NotPublic, Public, Serializable
```

Member-Liste:

```
*GetHashCode, Method
*Equals, Method
*ToString, Method
*get_Name, Method
*set_Name, Method
*get_ErfasstAm, Method
*get_ErfasstVon, Method
*get_BefreundetMit, Method
*set_BefreundetMit, Method
*get_Vorname, Method
*set_Vorname, Method
*get_Straße, Method
*set_Straße, Method
*get_PLZOrt, Method
*set_PLZOrt, Method
*ToStringShort, Method
*GetType, Method
*.ctor, Constructor
*Name, Property
Wert: Halek
*ErfasstAm, Property
Wert: 12.04.2004 09:06:27
*ErfasstVon, Property
Wert: Administrator
*BefreundetMit, Property
Wert: System.Collections.ArrayList
*Vorname, Property
Wert: Gaby
*Straße, Property
Wert: Buchstraße 223
*PLZOrt, Property
Wert: 32154 Autorhausen
```

Sie sehen, dass dieses Programm nicht nur in der Lage ist, eine (fast) vollständige Member-Liste der übergebenden Objekt-Instanz zu ermitteln; es kann darüber hinaus auch die Inhalte der Eigenschaften des Objektes anzeigen.

Bevor wir die Funktionsweise dieses Programms genauer unter die Lupe nehmen, lassen Sie mich zunächst ein paar Grundlagen zum Thema Typen und Reflection zwecks späteren besseren Verständnisses klären.

## Die Type-Klasse als Ausgangspunkt für alle Typenuntersuchungen

Den Schlüssel dazu bilden Klassen und Methoden, die Sie im *Reflection-Namespace* vom Framework finden. Ausgangspunkt für alle Reflection-Operationen bildet dabei die so genannte Type-Klasse, die den Typ einer Objektvariablen zur Laufzeit ermitteln kann.

Die Type-Klasse selbst können Sie nicht instanzieren, da sie eine abstrakte Klasse darstellt. Sie können sie aber verwenden, um den Typ eines bestehenden Objekts zu ermitteln und festzuhalten. Zwar gehört die Type-Klasse selbst nicht zum Reflection-Namensbereich – Sie benötigen sie aber, um mit weiteren Klassen dieses Namensbereichs und deren Funktionen Informationen über die jeweilige Typdeklaration eines zu untersuchenden Objektes abzurufen, z.B. Konstruktoren, Methoden, Felder, Eigenschaften und Ereignisse. Auch Informationen über Modul und Assembly, in denen die Klasse bereitgestellt wird, lassen sich anschließend mit diesen Funktionen ermitteln.

Jedes Objekt im Framework stellt eine GetType-Funktion zur Verfügung, mit der ihr zugrunde liegenden Type-Objekt ermittelt werden kann. Darüber hinaus können Sie mit der statischen Funktionsvariante auch ein Type-Objekt erstellen, wenn nur der voll qualifizierte Name des Typs bekannt ist. Mit TypeOf in Zusammenhang mit dem Is-Operator können Sie einen Typenvergleich durchführen. Alternativ funktioniert das auch mit GetType, welches auch als Operator eingesetzt werden kann. Die folgenden Codeauszüge demonstrieren den generellen Einsatz dieser Funktionen:

```
'Ein paar Type-Experimente:
Dim locTest As New Adresse("Klaus", "Löffelmann", "Urlaubsgasse 17", "59555 Lippende")
Dim locType1, locType2 As Type
locType1 = locTest.GetType

'Wichtig: TypeOf funktioniert nur zusammen mit dem Is-Operator:
If TypeOf locTest Is Reflection01.Adresse Then
    Console.WriteLine("Adresse-Typ erkannt!")
Else
    Console.WriteLine("Adresse-Typ nicht erkannt!")
End If

'Und auch das ist eine Alternative, die dasselbe bewirkt:
If locTest.GetType Is GetType(Reflection01.Adresse) Then
    Console.WriteLine("Adresse-Typ erkannt!")
Else
    Console.WriteLine("Adresse-Typ nicht erkannt!")
End If

'So funktioniert's, wenn zwei Typobjekte
'miteinander verglichen werden sollen:

locType2 = GetType(Adresse)           ' Der Normalfall, GetType als Operator
locType2 = GetType(Reflection01.Adresse) ' Alternativ: mit Namespace
locType2 = Type.GetType("Reflection01.Adresse") ' Alternativ: aus String

Console.WriteLine("Der Typ " + locType1.FullName +
    CStr(IIf(locType1 Is locType2, " entspricht ", " entspricht nicht ")) + _
    "dem Typ " + locType2.FullName)

Console.ReadLine()
```

Dieses Programm würde die folgende Ausgabe produzieren:

```
Der Typ ReflectionDemo.Adresse entspricht dem Typ ReflectionDemo.Adresse
Adresse-Typ erkannt!
```

---

**HINWEIS:** Dieser Codeschnipsel ist ebenfalls im besprochenen Beispielprojekt vorhanden (direkt im Anschluss an die `Main`-Prozedur, die mit `Exit Sub` endet). Möchten Sie mit ihm experimentieren, kommentieren Sie das `Exit Sub` einfach aus.

---

Wenn Sie auf diese Weise ein Type-Objekt ermittelt haben, können Sie mithilfe seiner Funktionen weitere Informationen über den entsprechenden Typen erhalten.

## Klassenanalysefunktionen, die ein Type-Objekt bereitstellt

Die folgende Tabelle gibt Ihnen einen Überblick über die wichtigsten Funktionen und Klassentypen, die von der Type-Klasse bereitgestellt werden und die dazu dienen, nähere Informationen zum entsprechenden Typ zu erhalten:

Funktion der Type-Klasse	Aufgabe
Assembly	Ruft die Assembly ab, in der der Typ definiert ist. Rückgabetyp: <code>System.Reflection.Assembly</code>
AssemblyQualifiedName	Ruft den voll qualifizierten Assembly-Namen ab. Rückgabetyp: <code>String</code>
Attributes	Ruft eine Bitkombination ( <i>Flags-Enum</i> ) ab, die Auskunft über alle nicht-benutzerdefinierten Attribute gibt. Rückgabetyp: <code>TypeAttributes</code>
BaseType	Ruft den Typ ab, von dem der angegebene Typ direkt vererbt wurde. Rückgabetyp: <code>Type</code>
FullName	Ruft den voll qualifizierten Namen des angegebenen Typs ab. Rückgabetyp: <code>String</code>
GetCustomAttributes	Ruft ein Array mit allen benutzerdefinierten Attributen ab. Wurden für den Typ keine benutzerdefinierten Attribute definiert, wird <i>Nothing</i> zurückgegeben. Rückgabetyp: <code>Object()</code>
GetEvent	Ruft das <i>EventInfo</i> -Objekt eines bekannten Ereignisses ab. Der Ereignisname wird als String übergeben. Rückgabetyp: <code>EventInfo</code>
GetEvents	Ruft eine Liste (als Array) mit allen Ereignissen des Objektes ab. Rückgabetyp: <code>EventInfo()</code>
GetField	Ruft das <i>FieldInfo</i> -Objekt eines bekannten öffentlichen Feldes ab. Der Feldname wird als String übergeben. Rückgabetyp: <code>FieldInfo</code>
GetFields	Ruft eine Liste (als Array) mit allen öffentlichen Feldern des Objektes ab. Rückgabetyp: <code>FieldInfo()</code>
GetMember	Ruft das <i>MemberInfo</i> -Objekt eines bekannten <i>Members</i> des Objektes an. Ein <i>Member</i> ist eine Oberkategorie eines Objektelementes, wie eine Eigenschaft, eine Methode, ein Ereignis oder ein Feld. Mit der <i>MemberType</i> -Eigenschaft eines <i>MemberInfo</i> -Objektes können Sie feststellen, um was für ein Objektelement es sich handelt. Rückgabetyp: <code>MemberInfo</code>
GetMembers	Ruft eine Liste (als Array) aller Elemente ( <i>Member</i> ) des Objektes ab. Rückgabetyp: <code>MemberInfo()</code>
GetProperty	Ruft das <i>PropertyInfo</i> -Objekt einer bekannten Eigenschaft ab. Der Eigenschaftenname wird als String übergeben. Rückgabetyp: <code> PropertyInfo</code>
GetProperties	Ruft eine Liste (als Array) aller Eigenschaften des Objektes ab. Rückgabetyp: <code> PropertyInfo()</code>

**Tabelle 23.2:** Die wichtigsten Funktionen, mit denen Sie Informationen über einen Typ abrufen können

Mit diesen Informationen können wir nun das Beispielprogramm betrachten. Es nutzt im Wesentlichen die GetMembers-Funktion des Type-Objektes, um die Informationen über ein beliebiges Objekt zu ermitteln:

```
Module mdlMain
    Sub Main()
        Dim locAdresse As New Adresse("Gaby", "Halek", "Buchstraße 223", "32154 Autorhausen")
        PrintObjectInfo(locAdresse)
        Console.ReadLine()
    End Sub

    Sub PrintObjectInfo(ByVal [Object] As Object)

        'Den Objekttypen ermitteln, um auf die Objektinhalte zuzugreifen.
        Dim locTypeInstanz As Type = [Object].GetType

        'Die nicht benutzerdefinierten Attribute ausgeben:
        Console.WriteLine("Attribute der Klasse:" + locTypeInstanz.FullName)
        Console.WriteLine("Standardattribute:")
        Console.WriteLine("    *" + locTypeInstanz.Attributes.ToString())
        Console.WriteLine()

        'Member und deren mögliche Attribute ermitteln.
        Dim locMembers() As MemberInfo
        locMembers = locTypeInstanz.GetMembers()
        Console.WriteLine("Member-Liste:")
        For Each locMember As MemberInfo In locMembers
            Console.WriteLine("    *" + locMember.Name + ", "
                + locMember.MemberType.ToString())
            If locMember.GetCustomAttributes(True).Length > 0 Then
                Console.WriteLine("        " + New String("-c", locMember.Name.Length))
            End If

            If locMember.MemberType = MemberTypes.Property Then
                Dim locPropertyInfo As PropertyInfo = CType(locMember, PropertyInfo)
                Console.WriteLine("            Wert: " + locPropertyInfo.GetValue([Object], Nothing).ToString())
            End If
        Next
    End Sub
End Module
```

---

**TIPP:** Wenn Sie alle Elemente eines Typs ermitteln wollen, verwenden Sie die Funktion GetMembers, wie hier im Beispiel gezeigt (erster in Fettschrift gesetzter Block). Sie können anschließend MemberType jedes einzelnen Elements überprüfen, um herauszufinden, um was für einen Elementtyp es sich genau handelt (Eigenschaft, Methode etc.).

---

**HINWEIS:** Beachten Sie auch, dass zu jeder Eigenschaft auch Methoden existieren. Framework-intern werden Eigenschaften wie solche behandelt, sie bekommen dann entweder das Präfix set\_ oder get\_ verpasst, um die Zugriffsart unterscheiden zu können. Aus diesem Grund finden Sie in der Elementliste, die Sie mit GetMembers ermitteln, drei Elemente für jede Eigenschaft (zwei Methoden, eine Eigenschaft – Voraussetzung dafür ist natürlich, dass es sich dabei nicht um eine Nur-Lesen- oder um eine Nur-Schreiben-Eigenschaft handelt).

---

## Objekthierarchie von MemberInfo und Casten in den spezifischen Info-Typ

MemberInfo ist die Basisklasse für alle spezifischeren Reflection-XXXInfo-Objekte. Von ihr abgeleitet sind:

- EventInfo zur Speicherung von Informationen über Ereignisse,
- FieldInfo zur Speicherung von Informationen über öffentliche Felder einer Klasse,
- MethodInfo zur Speicherung von Informationen über Methoden einer Klasse,
- PropertyInfo zur Speicherung von Informationen über die Eigenschaften einer Klasse.

Wenn Sie Informationen über eine Klasse oder ein Objekt mit der GetMembers-Funktion ermitteln, dann sind die spezifischeren Info-Objekte in den einzelnen MemberInfo-Objekten des MemberInfo-Arrays geboxt. Sie können sie mit einer CType oder DirectCast-Anweisung in den eigentlichen Infotyp zurückwandeln, um auf spezielle Eigenschaften des Infotyps zugreifen zu können, etwa mit:

```
If locMember.MemberType = MemberTypes.Property Then  
    Dim locPropertyInfo As PropertyInfo = CType(locMember, PropertyInfo)  
End If
```

## Ermitteln von Eigenschaftswerten über PropertyInfo zur Laufzeit

Wenn Sie auf die beschriebene Weise ein PropertyInfo-Objekt ermittelt haben, können Sie auch den eigentlichen Wert der Eigenschaft abrufen. Voraussetzung dafür ist, dass es einen entsprechenden Typ gibt, der zuvor instanziert wurde. Sie verwenden dazu die GetValue-Methode des PropertyInfo-Objektes. Das Beispielprogramm verwendet diese Vorgehensweise, um den aktuellen Inhalt einer Eigenschaft als Text anzuzeigen.

---

**TIPP:** Da jede Klasse über eine zumindest standardmäßig implementierte ToString-Funktion verfügt, ist die Ermittlung des Wertes als String sicher. Inwieweit ToString ein brauchbares Ergebnis zurückliefert, hängt natürlich von der Implementierung der ToString-Methode des jeweiligen Objektes ab.

---

Ein Beispiel für die Ermittlung von Eigenschaftsinhalten von Objekten zuvor nicht bekannten Typs finden Sie ebenfalls im Beispielprogramm:

```
If locMember.MemberType = MemberTypes.Property Then  
    Dim locPropertyInfo As PropertyInfo = CType(locMember, PropertyInfo)  
    Console.WriteLine("    Wert: " + locPropertyInfo.GetValue([Object], Nothing).ToString)  
End If
```

GetValue erfordert mindestens zwei Parameter: Der erste Parameter bestimmt, von welchem Objekt die angegebene Eigenschaft ermittelt werden soll. Da es Eigenschaften mit Parametern gibt, übergeben Sie im zweiten Parameter ein Object-Array, das diese Parameter enthält. Wenn die Eigenschaft parameterlos verwendet wird, übergeben Sie Nothing als zweiten Parameter, wie im Beispiel zu sehen.

# Erstellung benutzerdefinierter Attribute und deren Erkennen zur Laufzeit

Neben der manuellen Serialisierung von Daten ist das Erkennen und Reagieren auf benutzerdefinierte Attribute zur Laufzeit die wohl häufigste Anwendung von Reflection-Techniken. Attribute dienen, wie eingangs erwähnt, zur Kennzeichnung von Klassen oder Klassenelementen; Attributklassen erfüllen in der Regel aber keine weitere wirkliche Funktionalität, da nur in seltenen Fällen ihr Klassecode ausgeführt wird.

---

**BEGLEITDATEIEN:** Sie finden dieses Beispiel im Verzeichnis .\VB 2005 - Entwicklerbuch\E - Datentypen\Kap23\Reflection02).

---

Innerhalb der Codedatei *mdlMain.vb* finden Sie die Definition einer Attribute-Klasse, die folgendermaßen ausschaut.

```
'Benutzerdefiniertes Attribut erstellen.  
<AttributeUsage(AttributeTargets.A11)> Public Class MeinAttribute  
    Inherits Attribute  
    Private myName As String  
  
    Public Sub New(ByVal name As String)  
        myName = name  
    End Sub 'New  
  
    Public ReadOnly Property Name() As String  
        Get  
            Return myName  
        End Get  
    End Property  
End Class
```

Zwei Sachen fallen hier auf: Zum einen ist der Klassename nicht wirklich deutsch (Attribut wird im Deutschen nicht mit »e« am Ende geschrieben). Achten Sie aber darauf, wenn Sie selber Attribute-Klassen entwerfen, dass die Klassen grundsätzlich auf »... Attribute« enden. Der Entwickler, der das Attribut anschließend verwendet, kann in diesem Fall den verkürzten Namen verwenden. Anstelle von *MeinAttribute* reichte also die Angabe von *Mein*.

Die zweite Auffälligkeit: Eine Attribute-Klasse kann für den Gebrauch von nur bestimmten Elementen einer Klasse reglementiert werden, und dazu dient *AttributeUsageAttribute*. Sie bestimmen, wie im oben gezeigten Beispielcode, mit der *AttributeTargets-Enum*, für welche Elemente einer Klasse Ihre Attribute-Klasse zutreffend ist.

Wichtig ist auch, dass Sie eine benutzerdefinierte Attribute-Klasse mit *Inherits* von der Attribute-Basisklasse ableiten, damit die Reflection-Funktionen sie später überhaupt als Attribute ausmachen können.

Für das erweiterte Beispielprogramm kommt ebenfalls wieder die bekannte *Adresse-Klasse* zum Einsatz; allerdings verfügt sie an einigen Stellen über Attribute-Kennzeichnungen, und wir haben die neue *MeinAttribute-Klasse* dafür verwendet. Wie Sie im oben gezeigten Codeausschnitt sehen können, übernimmt die *MeinAttribute-Klasse* einen Parameter, der in diesem Beispiel nur informative Zwecke

hat. In der Adresse-Klasse nutzen wir den String-Parameter, um darüber zu informieren, in welchem Kontext wir das Attribut eingesetzt haben. Die mit dem benutzerdefinierten Attribut versehene Adresse-Klasse sieht folgendermaßen aus (die Attribute-Verwendungen sind fett markiert).

---

**HINWEIS:** Sie sehen anhand dieses Beispiels, dass der Namenszusatz Attribute weggelassen werden kann, wenn er korrekt buchstabiert wurde.

---

```
<Serializable(), Mein("Über der Klasse")> _
Public Class Adresse

    Private myName As String
    Private myVorname As String
    Private myStraße As String
    Private myPLZOrt As String
    Private myErfasstAm As DateTime
    Private myErfasstVon As String
    Private myBefreundetMit As ArrayList

    <Mein("Über dem Konstruktor")> -
    Sub New(ByVal Vorname As String, ByVal Name As String, ByVal Straße As String, ByVal PLZOrt As String)
        'Konstruktor legt alle Member-Daten an.
        myName = Name
        myVorname = Vorname
        myStraße = Straße
        myPLZOrt = PLZOrt
        myErfasstAm = DateTime.Now
        myErfasstVon = Environment.UserName
        myBefreundetMit = New ArrayList
    End Sub

    <Mein("Über einer Eigenschaft")> -
    Public Property Name() As String
        Get
            Return myName
        End Get
        Set(ByVal Value As String)
            myName = Value
        End Set
    End Property

    #Region "Die anderen Eigenschaften"
    'Aus Platzgründen hier nicht gezeigt.
    #End Region

    <Mein("Über einer Methode")> -
    Public Overrides Function ToString() As String
        Dim locTemp As String
        locTemp = Name + ", " + Vorname + ", " + Straße + ", " + PLZOrt + vbNewLine
        locTemp += "---- Befreundet mit: ---" + vbNewLine
        For Each locAdr As Adresse In BefreundetMit
            locTemp += "    * " + locAdr.ToStringShort() + vbNewLine
        Next
    End Function
```

```

    LocTemp += vbNewLine
    Return LocTemp
End Function

Public Function ToStringShort() As String
    Return Name + ", " + Vorname
End Function
End Class

```

Ziel von Reflection ist es nun, die Attribute zur Laufzeit zu erkennen. Schauen wir uns zunächst das Ergebnis vorweg an, damit das eigentliche Auswertungsprogramm, das die Attribute findet, anschließend leichter zu verstehen ist.

Wenn Sie das Programm starten, produziert es die folgende Bildschirmausgabe:

```

Attribute der Klasse:Reflection02.Adresse
Standardattribute:
*AutoLayout, AnsiClass, NotPublic, Public, Serializable

Benutzerattribute:
* Reflection02.MeinAttribute

Member-Liste:
*GetHashCode, Method
*Equals, Method
*ToString, Method
-----
* Reflection02.MeinAttribute
Name: Über einer Methode
TypeId: Reflection02.MeinAttribute
*get_Name, Method
*set_Name, Method
*get_ErfasstAm, Method
*get_ErfasstVon, Method
*get_BefreundetMit, Method
*set_BefreundetMit, Method
*get_Vorname, Method
*set_Vorname, Method
*get_Straße, Method
*set_Straße, Method
*get_PLZOrt, Method
*set_PLZOrt, Method
*ToStringShort, Method
*GetType, Method
*.ctor, Constructor
-----
* Reflection01.MeinAttribute
Name: Über dem Konstruktor
TypeId: Reflection02.MeinAttribute
*Name, Property
-----
* Reflection02.MeinAttribute
Name: Über einer Eigenschaft
TypeId: Reflection02.MeinAttribute

```

```

Wert: Halek
*ErfasstAm, Property
Wert: 12.04.2004 11:43:58
*ErfasstVon, Property
Wert: Administrator
*BefreundetMit, Property
Wert: System.Collections.ArrayList
*Vorname, Property
Wert: Gaby
*Straße, Property
Wert: Buchstraße 223
*PLZOrt, Property
Wert: 32154 Autorhausen

```

Die Passagen, bei denen sowohl das Attribut selbst als auch sein jeweils gültiger Parameter erkannt wurden, sind im Bildschirmauszug fett markiert.

## Ermitteln von benutzerdefinierten Attributen zur Laufzeit

Um benutzerdefinierte Attribute zu ermitteln, gibt es die Funktion `GetCustomAttributes`. Diese Funktion ist sowohl auf einen Typ (die Klasse oder Struktur selbst) als auch auf einzelne Member anwendbar. Als Parameter übernimmt sie entweder einen booleschen Wert, der bestimmt, ob in der Hierarchieliste des Objektes vorhandene Typen ebenfalls auf Attribute untersucht werden sollen (`True`), oder zum einen den Typ des Attributes, nach dem gezielt gesucht werden soll, und zum anderen den booleschen Wert für das Durchsuchen der Hierarchieliste zusätzlich.

Da die Parameter einer Attributklasse in der Regel als Eigenschaften abrufbar sind, können Sie, nachdem Sie das Attribut ermittelt haben, mit `GetType` seine Typ-Instanz abrufen, anschließend seine Eigenschaften mit `GetProperties` auflisten und schließlich mit `GetValue` den eigentlichen Wert einer Attributeigenschaft auslesen.

Das modifizierte Beispielprogramm zeigt, wie es geht. Es liest am Anfang sowohl ein mögliches, benutzerdefiniertes Attribut aus, das für die gesamte Klasse gilt, und untersucht anschließend jeden einzelnen Klassen-Member auf Attribute:

```

Module mdlMain

Sub Main()
    Dim locAdresse As New Adresse("Gaby", "Halek", "Buchstraße 223", "32154 Autorhausen")
    PrintObjectInfo(locAdresse)
    Console.ReadLine()
End Sub

Sub PrintObjectInfo(ByVal [Object] As Object)
    'Den Objekttypen ermitteln, um auf die Objektinhalte zuzugreifen
    Dim locTypeInstanz As Type = [Object].GetType

    'Die nicht Benutzerdefinierten Attribute ausgeben:
    Console.WriteLine("Attribute der Klasse:" + locTypeInstanz.FullName)
    Console.WriteLine("Standardattribute:")
    Console.WriteLine("    " + locTypeInstanz.Attributes.ToString())
    Console.WriteLine()

```

```

'Benutzerdefinierte Attribute der Klasse ermitteln.
'(Es können auf diese Weise *nur* benutzerdefinierte Attribute ermittelt werden)
Console.WriteLine("Benutzerattribute:")
For Each locAttribute As Attribute In locTypeInstanz.GetCustomAttributes(True)
    Console.WriteLine("    * " + locAttribute.ToString())
Next
Console.WriteLine()

'Member und deren mögliche Attribute ermitteln.
Dim locMembers() As MemberInfo
locMembers = locTypeInstanz.GetMembers()
Console.WriteLine("Member-Liste:")
For Each locMember As MemberInfo In locMembers
    Console.WriteLine("        *" + locMember.Name + ", "
        + locMember.MemberType.ToString())
    If locMember.GetCustomAttributes(True).Length > 0 Then
        Console.WriteLine("            " + New String("-"c, locMember.Name.Length))
        For Each locAttribute As Attribute In locMember.GetCustomAttributes(False)
            Console.WriteLine("                * " + locAttribute.ToString())
            For Each loc PropertyInfo As PropertyInfo In locAttribute.GetType.GetProperties
                Console.WriteLine("                    " + loc PropertyInfo.Name)
                Console.WriteLine(": " + loc PropertyInfo.GetValue(locAttribute, Nothing).ToString())
            Next
        Next
    End If
    If locMember.MemberType = MemberTypes.Property Then
        Dim loc PropertyInfo As PropertyInfo = CType(locMember, PropertyInfo)
        Console.WriteLine("            Wert: " + loc PropertyInfo.GetValue([Object], Nothing).ToString())
    End If
    Next
End Sub
End Module

```

---

**HINWEIS:** Bitte beachten Sie, dass Sie mit GetCustomAttributes ausschließlich benutzerdefinierte Attribute auslesen können. Möchten Sie wissen, ob eine Klasse beispielsweise serialisierbar ist, können Sie entweder mit einer der IsXxx-Funktionen ihres Type-Objektes (IsSerializable beispielsweise) oder mit der Attributes-Eigenschaft an diese Informationen gelangen.

---



# Teil F

## Vereinfachungen in Visual Basic 2005

---

- 705    Eine philosophische Betrachtung der Vereinfachungen in Visual Basic 2005**
  - 713    Der My-Namespace**
  - 731    Das Anwendungsframework**
- 

Visual Basic unterscheidet sich von anderen .NET-Programmiersprachen in ein paar Eigenarten, die in erster Linie durch seine Historie bedingt sind: Es gibt einige Elemente, die das Programmieren mit Visual Basic .NET-untypisch zu vereinfachen scheinen. Und dies gilt für Visual Basic 2005 in besonderem Maße. Namentlich sind das die Funktionalitäten, die durch das Konzept des Anwendungsframeworks und durch den My-Namespace zur Verfügung gestellt werden, und die in diesem Teil des Buches besprochen werden.

Auch Visual Basic .NET 2002 und 2003 unterschied sich von den Standards anderer .NET-Programmiersprachen, aber anders als Visual Basic 2005: Hier funktionierten viele Dinge, wie beispielsweise das Überladen von Operatoren, noch nicht.

Nun haben gerade die Visual Basic 2005-eigenen Elemente schon während der Entwicklung für teilweise heftige Diskussionen gesorgt, denn: Nachdem Visual Basic .NET endlich »erwachsen« wurde, indem es anders als noch Visual Basic 6.0 zur echten objektorientierten Programmiersprache avancierte, sorgte exakt diese Tatsache auch für Unmut bei dem Teil der VB6-Fraktion, die sich mit der OOP-Programmierung nicht so recht anfreunden konnte oder wollte. Die Entwickler von Visual Basic 2005 erkannten das nach Markteinführung von Visual Basic .NET 2003 recht schnell und sahen sich nun ein wenig in der Klemme: Zum einen wollte man den VB6-Programmierern der alten Riege nicht einfach die Tür vor der Nase zuknallen und sie vor die »Friss-oder-Stirb-Alternative« stellen, andererseits wollte man aber auch Visual Basic an sich weiter voranbringen, und gerade in Sachen OOP, Operatorenüberladung und Generics zum Standard anderer .NET-Sprachen aufschließen.

Auch mit dieser Diskussion beschäftigt sich der nun folgende Buchteil.



# 24 Eine philosophische Betrachtung der Vereinfachungen in Visual Basic 2005

---

- 706 Die VB2005-Vereinfachungen am Beispiel DotNetCopy
  - 711 Die prinzipielle Funktionsweise von DotNetCopy
- 

Es gibt exklusiv in Visual Basic 2005 einige Hilfsmittel (man könnte sie pessimistisch betrachtet auch als »Eigentümlichkeiten« bezeichnen), die auf den ersten Blick das OOP-Konzept des .NET-Framework zu umgehen scheinen. Es gibt obendrein einige Dinge, bei denen OOP-Puristen den VB6lern der alten Riege vorwerfen, man hätte die neuen Hilfsmittel in VB 2005 eigentlich nicht implementieren dürfen, weil es VB2005 wieder zurück in den Pool der »Baby-Programmiersprachen« wirft, und – Zitat – »man dürfe die Blödheit oder Sturheit der prozeduralen Fraktion nicht noch unterstützen«.

Ich gebe zu, ich sah mich eine Weile mehr in der letzteren Fraktion der OOP-Puristen als in der, der alten prozeduralen VB6-Riege. Und auch ich wollte nicht, dass mein schönes neues jetzt endlich auch erwachsenes VB.NET durch »nachgemachte« VB6-Features aufgeweicht und der Ruf »meines« VB 2005 dadurch vielleicht wieder angeknackst wird – Visual Basic hatte (und hat!) es schließlich in bestimmten Gemeinden eh schwer genug, sich als professionelle Programmiersprache zu etablieren.

Doch wissen Sie was: Diese ganze Diskussion ist eigentlich völlig überflüssig, und mit den folgenden Argumenten werde ich bewusst wahrscheinlich für Umsatzeinbußen in so manchen Kneipen sorgen, in denen Diskussionen über exakt dieses Thema geführt werden, denn:

OOP-Puristen, lasst euch gesagt sein: Visual Basic hat all das, was zum OOP-Programmieren benötigt wird. Und ich persönlich, der ich in C# und Visual Basic 2005 eigentlich gleichermaßen gut zu Recht komme, programmiere in VB2005 lieber, weil meiner Meinung nach Visual Basic-Editor und -Background-Compiler ein schnelleres und effizienteres Arbeiten als die C#-Äquivalente gestatten. Wenn ihr Funktionalitäten, die in diesem Buchteil besprochen werden, wie beispielsweise das Anwendungsframework oder den My-Namespace nicht nutzen wollt – niemand wird dazu gezwungen! Macht euch die Arbeit halt schwerer, als sie sein müsste, und braucht dann eben länger zum Ziel.

Euer Hauptargument, dass für die Vereinfachung Komponenten verwendet werden, die eigentlich nicht zum .NET-Framework gehören, ist – mit Verlaub gesagt – ziemlicher Unsinn. Die *Visual-Basic.dll* ist Bestandteil des Frameworks, und alle Vereinfachungen, mit denen wir vielleicht auch die »alten« VB6ler überzeugen könnten, bei »uns mitzumachen«, sind dort implementiert. Und wenn ihr sagt, ihr wollt auf diese Vereinfachungen verzichten, weil ihr Gerüchten glaubt, die sagen, dass diese Visual Basic-eigenen Dinge in Zukunft aus dem Framework verbannt werden, dann dürft ihr übrigens gar nicht mehr in Visual Basic programmieren. Denn, und das ist wichtig: Ohne die *Visual-Basic.dll* läuft überhaupt kein Visual Basic-Programm, egal welchen Ansatz (einfach oder .NET-

puristisch) ihr verfolgt oder welcher Komponenten aus dem Framework ihr euch bedient. Und wenn ihr wisst, dass die *VisualBasic.dll* erstens sowieso genauso zentraler Bestandteil des Frameworks ist, wie alle anderen .NET-Assemblies und sich zweitens diese Vereinfachungen wie Anwendungsframework und My-Namespace auch dort befinden, riskiert doch mal einen Blick in die folgenden Seiten. Viele Probleme lassen sich damit wirklich einfach und schnell lösen – ich kann's am Beispiel beweisen!

## Die VB2005-Vereinfachungen am Beispiel DotNetCopy

Jeder, der mit seinem Computer regelmäßig viele neue und wichtige Dateien generiert – und zu dieser Riege gehören wir Softwareentwickler ja zwangsläufig wohl alle – weiß, wie wichtig Datensicherungen sind. Aber getreu dem Motto »Der Schuster hat die schlechtesten Schuhe« gibt's hier und da den einen oder anderen, der dieses Gebiet ein wenig vernachlässigt – ist's nicht so? Der Autor dieser Zeilen will sich da gar nicht von frei sprechen. Ich persönlich kopiere meine Daten jeden Abend auf einen Backup-Rechner, und das mithilfe eines simplen Batch, der in der Eingabeaufforderung von Windows XP läuft. Eine regelrechte differenzierende Bandsicherung mit einem Band für jeden Wochentag empfehle ich lediglich meinen Kunden ...

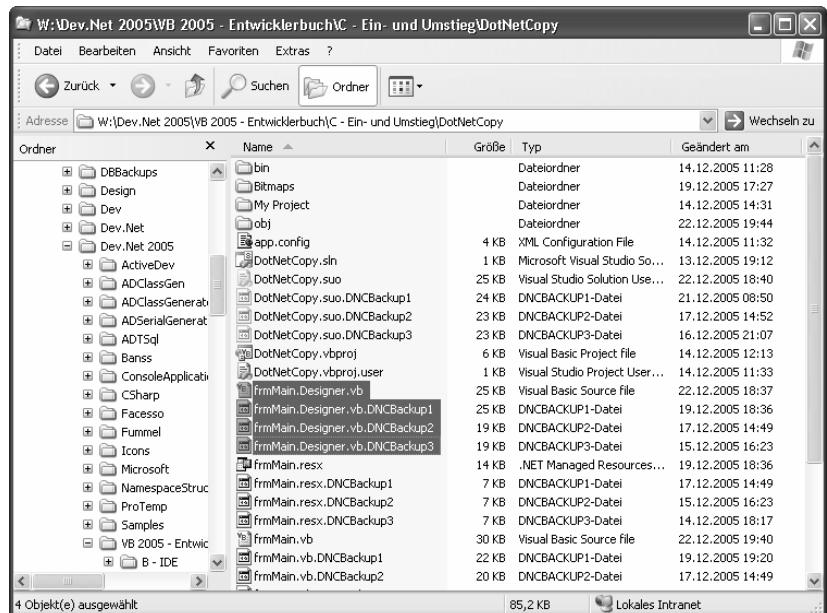
So wäre es immerhin ein Fortschritt, gäbe es ein Programm, das die Dateien bestimmter Verzeichnisse nicht nur kopiert, sondern auch beliebige Backup-Historien der Dateien erstellt, die vor einem erneuten Sichern einer aktualisierten Datei versionsmäßig »nach oben« rutschen, wie etwa in Abbildung 24.1 zu sehen.

Das Beispiel zu diesem Buchteil kann genau das, und es kann noch viel, viel mehr: Sie können DotNetCopy so einstellen, dass es nur Dateien kopiert, die im Zielverzeichnis entweder nicht vorhanden oder älter sind als die Dateien, die es zu sichern gilt. Und im Bedarfsfall werden die im Zielverzeichnis vorhandenen Dateien nicht einfach ersetzt, sodass die alte Version auf Nimmerwiedersehen verschwindet. Vielmehr können Sie bestimmen, wie viele Backup-Stufen der älteren Dateien automatisch vorgehalten werden sollen – Abbildung 24.1 zeigt auch das in Form einer Dateien-Backup-Historie, wie das später auf dem Backup-Laufwerk aussieht.

---

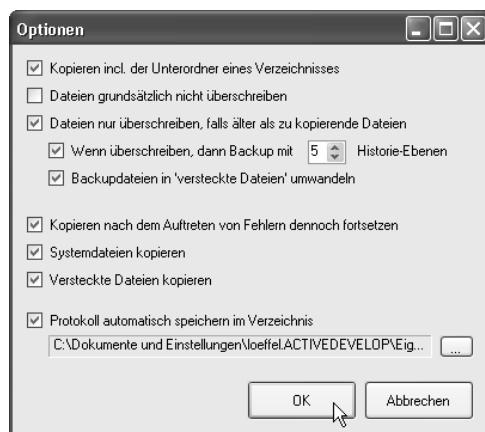
**BEGLEITDATEIEN:** Sie finden dieses Projekt übrigens unter dem Namen *DotNetCopy.sln* im Verzeichnis *.\VB 2005 - Entwicklerbuch\ F - Vereinfachung\Kap25\DotNetCopy*.

---



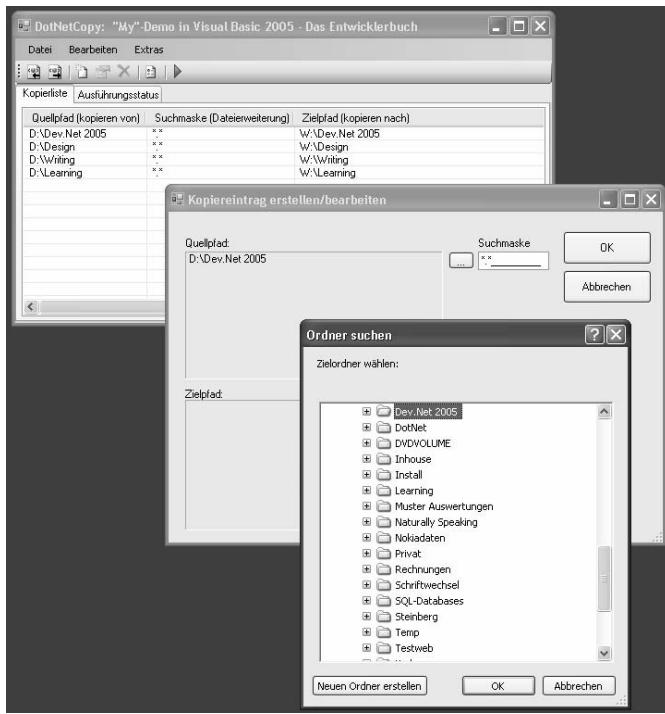
**Abbildung 24.1:** DotNetCopy kopiert Dateien im Bedarfsfall nur, wenn eine Datei neuer ist als eine vorhandene im Zielverzeichnis und legt automatisch Backup-Historien an

Der Optionsdialog von DotNetCopy erlaubt es, das Kopierverhalten genau zu kontrollieren:



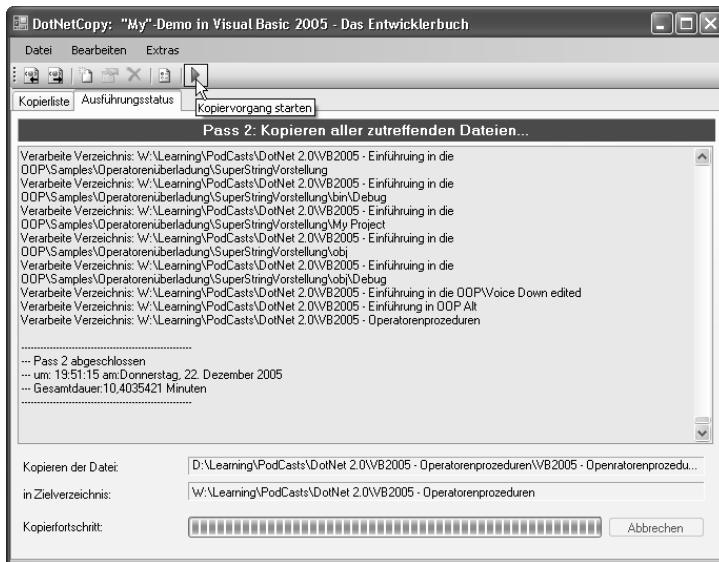
**Abbildung 24.2:** DotNetCopy erlaubt es, bestimmte Kopieroptionen im Optionen-Dialog zu konfigurieren

Sie erstellen mit DotNetCopy anschließend eine Kopierliste, die das Programm darüber informiert, welches Quellverzeichnis wohin kopiert (also gesichert) werden soll. Kopierlisten, die Sie auf diese Art und Weise erstellen, lassen sich sogar abspeichern und laden, was die Anwendung des Programms recht flexibel macht:



**Abbildung 24.3:** Erstellen Sie mit DotNetCopy Kopierlisten, die beliebig viele Quell- und Zielpfade enthalten und geladen sowie gespeichert werden können

Und wenn Sie eine Kopierliste erstellt und für die spätere Wiederverwendung gespeichert haben, drücken Sie einfach aufs Knöpfchen, und der Kopievorgang kann beginnen, wie in der folgenden Grafik zu sehen:



**Abbildung 24.4:** DotNetCopy nach erfolgreichem Backup-Lauf

## DotNetCopy mit /Autostart und /Silent-Optionen

Damit dieses Tool aber auch für Administratoren interessant ist, die es vielleicht in eigene Batch-Dateien an der Befehlszeile einbinden möchten, lässt es sich auch im Autostart- sowie im Silent-Modus starten.

- Der Autostart-Modus erlaubt es, eine Kopierlistendatei beim Aufruf des Programms anzugeben. Diese Kopierlistendatei wird dann anschließend geladen und der Backup-Vorgang beginnt unmittelbar. Der Aufruf von DotNetCopy im Kopierlistenmodus erfolgt dazu an der Eingabeaufforderung oder über Ausführen im Start-Menü. Sie geben zusätzlich zum Programmnamen *DotNetCopy* die Option */Autostart:kopierlistendatei.ols* an, also beispielsweise

```
Dotnetcopy /autostart:backup.ols
```

- Zusätzlich können Sie das Programm so starten, dass es ohne sichtbare Benutzeroberfläche läuft. In diesem Fall verwenden Sie die Option */silent*, die Sie durchaus mit der zuvor beschriebenen Option */autostart* kombinieren können. Um also beispielsweise die obige als Beispiel gezeigte Kopierliste automatisch anzeigen zu lassen, würden Sie die Befehlszeile

```
Dotnetcopy /autostart:backup.ols /silent
```

dazu verwenden können.

Nachdem Sie die Funktionsweise von DotNetCopy kennen gelernt haben, geht's als nächstes darum zu erfahren, was die Kapitel dieses Buches mit der ganzen Geschichte zu tun haben.

Und falls Sie denken, Sie müssten zum Verständnis des Programms tief in die Abgründe des .NET-Framework hinabsteigen – weit gefehlt. DotNetCopy benötigt dank der Visual Basic-eigenen Vereinfachungen überraschend wenig Code.

Und selbst Visual Basic 6-Umsteigern werden lediglich zwei Klassen des Framework neu sein – fast die gesamte andere Funktionalität basiert komplett auf Funktionalitäten, die durch den My-Namespace und das Anwendungsframework zur Verfügung gestellt werden.

## Exkurs: Die Klassen FileInfo- und DirectoryInfo zur Pflege von Verzeichnissen und Dateien

Um Zugriff auf Dateien und Verzeichnisse der Festplatte oder eines anderen Datenträgers Ihres Computers zu nehmen, stellt Ihnen das .NET-Framework die Klassen `FileInfo`- und `DirectoryInfo` zur Verfügung.

Die `FileInfo`-Klasse dient zur »Bearbeitung« von Dateien im Sinne des Windows-Explorers; `DirectoryInfo` stellt im Gegensatz dazu die äquivalente Funktionalität für Verzeichnisse zur Verfügung: Stellen Sie sich beide Klassen am besten wie einen Windows-Datei-Explorer vor, der nur eben keine sichtbare Benutzeroberfläche hat: Mit der `FileInfo`-Klasse haben Sie beispielsweise die Möglichkeit, Attribute von Dateien abzurufen, Dateien umzubenennen, sie an einen anderen Ort zu verschieben, auf deren Vorhandensein zu überprüfen und Ähnliches. Mit `DirectoryInfo` arbeiten Sie prinzipiell genau so – nur eben mit Verzeichnissen.

Das Schwierige bei Dateioperationen ist in der Regel gar nicht so sehr das Aufrufen bestimmter Funktionen, wenn Sie erst einmal den exakten Pfad und Dateinamen einer Datei kennen. ►

Es ist das richtige Zusammenbauen des Pfades, das Extrahieren des eigentlichen Dateinamens aus der Zeichenkettenkombination von Pfad und Dateinamen oder auch das Ermitteln der Dateiendung einer kompletten Pfad-/Dateinamenkombination. Und auch dabei helfen Ihnen die Klassen FileInfo- und DirectoryInfo. Beim Erstellen neuer Unterordner in mehreren Verzeichnisebenen brauchen Sie sich obendrein auch nicht darum zum kümmern, ob Ordner in Teilebenen schon existieren – wenn Sie beispielsweise mit der DirectoryInfo-Klasse ein Verzeichnis namens C:\Ordner1\UnterOrdner\EigentlicherOrdner anlegen wollen, werden im Bedarfsfall alle benötigten Ordner (Ordner1, UnterOrdner) gleich mit angelegt. Die folgenden Codezeilen geben einige Beispiele für die Anwendung der FileInfo- bzw. DirectoryInfo-Klasse:

### Beispiel DirectoryInfo-Objekt:

```
'Neues DirectoryInfo-Objekt aus Pfad-Zeichenkette erstellen:  
Dim einDirInfo As New DirectoryInfo("C:\Ordner1\Unterordner")  
'Existiert das Verzeichnis?  
If Not einDirInfo.Exists Then  
    Debug.Print("Ordner existiert nicht, wird erstellt:")  
    'Verzeichnis mit allen benötigten Unterverzeichnissen erstellen  
    einDirInfo.Create()  
End If
```

### Beispiel FileInfo-Objekt:

```
'Neues FileInfo-Objekt aus Pfad-/Dateinamenzeichenkette erstellen  
Dim einFileInfo As New FileInfo("c:\Order1\Unterordner\textfile.txt")  
  
'Jedes FileInfo-Objekt enthält auch ein DirectoryInfo-Objekt, das  
'über die Directory-Eigenschaft abrufbar ist:  
If Not einFileInfo.Directory.Exists Then  
    'Pfad zur Datei im Bedarfsfall erstellen  
    einFileInfo.Directory.Create()  
End If  
'Auf Vorhandensein der Datei prüfen:  
If Not einFileInfo.Exists Then  
    'Eine simple Textdatei erstellen.  
    My.Computer.FileSystem.WriteAllText(einFileInfo.FullName, "Dateinhalt", False)  
End If  
Debug.Print("Die Erweiterung der Datei lautet: " & einFileInfo.Extension)  
Debug.Print("Der reine Dateiname lautet: " & einFileInfo.Name)  
Debug.Print("Voller Pfad und Dateiname zur Datei lauten: " & einFileInfo.FullName)  
Debug.Print("Die Datei hat folgende Attribute: " & einFileInfo.Attributes)  
Debug.Print("Die Datei wird jetzt ins Hauptverzeichnis kopiert!")  
'Wichtig: Pfad und neuer Dateiname müssen als Ziel beide angegeben werden!  
einFileInfo.CopyTo("C:\" & einFileInfo.Name)  
Debug.Print("Die Ausgangsdatei wird gelöscht!")  
einFileInfo.Delete()
```

Dieser Code würde folgende Ausgabe erzeugen:

```
Die Erweiterung der Datei lautet: .txt
Der reine Dateiname lautet: myfile.txt
Voller Pfad und Dateiname zur Datei lauten: c:\Order1\Unterordner\myfile.txt
Die Datei hat folgende Attribute: 32
Die Datei wird jetzt ins Hauptverzeichnis kopiert!
Die Ausgangsdatei wird gelöscht!
```

## Die prinzipielle Funktionsweise von DotNetCopy

Wie gesagt: DotNetCopy dient dazu, die beiden Besonderheiten von Visual Basic 2005 in Vergleich zu den anderen .NET-Framework-Programmiersprachen zu demonstrieren. Dazu gehören die Funktionalitäten, die durch das so genannte Anwendungsframework zur Verfügung gestellt werden, sowie der My-Namespace, der die Aufgabe hat, bestimmte thematisch kategorisierte Probleme, die Ihnen bei den täglichen Entwicklungsaufgaben immer wieder begegnen, einfacher lösen zu können.

Beim Anwendungsframework, dessen Funktionalität und Handhabung das folgende Kapitel beschreibt, bedient sich DotNetCopy dreierlei Dinge: Dem Zur-Verfügung-Stellen der Visual Styles von Windows XP, den Anwendungseignissen und einer Funktion, die verhindert, dass eine zweite Instanz von DotNetCopy gestartet werden kann, wenn eine Instanz von DotNetCopy bereits aktiv ist.<sup>1</sup>

Da DotNetCopy die Option /Silent kennt muss es einen Mechanismus geben, der quasi noch vor dem Anzeigen des ersten Formulars greift, und der eben in diesem Modus verhindert, dass das erste Formular angezeigt wird. Das Anwendungsframework stellt zu diesem Zweck eine Reihe von Ereignissen zur Verfügung, und unter anderem auch ein Ereignis, das ausgelöst wird, nachdem das Programm gestartet ist. Ob es zur Anzeige des Hauptformulars kommt oder nicht, wird in dieser Ereignisbehandlungsroutine namens *MyApplication\_Startup* geregelt, die sich in der Codedatei *Application-Events.vb* des Projektes befindet.

Im Silent-Modus wird anschließend zwar das Hauptformular, das auch die komplette Funktionalität für den Backup-Vorgang in *frmMain.vb* enthält, instanziert, aber nicht dargestellt. Für die korrekte Funktionsweise ist das jedoch kein Nachteil – selbst der Code für die Formularaktualisierungen (Fortschrittsanzeige und Protokollausgabe aktualisieren, aktuell bearbeitetes Verzeichnis anzeigen etc.) funktioniert, nur dass der Anwender sie während des Backup-Vorgangs nicht sieht, da das Formular selbst eben nicht angezeigt wird.

Die weitere Funktionsweise ist daher auch in allen Modi gleich. Nachdem eine Kopierliste geladen wurde – die Funktion *LoadCopyEntryList* ist dafür zuständig, und sie wird entweder durch die Funktion *HandleAutoStart* (ihrerseits durch *MyApplication\_Startup* aufgerufen) oder durch das Auswählen des entsprechenden Menüpunkts oder Symbols vom Anwender getriggert – startet der Kopiervorgang in der Methode *CopyFiles* (auch entweder automatisch durch *HandleAutoStart* oder durch das Auswählen des entsprechenden Menüpunkts oder Symbols durch den Anwender).

---

<sup>1</sup> Ob dieser letzte Punkt für das richtige Funktionieren von DotNetCopy tatsächlich erforderlich ist, darüber lässt sich sicherlich streiten – vielleicht möchte auch der ein oder andere, dass DotNetCopy in mehreren Instanzen gleichzeitig laufen kann. Für die Demonstration des Anwendungsframeworks ist es alle Male geeignet.

Und in CopyFiles und allen Methoden, die dort aufgerufen werden, kommt das zweite Thema dieses Buchteils intensiv zum Einsatz – Funktionen aus dem My-Namespace.

Wenn Sie einen Blick in die Projektdateien werfen, werden Sie sehen, dass das Programm ausführlich kommentiert ist. Zudem werden Sie die wichtigen Ausschnitte in den nächsten Kapiteln wieder finden.

Obendrein haben Sie durch .NET-Copy nicht nur ein anschauliches Beispiel, wie einfach sich auch komplexere Anwendungen mit VB und den »VB-Vereinfachungen« entwickeln lassen, sondern auch ein Tool, das der eine oder andere von Ihnen vielleicht auch wirklich gebrauchen kann. Bei mir hat DotNetCopy jedenfalls seit seiner Fertigstellung meine alten Batch-Prozesse abgelöst, und die Backup-Historie von Dateien, die es generiert, habe ich schon mehr als einmal gut gebrauchen können.

Und vielleicht noch eine letzte Anmerkung zum Thema OOP und den Visual Basic-Vereinfachungen: DotNetCopy ist sicherlich kein Glanzbeispiel für OOP. Aber es ist brauchbar, stabil und war schnell entwickelt. Das gleiche Tool in der Sprache C#, in der es die VB-Vereinfachungen nicht gibt, hätte sicherlich einen Tag mehr Entwicklungsaufwand bedeutet. OOP ist gerade bei größeren Anwendungen sicherlich ein Muss, auch in Visual Basic und trotz der Tatsache, dass es My und das Anwendungsframework gibt. Aber OOP verleitet auch dazu, mehr zu machen, als eigentlich gefordert ist. Denken Sie daran, dass Sie für ein Einfamilienhaus auch kein Fundament für einen Wolkenkratzer ausheben würden, und diese Analogie sollten Sie im Hinterkopf behalten, wenn Sie die Klassenplanung für ein größeres Projekt in Angriff nehmen.

Es ist nichts Verwerfliches dabei, die speziellen VB-Vereinfachungen zu verwenden. Wenn Sie damit Zeit sparen können, und absehen können, dass Sie sich damit für spätere Erweiterungen den Weg nicht verbauen, nutzen Sie sie. Und lassen Sie die C#- oder VB-Entwickler, die Ihnen etwas anderes weismachen wollen, ruhig reden ...

# 25 Der My-Namespace

---

- 
- 715 **Formulare ohne Instanzierung aufrufen**
  - 716 **Auslesen der Befehlszeilenargumente mit My.Application.CommandLineArgs**
  - 718 **Gezieltes Zugreifen auf Ressourcen mit My.Resources**
  - 721 **Internationalisieren von Anwendungen mithilfe von Ressource-Dateien und dem My-Namespace**
  - 724 **Vereinfachtes Durchführen von Dateioperationen mit My.Computer.FileSystem**
  - 727 **Verwenden von Anwendungseinstellungen mit My.Settings**
- 

Der My-Namespace wurde in Visual Basic 2005 eingeführt, um den Umgang mit bestimmten Funktionalitäten zu vereinfachen – um sozusagen »Abkürzungen« zu bestimmten Zielen mit Funktionalitäten im .NET-Framework zur Verfügung zu stellen, die Sie normalerweise nur auf verschlungenen Pfaden erreichen würden.

So Sie das Beispielszenario aus ► Kapitel 3 komplett durchexerziert haben, sind Sie auch schon in den Genuss dieser Abkürzungen gekommen – Sie haben dort nämlich kennen gelernt, wie einfach Sie auf selbst speichernde Anwendungseinstellungen durch `My.Settings` zurückgreifen können.

Doch der My-Namespace verfügt über noch weit mehr wirklich coole Features. Welche das allerdings genau sind, lässt sich nicht verbindlich sagen, denn die Features, die Ihnen durch `My` zur Verfügung stehen, hängen von dem Projekttyp ab, den Sie gerade bearbeiten.

Generell teilt sich die Funktionalität im My-Namespace in die folgenden Kategorien auf:

- **My.Application:** Stellt Zugriff auf Informationen über die aktuelle Anwendung bereit, wie beispielsweise den Pfad zur ausführbaren Datei, die Programmversion, derzeitige Kulturinformationen oder den Benutzer-Authentifizierungsmodus.
- **My.Computer:** Ermöglicht Ihnen den Zugriff auf mehrere untergeordnete Objekte, die Ihnen wiederum das Abrufen von computerbezogenen Informationen gestatten, wie beispielsweise über das verwendete Dateisystem, über verwendete Audio- und Videospezifikationen, angeschlossene Drucker oder generelle I/O-Hardware wie Maus, Tastatur, Speicher, die verwendete Netzwerkumgebung, serielle Schnittstellen und Weiteres.
- **My.Forms:** Gestattet Ihnen das Abrufen von Standardinstanzen aller Formulare einer Windows Forms-Anwendung.

- **My.Resources:** Ermöglicht den einfachen Zugriff auf eingebettete Ressourcen, wie beispielsweise auf Zeichenkettentabellen, Bitmaps und Ähnliches.
- **My.Settings:** Ermöglicht einerseits, auf Anwendungseinstellungen Zugriff zu nehmen, und sorgt andererseits dafür, dass diese Anwendungseinstellungen im Bedarfsfall benutzerabhängig beim Programmstart wiederhergestellt und beim Programmende gesichert werden.
- **My.User:** Ermöglicht das Abrufen von Infos über den zurzeit angemeldeten Benutzer und erlaubt ferner das Implementieren benutzerdefinierter Authentifizierungsmechanismen.
- **My.WebServices:** Stellt eine Eigenschaft für jeden Webservice zur Verfügung, den das aktuelle Projekt referenziert, und erlaubt so auf Webservices Zugriff zu nehmen, ohne eine explizite Proxy-Klasse für den jeweiligen Webservice erstellen zu müssen.

In einer Konsolenanwendung, die nun einmal keine Formulare verwendet, steht Ihnen natürlich die Kategorie My.Forms nicht zur Verfügung. Und so hängt die tatsächliche Verfügbarkeit der My-Features durchweg von den Projekttypen ab, mit denen Sie es gerade zu tun haben, wie die folgende Tabelle zeigt:

My-Objekt	Windows-Anwendung	Klassenbibliothek	Konsolenanwendung	Windows-Steuerelementbibliothek	Web-Steuerelementbibliothek	Windows-Dienst
My.Application	Ja	Ja	Ja	Ja	Nein	Ja
My.Computer	Ja	Ja	Ja	Ja	Ja	Ja
My.Forms	Ja	Nein	Nein	Ja	Nein	Nein
My.Resources	Ja	Ja	Ja	Ja	Ja	Ja
My.Settings	Ja	Ja	Ja	Ja	Ja	Ja
My.User	Ja	Ja	Ja	Ja	Ja	Ja
My.WebServices	Ja	Ja	Ja	Ja	Ja	Ja

Nun finde ich, es würde an dieser Stelle keinen Sinn machen, die komplette Referenz aller Funktionalitäten jedes einzelnen Bereichs herunterzubeten – im Gegenteil: Schließlich dient My eben genau dazu, auch ohne großes Blättern und nur mithilfe von IntelliSense und dynamischer Hilfe, schnell an die gewünschte Funktionalität zu gelangen.

Stattdessen sollten Sie gerade von einem Entwicklerbuch erwarten können, dass Ihnen der Autor Sachverhalte im Rahmen einer Softwareentwicklung näher bringt – und das hat er für die Demonstration von My auch getan. Die in den folgenden Abschnitten beschriebenen My-Funktionalitäten erheben deshalb keinen Anspruch auf Vollständigkeit – sie sollen es auch gar nicht. Ziel ist es vielmehr, zum einen generelle Vorgehensweisen beim Umgang mit dem My-Namespace zu vermitteln. Die richtigen Funktionen für Ihre eigenen Bedürfnisse zu finden, wird dank IntelliSense und Online-Hilfe sicherlich nicht *das* Problem sein, und würde an dieser Stelle nur wertvollen Platz beschränken.

Es gibt zum anderen auch einiges Erwähnenswertes zum My-Namespace und seinen Funktionalitäten, was Sie nicht in der Online-Hilfe finden, und auch diesen Punkten widmen sich die folgenden Abschnitte.

---

**BEGLEITDATEIEN:** Viele der hier gezeigten Features können Sie sich direkt am Beispiel von DotNetCopy (vorgestellt im vorherigen Kapitel) anschauen, und innerhalb der Abschnitte werden Sie auch immer wieder Codeausschnitte aus diesem Beispiel finden. Sie finden dieses Projekt unter dem Namen *DotNetCopy.sln* im Verzeichnis *.\VB 2005 - Entwicklerbuch\ F - Vereinfachung\Kap25\DotNetCopy*. So Sie das vorherige Kapitel noch nicht gelesen haben, sollten Sie es zunächst tun, um die Beispiele besser nachvollziehen zu können.

---

## Formulare ohne Instanzierung aufrufen

Die Änderungen an Visual Basic, um die es jetzt geht, haben in der bis dato etablierten VB.NET-Gemeinde für das meiste Aufsehen gesorgt,<sup>1</sup> denn: Es geht um die Vereinfachung von Formularaufrufen, und das Hauptargument dabei ist: Damit nähert sich Visual Basic politisch betrachtet wieder einer »Kindersprache«, bei der man dem »unmündigen« VB-Programmierer nicht zumuten kann, die buchstäblichen Basics der OOP zu kennen und anzuwenden.

Worum geht es genau?

Auch schon in VB6 mussten Sie, um mit einem Objekt arbeiten zu können, es zunächst instanzieren. Haben Sie also eine Collection verwendet, in der Sie andere Objekte speichern wollten, waren folgende Zeilen notwendig:

```
Dim auflistung As Collection  
Set auflistung = New Collection  
auflistung.Add 5
```

Es ist klar, dass diese Zeilen nicht funktioniert hätten:

```
Collection.Add 10  
Collection.Add 15  
Collection.Add 20
```

Bei VB6-Formularen passiert aber genau das: Hier können Klassename und Instanz das gleiche sein. Diese Version funktioniert:

```
Dim f As Form1  
f.Show
```

genau wie die Verwendung der Klasse Form1 als Instanzvariable:

```
Form1.Show
```

Das entspricht natürlich nicht den Vorschriften zur objektorientierten Programmierung, war aber wahrscheinlich für viele VB6-Programmierer einfacher zu verstehen, und aus diesem Grund wurde der Basic-Compiler so frisiert, dass eine solche Verwendung einer Formulklassen möglich war.

Schon in Visual Basic 6 passierte »unter der Haube« dazu etwas, was nunmehr auch in VB2005 wieder möglich ist.

Der VB-Compiler sorgt nämlich bei der Komplilierung einer Windows Forms-Anwendung dafür, dass versteckter Quellcode, der im Compiler fest verdrahtet ist, mit in Ihre Anwendung kompiliert

---

<sup>1</sup> Vergleichen Sie dazu bitte auch den Einführungstext zu diesem Buchteil.

wird. Der VB-Compiler »erfindet« also quasi einen Haufen von Quellcode und fügt diesen, unsichtbar für den Entwickler, dem eigentlichen Windows-Projekt hinzu.

So können Sie auch in Visual Basic 2005 wieder folgende Zeile verwenden, um das unter dem Klassennamen Form2 erstellte Formular beispielsweise als modalen Dialog ins Leben zu rufen:

```
Form2.ShowDialog
```

Doch es sieht hier nur so aus, als würden Sie eine statische Klassenmethode direkt verwenden, denn in Wirklichkeit ist Form2 eine Eigenschaft, die eine *Instanz* vom Typ Form2 zurückliefert. Denn was hier eigentlich passiert, ist, dass der Compiler diese Zeile nur vervollständigt, und er im Grunde genommen folgenden Quellcode für die Generierung der eigentlichen ausführbaren Datei kompiliert:

```
MyProject.Forms.Form2.ShowDialog
```

Die Klasse MyProject ist dabei allerdings eine Klasse, die Sie unter diesem Namen nicht erreichen können, weil sie mit entsprechenden Attributen so gekennzeichnet ist, dass IntelliSense sie nicht »sieht«. Sie ist aber dennoch vorhanden. Quasi versteckt legt der Compiler also für jedes Formular, das Sie Ihrem Projekt hinzufügen, eine Eigenschaft vom Typ des jeweiligen Formulars in der (ebenfalls unsichtbaren) MyForms-Klasse an, die eine eingebettete Klasse der Klasse MyProject selbst ist, deren Instanz durch die Forms-Eigenschaft der MyProject-Klasse abgerufen werden kann. Und erst diese Eigenschaft, die den Namen des Formulars trägt, sorgt dafür, dass die gewünschte Formularklasse vor Gebrauch instanziert und dann ohne Probleme (bzw. NullReference-Ausnahmen) verwendet werden kann.

Sie können das ausprobieren. Tippen Sie »**My**.«, dann werden Sie sehen, dass IntelliSense MyProject nicht zur Auswahl anbietet.

Komplettieren Sie jedoch die Zeile mit »**MyProject.Forms.Form2.ShowDialog**«, meldet der Visual Basic-Compiler dennoch keinen Fehler.

Zur Veranschaulichung: Die Programmzeile

```
MyProject.Forms.Form2.ShowDialog
```

bedeutet im Grunde genommen:

```
(MyProject-Klasse).(Forms-Eigenschaft der MyProject-Klasse vom Typ MyForms).  
(Form2-Eigenschaft der MyForms-Klasse vom Typ Form2).(ShowDialog-Methode der Form2-Klasse).
```

## Auslesen der Befehlszeilenargumente mit My.Application.CommandLineArgs

Anwendungen, die unter Windows laufen, können Sie auf unterschiedliche Weisen starten. Die einfachste und bekannteste ist, sie durch einen Doppelklick auf eine Verknüpfung beispielsweise aus dem Startmenü zum Laufen zu bewegen. Sie können jede Windows- oder Befehlszeilenanwendung aber auch direkt durch die Angabe des Namens der ausführbaren Datei aus der Befehlszeile oder durch den *Ausführen*-Befehl des Start-Menüs heraus in Gang bekommen. Und viele Anwendungen erlauben es hier, weitere Parameter anzugeben, die das Startverhalten steuern. So starten Sie beispielsweise den Windows-Explorer mit

Explorer c:\

um zu definieren, dass der Explorer nicht nur gestartet wird, sondern dass er auch direkt den Inhalt des Wurzelverzeichnisses von Laufwerk c: anzeigt.

Parameter, die Sie Anwendungen auf diese Weise übergeben, nennt man Befehlszeilenargumente, und mithilfe des My-Namespace können Sie Befehlszeilenargumente auf einfache Weise ermitteln und auswerten.

Das My-Beispiel DotNetCopy arbeitet ebenfalls mit solchen Befehlszeilenargumenten, nämlich um die Anwendung in den Autostart- oder Silent-Modus zu versetzen. Der folgende Code, der sich im StartUp-Ereignis des MyApplication-Objektes befindet, und der ausgelöst wird, sobald die Anwendung startet, demonstriert den Umgang mit Befehlszeilenargumenten.

---

**HINWEIS:** Sie finden diesen Code in der Datei *ApplicationEvents.vb* des Projektes. Mehr zu Anwendungereignissen erfahren Sie übrigens im nächsten Kapitel.

---

Partial Friend Class MyApplication

```
Private Sub MyApplication_Startup(ByVal sender As Object, ByVal e As Microsoft.VisualBasic.ApplicationServices.StartupEventArgs) Handles Me.Startup

    'Das Verzeichnis für die Protokolldatei beim ersten Mal setzen...
    If String.IsNullOrEmpty(My.Settings.Option_AutoSaveProtocolPath) Then
        My.Settings.Option_AutoSaveProtocolPath =
            My.Computer.FileSystem.SpecialDirectories.MyDocuments & "\DotNetCopy Protokolle"
        Dim locDi As New DirectoryInfo(My.Settings.Option_AutoSaveProtocolPath)

        'Überprüfen und
        If Not locDi.Exists Then
            'im Bedarfsfall anlegen
            locDi.Create()
        End If

        'Settings speichern
        My.Settings.Save()
    End If

    Dim locFrmMain As New frmMain

    'Kommandozeile auslesen
    'Sind überhaupt Befehlszeilenargumente vorhanden?
    If My.Application.CommandLineArgs.Count > 0 Then
        'Durch jedes einzelne Befehlszeilenargument durchiterieren
        For Each locString As String In My.Application.CommandLineArgs

            'Alle unnötigen Leerzeichen entfernen und
            'Groß-/Kleinschreibung 'Unsensibilisieren'
            'HINWEIS: Das funktioniert nur in der Windows-Welt;
            'kommt die Kopierlistendatei von einem Unix-Server, bitte darauf achten,
            'dass der Dateiname dafür auch komplett in Großbuchstaben gesetzt ist,
            'da Unix- (und Linux-) Derivate Groß-/Kleinschreibung berücksichtigen!!!
            locString = locString.ToUpper.Trim
        End If
    End If
End Sub
```

```

If locString = "/SILENT" Then
    locFrmMain.SilentMode = True
End If

If locString.StartsWith("/AUTOSTART") Then
    locFrmMain.AutoStartCopyList = locString.Replace("/AUTOSTART:", "")
    locFrmMain.AutoStartMode = True
End If
Next
End If
.
.
```

## Gezieltes Zugreifen auf Ressourcen mit My.Resources

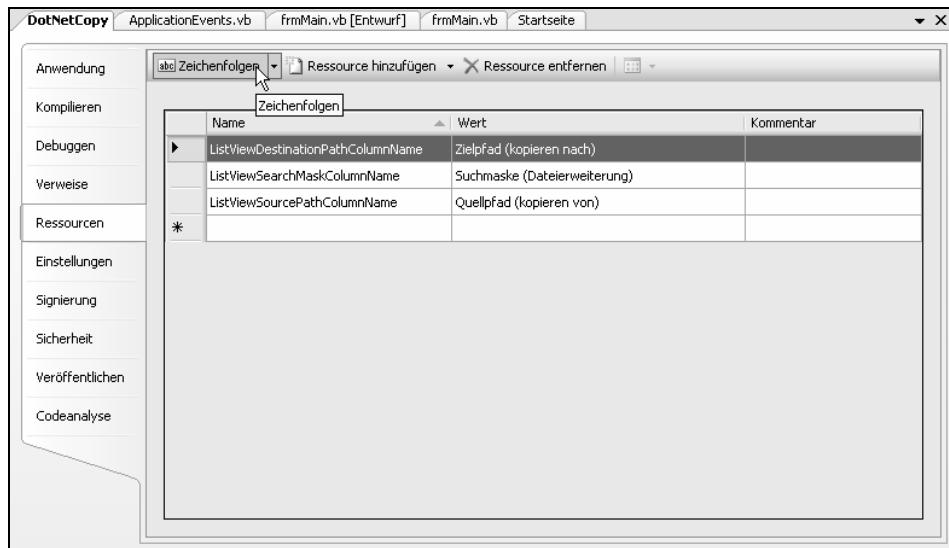
Ressource-Dateien speichern bestimmte Elemente einer Anwendung in einer separaten Datei. Bei solchen Elementen handelt es sich beispielsweise um Bitmaps, Symbole, Zeichenketten oder Ähnliches. Diese Elemente können mithilfe von My.Resources zur Laufzeit aus Ressource-Dateien gelesen und anschließend im Kontext verwendet werden.

Der Vorteil dieser Vorgehensweise: Anwendungen können auch im Nachhinein angepasst werden, ohne dem Bearbeiter den eigentlichen Code der Anwendung zur Verfügung zu stellen. Denken Sie dabei nur zum Beispiel an die Lokalisierung einer Anwendung in eine andere Sprache. Wären die Texte der Anwendung fest verdrahtet, müsste man den Übersetzern die kompletten Quellen der Anwendung zur Verfügung stellen. Ein weiterer Nachteil wäre, dass es verschiedene Versionen des Quellcodes gäbe, von denen jede einzelne bei Änderungen oder Fehlerbehebungen in der Anwendung bearbeitet werden müsste – ein Aufwand, der bei größeren Projekten nicht zu leisten ist.

## Anlegen und Verwalten von Ressource-Elementen

In unserer Beispielanwendung werden die Texte für die Spaltennamen der ListView, die die Kopierlisteneinträge beinhaltet, aus einer Ressource-Datei entnommen. Die Ressource-Datei selber wird ähnlich verwaltet, wie Sie es bei den Anwendungseinstellungen (den ApplicationSettings) bereits im ► 3. Kapitel kennen gelernt haben.

- Rufen Sie, um die Ressource-Datei einer Anwendung zu bearbeiten, einfach die Projekteigenschaften über das Kontext-Menü des Projektmappen-Explorers auf.
- Wählen Sie anschließend die Registerkarte Ressourcen.
- Wählen Sie aus der linken Schaltfläche, die Sie aufklappen können (siehe Abbildung 25.1), den Typ Ressource, den Sie einpflegen möchten. Bei Zeichenketten erfassen Sie die Texte wie bei Anwendungseinstellungen in Form von Texttabellen. Bei anderen Ressource-Typen, wie beispielsweise Bitmaps, Symbolen oder Audio-Clips, verwenden Sie die Schaltfläche Ressource hinzufügen, um eine entsprechende Datei der Ressource-Datei hinzuzufügen.



**Abbildung 25.1:** Bestimmen Sie wie hier den Typ Ressource, den Sie der Ressource-Datei hinzufügen möchten. Bei anderen Typen als Zeichenketten verwenden Sie Ressource hinzufügen, um Ressource-Elemente wie Bitmaps, Symbole oder Audio-Clips der Ressource-Datei hinzuzufügen.

## Abrufen von Ressourcen mit My.Ressources

Wenn Sie Ressourcen auf die im letzten Abschnitt beschriebene Weise Ihrem Projekt hinzugefügt haben, ist das Abrufen der Ressource-Elemente zur Laufzeit dank My ein Einfaches. Im Beispielprojekt DotNetCopy wird das beispielsweise im Ereignisbehandler `form_Load` des Formulars gemacht; hier werden beispielhaft die Spaltentexte der `ListView` eingerichtet:

```

'<summary>
'<> Wird aufgerufen, wenn das Formular geladen wird, und enthält die
'<> Initialisierung der ListView sowie das Anstoßen des Kopievorgangs
'<> (und das zuvor notwendige Laden der Kopierliste), wenn das Programm
'<> im Autostart (aber nicht im Silent) -Modus läuft.
'<> (Silent-Modus wird in ApplicationEvents.vb gehandelt).
'</summary>
'<param name="sender"></param>
'<param name="e"></param>
'<remarks></remarks>
Private Sub frmMain_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles Me.Load
    'Listview einrichten
    With Me.lvwCopyEntries.Columns
        'Die Texte für die Listview-Spalten aus der Ressource-Datei entnehmen
        .Add(My.Resources.ListViewSourcePathColumnName)
        .Add(My.Resources.ListViewSearchMaskColumnName)
        .Add(My.Resources.ListViewDestinationPathColumnName)
    End With
End Sub

```

```
'Spalten ausrichten  
AlignColumns()  
End With
```

.

.

Natürlich könnte man dieses Verfahren auch beispielsweise auf die angezeigten Texte im Backup-Protokoll ausweiten. Der Einfachheit halber habe ich die Verwendung von Text-Ressourcen auf diesen Bereich beschränkt.

**HINWEIS:** Im Übrigen machen WinForms-Anwendungen beim Zuweisen von Texten oder Bildern ebenfalls implizit von Ressource-Dateien Gebrauch. Zwar verwendet der Windows Forms-Designer für den generierten Code nicht die Funktionalität aus dem My-Namespace, aber das Holen beispielsweise von Bitmap-Dateien aus einer Ressource, um dieser ein Symbol etwa einer Symbolleistenschaltfläche hinzuzufügen, funktioniert prinzipiell auf die gleiche Weise. Sie können sich selbst ein Bild davon machen: Wenn Sie im Projektmappen-Explorer die ausgeblendeten Dateien des Projektes mit dem Symbol *Alle Dateien anzeigen* (der Tooltip hilft beim Finden des Symbols) einblenden, können Sie einen Blick in den Code werfen, der zum Aufbau des Formulars führt. Dazu doppelklicken Sie anschließend auf die Datei *frmMain.Designer.vb*, die sich jetzt im Zweig unterhalb der Formulardatei *frmMain.vb* befindet.

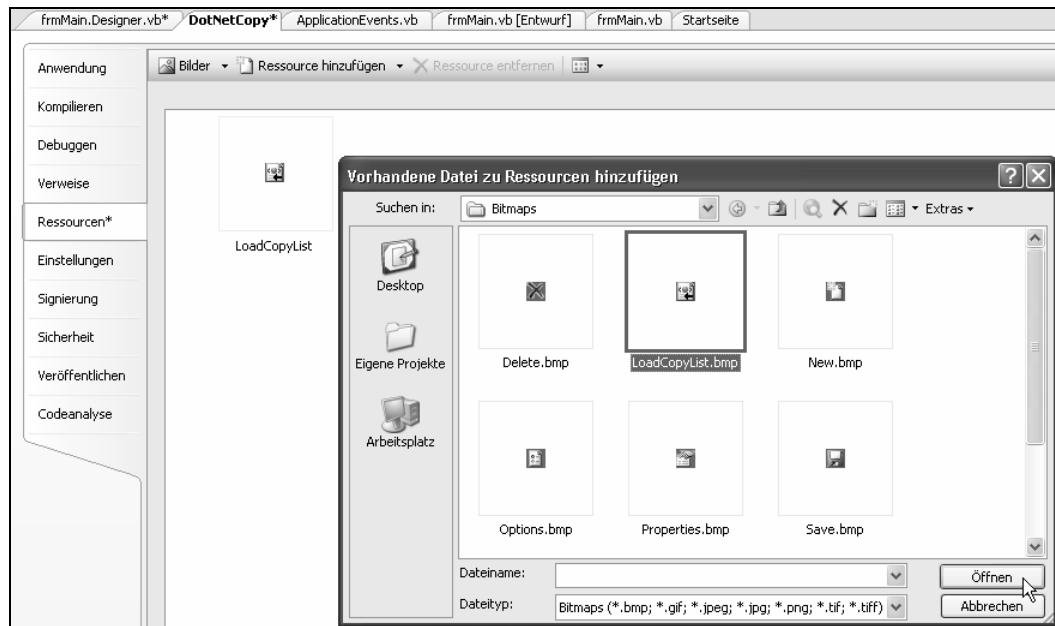
---

```
'tsbLoadCopyList – Auszug aus der FormsDesigner generierten Datei zum Zuweisen  
'eines Images an ein Symbol in DotNetCopy:  
Me.tsbLoadCopyList.DisplayStyle = System.Windows.Forms.ToolStripItemDisplayStyle.Image  
Me.tsbLoadCopyList.Image = CType(resources.GetObject("tsbLoadCopyList.Image"), _  
System.Drawing.Image)  
  
Me.tsbLoadCopyList.ImageTransparentColor = System.Drawing.Color.Magenta  
  
.
```

Die Zuweisung eines Symbols mithilfe des My-Namespace würde folgendermaßen ausschauen:

```
Me.tsbLoadCopyList.Image = My.Resources.LoadCopyList
```

Voraussetzung dafür wäre, dass eine entsprechende Image-Datei auf die zuvor beschriebene Weise der Ressource-Datei hinzugefügt wurde, etwa wie in Abbildung 25.2 zu sehen.



**Abbildung 25.2:** Klappen Sie die Schaltfläche *Ressource hinzufügen* auf (nicht darauf klicken!), und wählen Sie aus der Aufklappliste *Datei hinzufügen*, um beispielsweise eine Bitmap der Ressource-Datei hinzuzufügen, die Sie dann zur Laufzeit mit *My.Resource* und dem Element-Laden zuweisen können

## Internationalisieren von Anwendungen mithilfe von Ressource-Dateien und dem My-Namespace

Der Einsatz von Ressource-Dateien macht natürlich nur dann Sinn, wenn Sie planen, eine Anwendung tatsächlich auch in einer anderen Sprache zur Verfügung zu stellen. Für diesen Vorgang des Lokalisierens müssen Sie weitere Ressource-Dateien erstellen, die auf Basis eines bestimmten Schemas benannt und dem Projekt hinzugefügt werden.

Leider gibt es in Visual Studio 2005 für Windows Forms-Anwendungen in Visual Basic keine Designer-Unterstützung, die Ihnen wirklich dabei hilft, neue Ressource-Dateien für andere Kulturen aus schon vorhandenen zu erstellen. Sie müssen dazu selbst Hand anlegen und die IDE auch ein wenig austricksen. Die Unterstützung im Designer beschränkt sich lediglich auf Formulare selbst, bei denen Sie einfach mit Ihrer Localizable-Eigenschaft bestimmen können, ob das Formular lokalisiert werden soll oder nicht.

---

**HINWEIS:** Wenn Sie die folgenden Schritte nachvollziehen möchten, empfehle ich Ihnen, die Projektdateien zum »Kaputtexperimentieren« zunächst an einen anderen Ort auf Ihrer Festplatte zu kopieren. Sollte bei den Versuchen dann etwas schief gehen, können Sie das Quellprojekt einfach wieder dort rüberkopieren, und ohne Verlust von vorne beginnen.

---

- Öffnen Sie als Erstes die »Experimentier-Kopie« von DotNetCopy und lassen Sie den Projektmappen-Explorer mit **Strg+Alt+L** darstellen, falls dieser noch nicht angezeigt wird.
- Klicken Sie auf das entsprechende Symbol im Projektmappen-Explorer, um alle Dateien des Projektes anzeigen zu lassen. Öffnen Sie den Zweig, der sich neben dem jetzt sichtbaren Eintrag *My Project* befindet.
- Die Datei *Resources.resx*, in der sich die Ressourcen für die deutsche Kultur befinden, wird nun neben anderen angezeigt.
- Klicken Sie mit der rechten Maustaste auf diese Datei, um das Kontextmenü zu öffnen, und wählen Sie dort den Eintrag *Kopieren*.
- Im nächsten Schritt müssen Sie die IDE ein wenig austricksen: Sie müssen nun die Datei exakt an der Stelle einfügen, und damit eine Kopie der Ressource-Datei erstellen, an der die Ausgangsdatei ebenfalls platziert war. Klicken Sie allerdings wieder mit der rechten Maustaste, dann befindet sich im Kontextmenü kein Eintrag namens *Einfügen*. Aus diesem Grund klicken Sie die Datei *Resources.resx* mit der linken Maustaste an, um sie zu selektieren, und wählen Sie anschließend *Einfügen* aus dem Menü *Bearbeiten* der Visual Studio IDE. Eine Kopie der Datei steht anschließend ebenfalls im Zweig *My Project*.
- Nun benennen Sie die Ressource-Datei um. Das .NET-Framework wird dann später, wenn es Ihr Programm ausführen muss, anhand des Namens der Ressource-Datei wissen, welche Datei es für welche zugrunde liegende Kultur verwenden soll. Auf deutschen Systemen wird, wenn Sie auf einer deutschen Windows-Plattform entwickelt haben, immer die Ressource-Datei verwendet, mit der Sie die Anwendung entwickelt haben. Für ein englisches System (bzw. US-amerikanisches) benennen Sie die gerade erstellte Kopie der Ressource-Datei in *Resources.en.resx* um, für französisch in *Resources.fr.resx*, für italienisch in *Resources.it.resx* usw.

---

**TIPP:** Die gängigen Sprachkürzel entsprechen denen, wie sie bei der Programmierung von kulturabhängigen Format Providern verwendet werden. Sie können deswegen die entsprechende Liste in ► Kapitel 17 (Tabelle 17.1) verwenden.

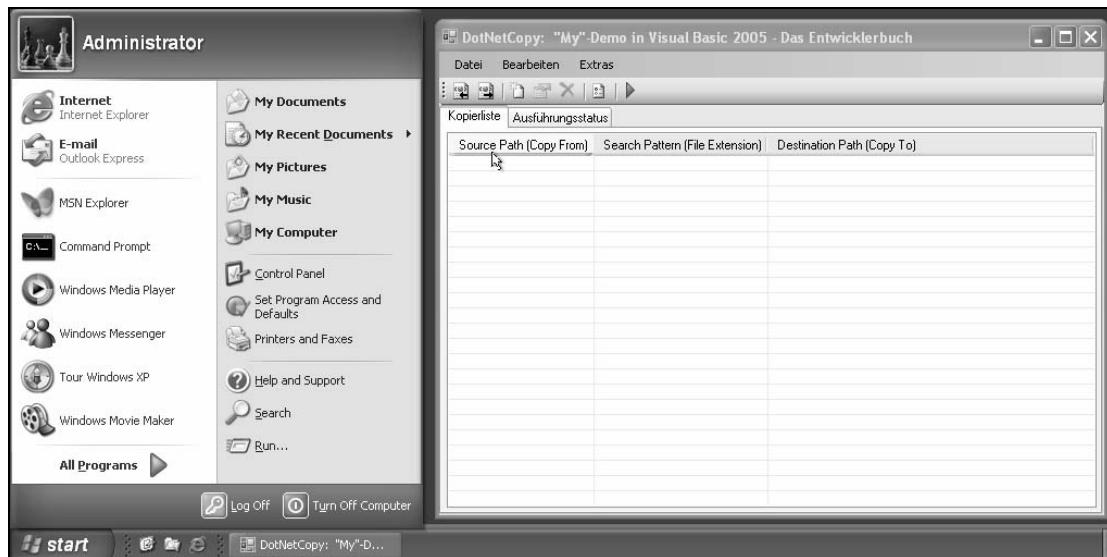
---

Name	Wert	Kommentar
ListViewDestinationPathColumnName	Destination Path (Copy To)	
ListViewSearchMaskColumnName	Search Pattern (File Extension)	
ListViewSourcePathColumnName	Source Path (Copy From)	

**Abbildung 25.3:** Wenn Sie die zusätzliche Ressource-Datei ins Projekt kopiert haben, können Sie sich an das Übersetzen der Ressourcen machen

- Wenn Sie die Datei umbenannt haben,<sup>2</sup> müssen Sie noch mit ein paar Handgriffen dafür sorgen, dass diese neue Ressource-Datei nicht dazu herhält, eine weitere Codedatei zu erzeugen, bei der dann, bedingt durch die schon vorhandene deutsche Version, Codeelemente für das Abrufen der Ressourcen doppelt definiert würden und so Fehler generieren. Dazu klicken Sie die umbenannte Datei an und löschen im Eigenschaftenfenster die Werte für *Benutzerdefiniertes Tool* und *Namespace des benutzerdefinierten Tools*.
- Doppelklicken Sie anschließend auf die unbenannte Datei, um die Einträge, die sie beinhaltet, zu lokalisieren – etwa wie in Abbildung 25.3 zu sehen.
- Speichern Sie die Änderungen anschließend und lassen Sie das komplette Projekt neu erstellen.

Sie werden anschließend sehen, dass im Verzeichnis der ausführbaren Dateien des Projektes (standardmäßig ist das der Pfad *Projekt\bin\Debug*) ein weiteres Unterverzeichnis hinzugekommen ist, das eine so genannte Satelliten-Assembly beinhaltet. Diese Satelliten-Assembly beinhaltet die lokalisierten Ressourcen für jede Ausgangsressource-Datei und wird vom .NET-Framework automatisch verwendet, wenn der Name des Unterordners mit dem entsprechenden Kulturkürzel der Kultur des Systems übereinstimmt, auf dem die Anwendung ausgeführt wird (*en* für Englisch, *fr* für Französisch usw.). Das bedeutet: Wenn Sie die Satelliten-Assemblies auf diese Weise erzeugt haben, dann brauchen Sie sich selbst um nichts Weiteres mehr zu kümmern – sorgen Sie lediglich dafür, dass sie in entsprechenden Unterverzeichnissen vorhanden sind.



**Abbildung 25.4:** Auf diesem ohne Zweifel englischen Betriebssystem wird automatisch – so vorhanden – die englische Satelliten-Assembly verwendet; in der Ressource-Datei zuvor lokalisierte Texte erscheinen dann auch in der Landessprache.

---

<sup>2</sup> Laut Aussagen des Entwicklerteams ist dieser Schritt in der finalen Version von Visual Studio 2005 nicht mehr nötig; ein Auslassen dieses Schrittes hat aber bei einigen Experimenten Probleme verursacht, und so kann es nicht schaden, diesen zusätzlichen Schritt dennoch zu erledigen.

Auf einem englischen System würde die Anwendung dann automatisch so erscheinen, wie es Abbildung 25.4 auch wiedergibt (achten Sie hier auf die Bezeichnungen der ListView-Spalten, denn nur diese werden beispielhaft bei DotNetCopy aus der Ressource-Datei entnommen).

## Vereinfachtes Durchführen von Dateioperationen mit My.Computer.FileSystem

FileInfo und DirectoryInfo unterstützten Sie bei der Ausführung von Dateioperationen schon auf eine sehr komfortable Art und Weise. Doch bestimmte Funktionen im My-Namespace können das noch besser, und sie machen erst es möglich, dass unser Backup-Tool DotNetCopy trotz der vergleichsweise großen Flexibilität dennoch relativ wenig Code benötigt.

Die Funktionen, von denen DotNetCopy gerade bei der Durchführung des eigentlichen Backup-Vorgangs sehr intensiv Gebrauch macht, finden sich in My.Computer.FileSystem. Besonders bemerkenswert sind hier Funktionen, mit denen ganze Verzeichnisstrukturen bzw. die Namen der sich dort befindlichen Dateien rekursiv eingelesen werden können. Sie kennen das vielleicht von dem *Dir*-Befehl der Befehlszeile von Windows: Nutzen Sie diesen mit der Option /S, werden nicht nur die Verzeichnisse bzw. Dateien des Verzeichnisses aufgelistet, das Sie angegeben haben bzw. in dem Sie sich derzeitig befinden, sondern es werden auch alle Unterverzeichnisse berücksichtigt.

Die eigentliche Kopierroutine von DotNetCopy macht von dieser GetFiles-Funktion Gebrauch, indem sie alle Unterverzeichnisse samt der darin enthaltenen Dateien ermittelt, zwischenspeichert und gemäß der so erstellten Liste im zweiten Schritt all diese Dateien kopiert. Der folgende Codeausschnitt demonstriert den Umgang:

```
''' <summary>
''' Die eigentliche Kopierroutine, die durch die My.Settings-Optionen gesteuert wird.
''' </summary>
''' <remarks></remarks>
Public Sub CopyFiles()

    'Aus Platzgründen gekürzt

    'In dieser Schleife werden zunächst alle Dateien ermittelt,
    'die es daraufhin zu untersuchen gilt, ob sie kopiert werden
    'müssen oder nicht.
    For locItemCount As Integer = 0 To lvwCopyEntries.Items.Count - 1
        Dim locCopyListEntry As CopyListEntry
        'CType ist notwendig, damit .NET weiß, welcher Typ
        'in der Tag-Eigenschaft gespeichert war.
        locCopyListEntry = CType(lvwCopyEntries.Items(locItemCount).Tag, CopyListEntry)
        'UI aktualisieren
        lblCurrentPass.Text = "Pass 1: Kopiervorbereitung durch Zusammenstellung der Pfade..."
        lblSourceFileInfo.Text = "Unterverzeichnisse suchen in:"
        lblDestFileInfo.Text = ""
        lblProgressCaption.Text = "Fortschritt Vorbereitung:"
        lblCurrentSourcePath.Text = locCopyListEntry.SourceFolder.ToString
        LogInProtocolWindow(locCopyListEntry.SourceFolder.ToString)
        pbPrepareAndCopy.Value = locItemCount + 1
```

```

'Windows die Möglichkeit geben, die Steuerelemente zu aktualisieren
My.Application.DoEvents()

'GetFiles aus My.Computer.FileSystem liefert die Namen aller
'Dateien im Stamm- und in dessen Unterverzeichnissen.
Dim locFiles As ReadOnlyCollection(Of String)
Try
    locNotCaught = False
    locFiles = My.Computer.FileSystem.GetFiles(
        locCopyListEntry.SourceFolder.ToString(),
        FileIO.SearchOption.SearchAllSubDirectories,
        locCopyListEntry.SearchMask)
Catch ex As Exception
    LogError(ex)
    'Wenn schon beim Zusammenstellen der Dateien Fehler aufgetreten sind,
    'sollte das Verzeichnis ausgelassen werden. In diesem Fall sollte dieser
    'schwerwiegende Fakt im Anwendungsprotokoll protokolliert werden!
    My.Application.Log.WriteLine(ex, TraceEventType.Error, _
        "DotNetCopy konnte das Verzeichnis " &
        locCopyListEntry.SourceFolder.ToString() & " nicht durchsuchen., " &
        "Das Verzeichnis wurde daher nicht berücksichtigt!")

    ' Es müsste wie "Catch" einen "NotCaught"-Zweig geben...
    locNotCaught = True
End Try
.
.
.
```

Eine weitere Funktionalität, die ebenfalls sehr sinnvoll und praktisch ist, stellt `My.Computer.FileSystem` in Form der `CopyFile`-Methode zur Verfügung. Anders als die Methode `CopyTo` der `FileInfo`-Klasse greift diese exakt auf die gleiche Betriebssystemfunktion von Windows zurück, die auch beispielsweise der Windows-Explorer verwendet. Und das bedeutet – auch wenn `DotNetCopy` in diesem Beispiel davon keinen Gebrauch macht – dass Sie bei der Verwendung von `CopyFile` aus dem `My`-Namespace automatisch den bekannten Fortschrittsdialog anzeigen lassen können.

```

''' <summary>
''' Interne Kopierroutine, die den eigentlichen 'Job' des Kopierens übernimmt.
''' </summary>
''' <param name="SourceFile">Quelldatei, die kopiert werden soll.</param>
''' <param name="DestFile">Zielpfad, in den die Datei hineinkopiert werden soll.</param>
''' <remarks></remarks>
Private Sub CopyFileInternal(ByVal SourceFile As FileInfo, ByVal DestFile As FileInfo)
    'Falls die Zielfile noch gar nicht existiert,
    'dann in jedem Fall kopieren
    If Not DestFile.Exists Then
        Try
            'Datei kopieren, ohne Windows-Benutzeroberfläche anzuzeigen.
            My.Computer.FileSystem.CopyFile(SourceFile.ToString(), DestFile.ToString())
            LogInProtocolWindow("Kopiert, OK: " & SourceFile.ToString())
        Catch ex As Exception
            LogError(ex)
        End Try
    End If
End Sub
```

```

    Exit Sub
End If
'Datei nur kopieren, wenn Sie jünger als die zu überschreibende ist
If My.Settings.Option_OnlyOverwriteIfOlder Then
    If SourceFile.LastWriteTime > DestFile.LastWriteTime Then
        'Wenn das Historien-Backup aktiviert ist...
        If My.Settings.Option_EnableBackupHistory Then
            '...dieses durchführen.
            ManageFileBackupHistory(DestFile)
        End If
    Try
        'Datei kopieren.
        My.Computer.FileSystem.CopyFile(SourceFile.ToString, DestFile.ToString)
        LogInProtocolWindow("Kopiert, OK: " & SourceFile.ToString)
    Catch ex As Exception
        LogError(ex)
    End Try
End If
End If
End Sub

```

**HINWEIS:** Leider werden bei der Verwendung von CopyFile in Windows-Anwendungen, bei denen – wie es standardmäßig bei Windows Forms-Anwendungen geschieht – Formulare an eine Meldungswarteschlange gebunden werden, auch Fehlermeldungen als modale Dialoge dargestellt. Gerade bei Fehlern, die das Programm behandeln soll, kann das lästig und unerwünscht sein. Wichtig ist es aber auch zu wissen, dass bei Anwendungen, die keine Meldungswarteschlange initiieren, Fehlermeldungen nicht vom Betriebssystem in Form eines sichtbaren modalen Dialogs abgefangen werden, oder mit anderen Worten: Wenn Sie dafür sorgen, dass Ihr Programm keine sichtbare Benutzeroberfläche hat, dann erscheint auch keine sichtbare Fehlermeldung, selbst wenn bei CopyFile ein Fehler auftritt.

Im Falle unseres Beispiels DotNetCopy bedeutet das: Wir können im Silent-Modus keine sichtbaren Fehlermeldungen gebrauchen. Die gibt es aber auch bei CopyFile in diesem Modus sowieso nicht, denn es gibt keine sichtbare Benutzeroberfläche. Zwar verwenden wir die gleiche Formularklasse, die wir auch verwenden, wenn DotNetCopy nicht im Silent-Modus läuft, aber wir zeigen das Formular nicht an.

Der folgende Code, der sich im StartUp-Ereignis des MyApplication-Objektes befindet, und der ausgelöst wird, sobald die Anwendung startet, demonstriert das:

```

'Hinweis: locFrmMain wurde weiter oben im Code schon instanziert, aber nicht dargestellt!
'Silentmode bleibt nur "an", wenn AutoStart aktiv ist.
locFrmMain.SilentMode = locFrmMain.SilentMode And locFrmMain.AutoStartMode

'Und wenn Silentmode, erfolgt keine Bindung des Formulars an den Anwendungskontext!
If locFrmMain.SilentMode Then
    'Alles wird in der nicht sichtbaren Instanz des Hauptforms durchgeführt,
    locFrmMain.HandleAutoStart()
    'und bevor das "eigentliche" Programm durch das Hauptformular gestartet wird,
    'ist der ganze Zauber auch schon wieder vorbei.
    e.Cancel = True
Else
    'Im Nicht-Silent-Modus wird das Formular an die Anwendung gebunden,

```

```

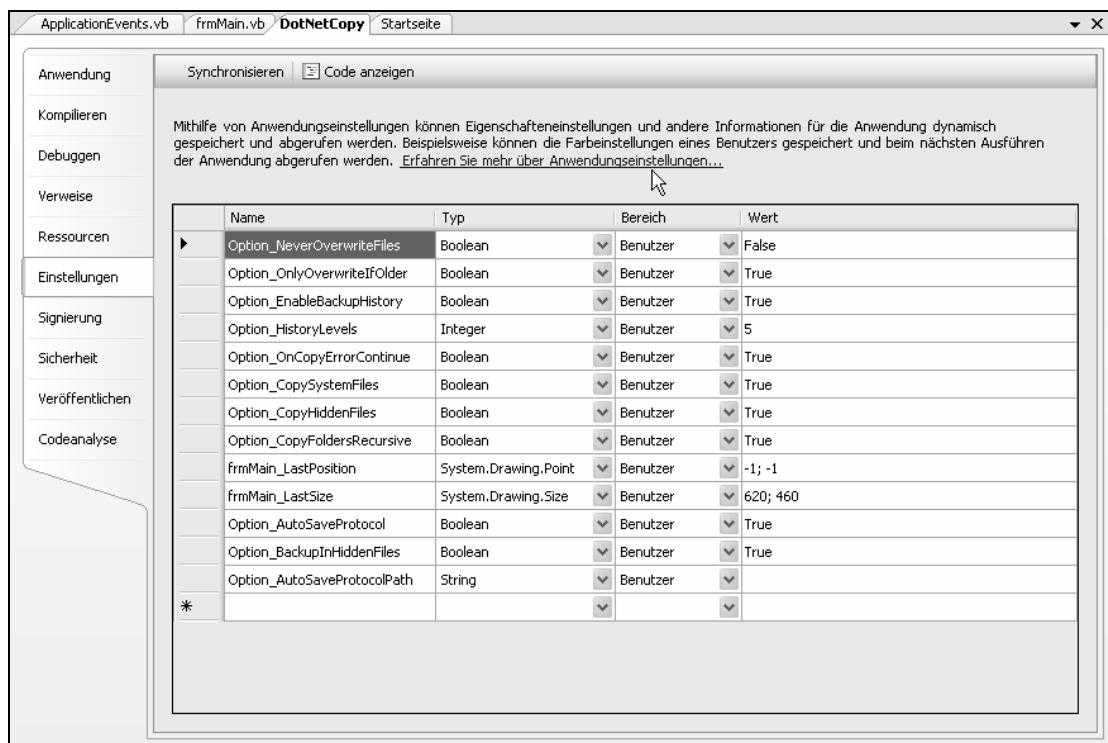
'und los geht's!
My.Application.MainForm = locFrmMain
End If
End Sub
End Class

```

## Verwenden von Anwendungseinstellungen mit My.Settings

DotNetCopy verwendet Anwendungseinstellungen (*ApplicationSettings*), um die Einstellungen auch nach dem Beenden des Programms zu erhalten, die der Anwender vornehmen kann, wenn er aus dem Menü *Extras* den Menüpunkt *Optionen* auswählt. Da schon ► Kapitel 3 sich intensiv mit Anwendungseinstellungen auseinandergesetzt hat, erfolgt an dieser Stelle lediglich eine kompakte Zusammenfassung des dazu Bekannten:

Anwendungseinstellungen pflegen Sie – ähnlich wie die Zeichenkettentabellen der Ressourcen – über eine Tabelle, die Sie über die Projekteigenschaften erreichen.



**Abbildung 25.5:** Anwendungseinstellungen, die Sie über *My.Settings* abrufen können, verwalten Sie wie Ressourcen in den Projekteigenschaften unter der Registerkarte *Einstellungen*

Für jeden Namen, den Sie in der Tabelle anlegen, wird dabei eine über My.Settings zu erreichende Objektvariable von dem Typ angelegt, den Sie in der Tabelle unter Typ bestimmt haben. Sie können für die Variablen, die Sie an dieser Stelle anlegen, aus zwei Bereichen auswählen: *Benutzer* wählen Sie, wenn Sie den Variableninhalt zur Laufzeit verändern wollen, und der neue Inhalt nach Programmende auch automatisch gesichert werden soll; *Anwendung* wählen Sie, wenn es sich um eine Konstante handeln soll, die nur Sie zur Entwurfszeit in den Projekteigenschaften (in der in Abbildung 25.5 gezeigten Tabelle) einstellen können, die man aber zur Laufzeit nicht verändern darf.

Das Abspeichern oder Auslesen von Anwendungseinstellungen ist schließlich eine Kleinigkeit. Der folgende Code zeigt, wie von Anwendungseinstellungen intensiv Gebrauch gemacht wird, wenn der Anwender im Optionsdialog Änderungen vornimmt. Die folgende Routine, die aufgerufen wird, wenn der Optionsdialog geladen wird, sorgt zunächst dafür, dass die Steuerelemente die Anwendungseinstellungen widerspiegeln:

```
Private Sub frmOptions_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
    'Anwendungseinstellungen im Dialog widerspiegeln lassen:
    'Auch dazu wird My.Settings verwendet.
    chkCopyFoldersRecursive.Checked = My.Settings.Option_CopyFoldersRecursive
    chkCopyHiddenFiles.Checked = My.Settings.Option_CopyHiddenFiles
    chkCopySystemFiles.Checked = My.Settings.Option_CopySystemFiles
    chkEnableBackupHistory.Checked = My.Settings.Option_EnableBackupHistory
    nudHistoryLevels.Value = My.Settings.Option_HistoryLevels
    chkNeverOverwriteFiles.Checked = My.Settings.Option_NeverOverwriteFiles
    chkOnCopyErrorContinue.Checked = My.Settings.Option_OnCopyErrorContinue
    chkOnlyOverwriteIfOlder.Checked = My.Settings.Option_OnlyOverwriteIfOlder
    chkBackupInHiddenFiles.Checked = My.Settings.Option_BackupInHiddenFiles
    chkAutoSaveProtocol.Checked = My.Settings.Option_AutoSaveProtocol
    lblProtocolPath.Text = My.Settings.Option_AutoSaveProtocolPath
End Sub
```

Beim Verlassen des Dialogs mit OK erfolgt der umgekehrte Weg: Die Werte oder Zustände der Steuerelemente werden ausgelesen und in den Anwendungseinstellungen gespeichert.

```
Private Sub btnOK_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnOK.Click
    'Die Einstellungen des Options-Dialogs in den Anwendungseinstellung speichern.
    'Auch dazu wird My.Settings verwendet.
    My.Settings.Option_CopyFoldersRecursive = chkCopyFoldersRecursive.Checked
    My.Settings.Option_CopyHiddenFiles = chkCopyHiddenFiles.Checked
    My.Settings.Option_CopySystemFiles = chkCopySystemFiles.Checked
    My.Settings.Option_EnableBackupHistory = chkEnableBackupHistory.Checked
    My.Settings.Option_HistoryLevels = CInt(nudHistoryLevels.Value)
    My.Settings.Option_NeverOverwriteFiles = chkNeverOverwriteFiles.Checked
    My.Settings.Option_OnCopyErrorContinue = chkOnCopyErrorContinue.Checked
    My.Settings.Option_OnlyOverwriteIfOlder = chkOnlyOverwriteIfOlder.Checked
    My.Settings.Option_BackupInHiddenFiles = chkBackupInHiddenFiles.Checked
    My.Settings.Option_AutoSaveProtocol = chkAutoSaveProtocol.Checked
    My.Settings.Option_AutoSaveProtocolPath = lblProtocolPath.Text

    'Das Setzen des Dialogergebnisses bei einem modal aufgerufenen Dialog
    'bewirkt gleichzeitig, dass dieser geschlossen wird!
    Me.DialogResult = Windows.Forms.DialogResult.OK
End Sub
```

## Speichern von Anwendungseinstellungen mit dem Bereich Benutzer

Den Code zum Abspeichern der Anwendungseinstellungen werden Sie übrigens im Optionsdialog vergebens suchen – es gibt ihn nicht, da er im Grunde genommen nicht notwendig ist. Wenn eine Windows Forms-Anwendung beendet wird, für die sowohl das Anwendungsframework als auch die Option *Eigene Einstellungen beim Herunterfahren speichern* in den Projekteigenschaften unter Anwendung aktiviert ist (siehe auch ► Kapitel 26), dann regelt das Framework das Speichern der Anwendungseinstellungen für den Bereich *Benutzer* automatisch.

In einigen Fällen kann es aber sinnvoll sein, die Anwendungseinstellungen selbst oder schon vor dem Beenden des Programms zu speichern. In DotNetCopy ist das beispielsweise der Fall, wenn das Programm zum allerersten Mal gestartet wird. In diesem Fall werden die Einstellungen für die Position und die Größe des Hauptformulars festgelegt, und direkt gespeichert – und das erfolgt mit der Methode `My.Settings.Save`, wie das folgende Beispiel zeigt:

```
Private Sub frmMain_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
    .
    .
    .
    ' Aus Platzgründen ausgelassen
    .
    .
    'Fenstergröße und -position wiederherstellen, wenn
    'nicht im Silent-Modus
    If Not mySilentMode Then
        'Beim ersten Aufruf ist die Größe -1, dann bleibt die Default-
        'Position, die Settings werden aber sofort übernommen...
        If My.Settings.frmMain_LastPosition.X = -1 Then
            My.Settings.frmMain_LastPosition = Me.Location
            Me.Size = My.Settings.frmMain_LastSize
            My.Settings.Save()

        'anderenfalls werden die Settings-Einstellungen in die Formular-
        'Position übernommen
        Else
            Me.Location = My.Settings.frmMain_LastPosition
        End If
        Me.Size = My.Settings.frmMain_LastSize
    End If
```



# 26 Das Anwendungsframework

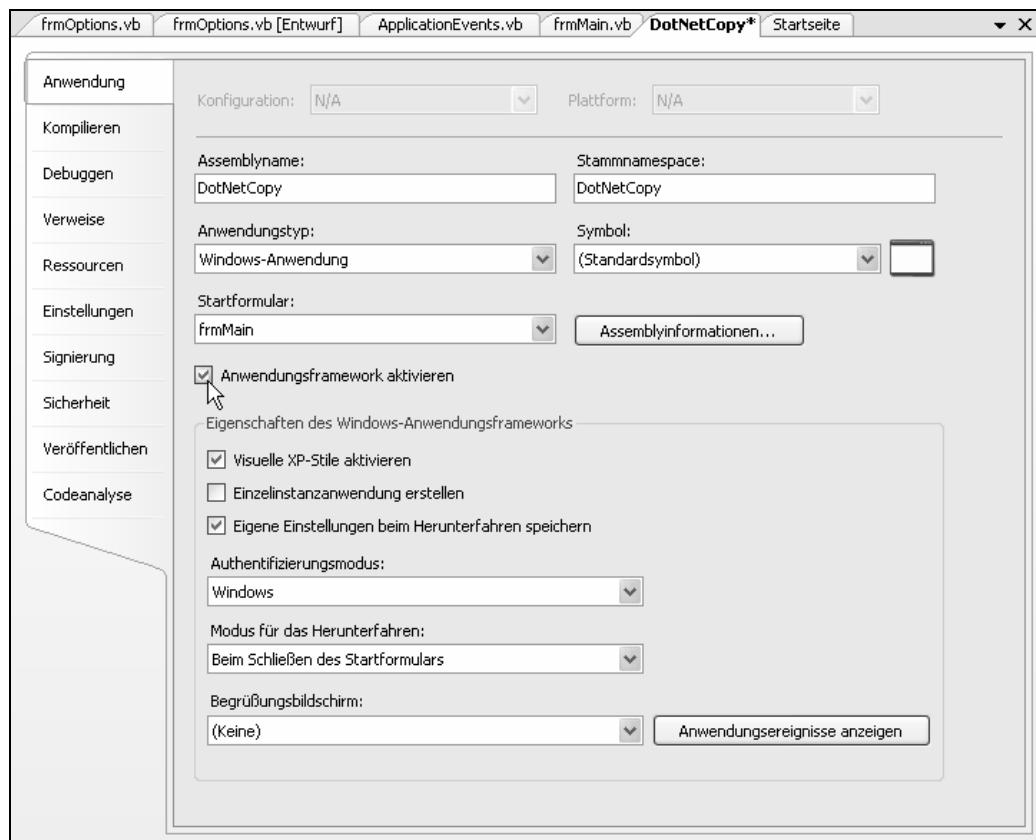
---

- 
- 733 **Die Optionen des Anwendungsframeworks**
  - 734 **Eine Codedatei implementieren, die Anwendungsereignisse (Starten, Beenden, Netzwerkzustand, generelle Fehler) behandelt**
- 

Das, was Microsoft als »Anwendungsframework« bezeichnet, sind eine Reihe von Funktionalitäten, die unter anderem auch dafür verantwortlich sind, dass Anwendungseinstellungen nach dem Beenden eines Programms abgespeichert werden oder auch das Anwendungsereignisse zur Verfügung gestellt werden (siehe ► Abschnitt »Eine Codedatei implementieren, die Anwendungsereignisse (Starten, Beenden, Netzwerkzustand, generelle Fehler) behandelt« ab Seite 734).

Um das zu erreichen, bedient sich Visual Basic 2005 übrigens zwei Mitteln: Zum einen ergänzt der Visual Basic-Compiler Ihren Quellcode um weiteren Code, den Sie selbst aber nie zu Gesicht bekommen. Dieser Quellcode ist quasi fest im Compiler hinterlegt, und werden bestimmte Funktionalitäten benötigt, die Sie durch die Optionen des Anwendungsframework in den Projekteigenschaften für Ihr Projekt aktivieren, ergänzt der Compiler dafür während des Kompilierens den entsprechenden Quellcode, der dann wiederum dafür sorgt, dass Ihr Programm die entsprechenden Funktionalitäten zur Verfügung stellt. Zum anderen benötigt Visual Basic 2005 dazu bestimmte Funktionsaufrufe, die durch die *VisualBasic.dll* des Frameworks zur Verfügung gestellt werden.

Für Windows-Anwendungen ist das Anwendungsframework standardmäßig aktiviert. Sie können die Aktivierung des Anwendungsframeworks in den Projekteigenschaften unter der Registerkarte Anwendung steuern (siehe Abbildung 26.1).



**Abbildung 26.1:** Erst durch das aktivierte Anwendungsframework stehen viele der hier beschriebenen Funktionalitäten zur Verfügung. Bei Windows Forms-Anwendungen ist das Anwendungsframework standardmäßig aktiviert.

Wenn das Kontrollkästchen *Anwendungsframework aktivieren* aktiviert ist, verwendet die Anwendung dann eine quasi »versteckte« Sub Main (die durch den versteckten Quellcode des Compilers in Ihr Projekt »gelangt«), mit der Ihre Windows-Anwendung startet, und die es erst ermöglicht, dass alle weiteren Optionen des Anwendungsframeworks funktionieren können. In diesem Fall müssen Sie ein Startformular auswählen, das von dieser versteckten Sub Main aufgerufen wird, wenn alle nötigen Vorbereitungen für das Zur-Verfügung-Stellen aller anderen Funktionen des Anwendungsframeworks abgeschlossen sind.

Wenn dieses Kontrollkästchen deaktiviert ist, verwendet die Anwendung die benutzerdefinierte Sub Main, die Sie unter Startformular angegeben haben, und die Sie in einem Modul Ihres Projektes platzieren müssen. In diesem Fall können Sie entweder ein Startobjekt (eine benutzerdefinierte Sub Main in einer Methode oder Klasse) oder ein Formular festlegen. Die Optionen im Bereich *Eigenschaften des Windows-Anwendungsframeworks* sind in diesem Fall nicht verfügbar.

# **Die Optionen des Anwendungsframeworks**

Die folgenden Einstellungen dienen der Konfiguration des Abschnitts Eigenschaften des Windows-Anwendungsframeworks. Diese Optionen sind nur verfügbar, wenn das Kontrollkästchen Anwendungsframework aktivieren aktiviert ist.

## **Windows XP Look and Feel für eigene Windows-Anwendungen – Visuelle XP-Stile aktivieren**

Aktivieren oder deaktivieren Sie die visuellen Windows XP-Stile, die auch Windows XP-Designs genannt werden. Die visuellen Windows XP-Stile lassen z. B. Steuerelemente mit gerundeten Ecken und dynamischen Farben zu. Die Option ist standardmäßig aktiviert.

## **Verhindern, dass Ihre Anwendung mehrfach gestartet wird – Einzelinstanzanwendung erstellen**

Aktivieren Sie dieses Kontrollkästchen, um zu verhindern, dass Benutzer mehrere Instanzen der Anwendung ausführen. Dieses Kontrollkästchen ist standardmäßig deaktiviert, wodurch mehrere Instanzen der Anwendung ausgeführt werden können.

## **MySettings-Einstellungen automatisch sichern – Eigene Einstellungen beim Herunterfahren speichern**

Durch das Aktivieren dieses Kontrollkästchens wird festgelegt, dass die My.Settings-Einstellungen der Anwendung beim Herunterfahren gespeichert werden. Die Option ist standardmäßig aktiviert. Wenn diese Option deaktiviert wird, können Sie diese Festlegung durch Aufrufen von My.Settings.Save manuell durchführen.

## **Bestimmen, welcher Benutzer-Authentifizierungsmodus verwendet wird**

Wählen Sie Windows (Standard) aus, um die Verwendung der Windows-Authentifizierung für die Identifikation des gerade angemeldeten Benutzers festzulegen. Diese Informationen können zur Laufzeit übrigens mit dem My.User-Objekt abgerufen werden. Wählen Sie Anwendungsdefiniert aus, wenn Sie eigenen Code anstatt der Windows-Standardauthentifizierungsmethoden für die Authentifizierung von Benutzern verwenden möchten.

## Festlegen, wann eine Anwendung »zu Ende« ist – Modus für das Herunterfahren

Wählen Sie *Beim Schließen des Startformulars (Standard)* aus, um festzulegen, dass die Anwendung beendet wird, wenn das als Startformular festgelegte Formular geschlossen wird. Dies gilt auch, wenn andere Formulare geöffnet sind. Wählen Sie *Beim Schließen des letzten Formulars* aus, um festzulegen, dass die Anwendung beendet wird, wenn das letzte Formular geschlossen oder wenn `My.Application.Exit` oder die End-Anweisung explizit aufgerufen wird.

## Einen Splash-Dialog beim Starten von komplexen Anwendungen anzeigen – Begrüßungsdialog

Wählen Sie das Formular aus, das als Begrüßungsbildschirm angezeigt werden soll. Zuvor müssen Sie einen Begrüßungsbildschirm mithilfe eines Formulars oder einer Vorlage erstellt haben.

Mehr zum Thema Splash-Dialoge finden Sie im nächsten Kapitel, ► Kapitel 27.

## Eine Codedatei implementieren, die Anwendungsereignisse (Starten, Beenden, Netzwerkzustand, generelle Fehler) behandelt

In den vorangegangenen Abschnitten haben Sie schon hier und da am Rande mitbekommen können, dass es offensichtlich – anders als noch in Visual Basic 2002 und 2003 – möglich sein muss, bestimmte Anwendungsereignisse »mithören« zu können. DotNetCopy klingt sich so beispielsweise in den Programmstart ein, und steuert das weitere Schicksal des Programms aufgrund der übergebenen Optionen sogar.

Wenn bei bestimmten Anwendungsereignissen Ihr eigener Code ausgeführt werden soll, müssen Sie dazu eine kleine Vorbereitung treffen:

- Rufen Sie dazu die Eigenschaften Ihres Projektes auf, und wählen Sie die Registerkarte *Anwendung*.
- Klicken Sie auf die Schaltfläche *Anwendungsereignisse anzeigen* (siehe Abbildung 26.1).
- Die Visual Studio-IDE fügt Ihrem Projekt nun eine Codedatei namens *ApplicationEvents.vb* hinzu und öffnet diese im Editor.
- Um den Funktionsrumpf für die Behandlungsroutine in eine der fünf existierenden Anwendungsereignisse einzufügen, wählen Sie im Codeeditor aus der linken Aufklappliste am oberen Rand des Editors zunächst den Eintrag `myApplicationEvents`.
- Wählen Sie anschließend in der rechten Liste das Ereignis, das Sie behandeln wollen, und für das der Funktionsrumpf in den Editor eingefügt werden soll. Orientieren Sie sich dabei auch an Abbildung 26.2.

```

DotNetCopy ApplicationEvents.vb frmMain.vb* Startseite
( MyApplication-Ereignisse ) Startup
Partial Friend Class M
    Private Sub MyApp1()
        ' Das Verzeichnis für die Optionen erstellen
        If String.IsNullOrEmpty(My.Settings.OptionPath) Then
            My.Settings.Option_AutoSaveProtocolPath = My.Computer.FileSystem.SpecialDirectories.MyDocuments & "MyApp"
            Dim locDi As New DirectoryInfo(My.Settings.OptionPath)
            ' Überprüfen und falls nötig erstellen
            If Not locDi.Exists Then
                ' im Bedarfsfall anlegen
                locDi.Create()
            End If
            ' Settings speichern
            My.Settings.Save()
        End If
    End Sub

```

**Abbildung 26.2:** Wählen Sie zum Einfügen des Rumpfs der Ereignisbehandlungsroutine eines der fünf existierenden Ereignisse aus der Aufklappliste aus

- Formulieren Sie den Behandlungscode aus.

**HINWEIS:** Beachten Sie auch, dass einige Ereignisse besondere Ableitungen der EventArgs-Klasse als Instanz übergeben, mit denen Sie das weitere Programmverhalten beeinflussen können. Die folgende Tabelle verrät mehr zu den vorhandenen Ereignissen, und wie Sie in ihren Rahmen den weiteren Programmverlauf beeinflussen können:

Ereignisname	Beschreibung	Steuerungsmöglichkeiten
Startup	Wird beim Starten der Anwendung noch vor dem Erstellen des Startformulars und dessen Binden an den Anwendungskontext ausgelöst.	Ja: Der Start der Anwendung kann durch Setzen von Cancel der StartupEventArgs-Instanz verhindert werden. In CommandLineArgs dieser Instanz werden ferner die Befehlszeilen-Argumente an die Anwendung als String-Array übergeben.
Shutdown	Wird nach dem Schließen aller Anwendungsformulare ausgelöst. Dieses Ereignis wird nicht ausgelöst, wenn die Anwendung nicht normal beendet wird, also beispielsweise durch einen Fehler, der eine Ausnahme ausgelöst hat.	Nein.

Ereignisname	Beschreibung	Steuerungsmöglichkeiten
StartUpNextInstance	Wird beim Starten einer Einzelinstanzanwendung ausgelöst, wenn diese bereits aktiv ist. <b>HINWEIS:</b> Sie bestimmen, dass eine Anwendung nur in einer Instanz (also nicht mehrmals gleichzeitig) gestartet werden darf, wenn Sie in den Projekteigenschaften Ihres Windows Forms-Projektes auf der Registerkarte <i>Anwendungen</i> die Option <i>Einzelinstanzanwendung erstellen</i> aktivieren.	Ja: Mit dem Setzen der BringToForeground-Eigenschaft der StartupNextInstanceEventArgs-Instanz können Sie festlegen, dass nach Verlassen der Ereignisbehandlungsroutine die schon laufende Instanz Ihrer Anwendung wieder zur aktiven Anwendung wird. In CommandLineArgs dieser Instanz werden ferner die Befehlszeilen-Argumente an die Anwendung als String-Array übergeben (und zwar für die, die zum erneuten Startversuch und damit auch zum Auslösen des Ereignisses führen).
NetworkAvailabilityChanged	Wird beim Herstellen oder Trennen der Netzwerkverbindung ausgelöst.	Nein. Sie können jedoch mit der Nur-Lesen-Eigenschaft IsNetworkAvailable abfragen, ob das Netzwerk zur Verfügung steht oder nicht.
UnhandledException	Wird ausgelöst, wenn in der Anwendung auf Grund eines nicht behandelten Fehlers eine unbehandelte Ausnahme auftritt.	Ja: Das Beenden der Anwendung aufgrund des Fehlers kann durch Setzen von Cancel der UnhandledEventArgs-Instanz verhindert werden. Darüber hinaus steht Ihnen durch die Exception-Eigenschaft dieser Instanz ein Exception-Objekt zur Verfügung, das Auskunft über die aufgetretene unbehandelte Ausnahme gibt.

Der folgende Beispielcode demonstriert den Umgang mit den Anwendungseignissen:

```

Private Sub MyApplication_NetworkAvailabilityChanged(ByVal sender As Object, _
    ByVal e As Microsoft.VisualBasic.Devices.NetworkAvailableEventArgs) _
    Handles Me.NetworkAvailabilityChanged
    MessageBox.Show("Die Netzwerkverfügbarkeit hat sich geändert!" & _
        "Das Netzwerk ist " & IIf(e.IsNetworkAvailable, "verfügbar", "nicht verfügbar").ToString)
End Sub

Private Sub MyApplication_Shutdown(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles Me.Shutdown
    Debug.Print("Die Anwendung wird jetzt beendet!")
End Sub

Private Sub MyApplication_StartupNextInstance(ByVal sender As Object, _
    ByVal e As Microsoft.VisualBasic.ApplicationServices.StartupNextInstanceEventArgs) _
    Handles Me.StartupNextInstance
    MessageBox.Show("Sie können keine 2. Instanz dieser Anwendung starten!")
    e.BringToForeground = True
End Sub

Private Sub MyApplication_UnhandledException(ByVal sender As Object, _
    ByVal e As Microsoft.VisualBasic.ApplicationServices.UnhandledEventArgs) _
    Handles Me.UnhandledException

```

```

'Eine bislang nicht behandelte Ausnahme ist aufgetreten!
Dim locDr As DialogResult = _
    MessageBox.Show("Eine unbehandelte Ausnahme ist aufgetreten!" & _
        vbNewLine & "Soll die Anwendung beendet werden?", _
        "Unbehandelte Ausnahme", MessageBoxButtons.YesNo, _
        MessageBoxIcon.Error, MessageBoxDefaultButton.Button1)
If locDr = DialogResult.No Then
    e.ExitApplication = False
End If
End Sub

```

Ein dokumentiertes Beispiel für das Ereignis StartUp, das beim Starten einer Windows Forms-Anwendung ausgelöst wird, finden Sie auch im Beispielprogramm DotNetCopy – der Vollständigkeit halber finden Sie im Folgenden den kompletten Code dieser Ereignisbehandlungsroutine:

---

**BEGLEITDATEIEN:** Viele der hier gezeigten Features können Sie sich direkt am Beispiel von DotNetCopy (vorgestellt in ► Kapitel 24) anschauen. Sie finden dieses Projekt unter dem Namen *DotNetCopy.sln* im Verzeichnis *.\VB 2005 - Entwicklerbuch\ F - Vereinfachung\Kap25\DotNetCopy*. So Sie ► Kapitel 24 noch nicht gelesen haben, sollten Sie es zunächst tun, um das Beispiel besser nachzuvollziehen zu können.

---

```

Private Sub MyApplication_Startup(ByVal sender As Object,
    ByVal e As Microsoft.VisualBasic.ApplicationServices.StartupEventArgs) Handles Me.Startup

    'Das Verzeichnis für die Protokolldatei beim ersten Mal setzen...
    If String.IsNullOrEmpty(My.Settings.Option_AutoSaveProtocolPath) Then
        My.Settings.Option_AutoSaveProtocolPath = _
            My.Computer.FileSystem.SpecialDirectories.MyDocuments & "\DotNetCopy Protokolle"
        Dim locDi As New DirectoryInfo(My.Settings.Option_AutoSaveProtocolPath)

        'Überprüfen und
        If Not locDi.Exists Then
            'im Bedarfsfall anlegen
            locDi.Create()
        End If

        'Settings speichern
        My.Settings.Save()
    End If

    Dim locFrmMain As New frmMain

    'Kommandozeile auslesen
    If My.Application.CommandLineArgs.Count > 0 Then
        For Each locString As String In My.Application.CommandLineArgs

            'Alle unnötigen Leerzeichen entfernen und
            'Groß-/Kleinschreibung 'Unsensibilisieren'
            'HINWEIS: Das funktioniert nur in der Windows-Welt;
            'kommt die Kopierlistendatei von einem Unix-Server, bitte darauf achten,
            'dass der Dateiname dafür auch komplett in Großbuchstaben gesetzt ist,
            'da Unix- (und Linux-) Derivate Groß-/Kleinschreibung berücksichtigen!!!
        Next
    End If

```

```

locString = locString.ToUpper.Trim

If locString = "/SILENT" Then
    locFrmMain.SilentMode = True
End If

If locString.StartsWith("/AUTOSTART") Then
    locFrmMain.AutoStartCopyList = locString.Replace("/AUTOSTART:", "")
    locFrmMain.AutoStartMode = True
End If
Next
End If

'Silentmode bleibt nur "an", wenn AutoStart aktiv ist.
locFrmMain.SilentMode = locFrmMain.SilentMode And locFrmMain.AutoStartMode

'Und wenn Silentmode, erfolgt keine Bindung des Formulars an den Anwendungskontext!
If locFrmMain.SilentMode Then
    'Alles wird in der nicht sichtbaren Instanz des Hauptforms durchgeführt,
    locFrmMain.HandleAutoStart()
    'und bevor das "eigentliche" Programm durch das Hauptformular gestartet wird,
    'ist der ganze Zauber auch schon wieder vorbei.
    e.Cancel = True
Else
    'Im Nicht-Silent-Modus wird das Formular an die Anwendung gebunden,
    'und los geht's!
    My.Application.MainForm = locFrmMain
End If
End Sub
End Class

End Namespace

```

# Teil G

## Entwickeln von SmartClient-Anwendungen

---

- 741 Programmieren mit Windows Forms**
  - 821 Im Motorraum von Formularen und Steuerelementen**
  - 863 GDI+ zum Zeichnen von Formular- und Steuerelementinhalten verwenden**
  - 893 Entwickeln von Steuerelementen**
  - 933 Mehreres zur gleichen Zeit erledigen – Threading in .NET**
  - 983 SQL Server 2005 und ADO.NET**
- 

In den vergangenen Kapiteln haben Sie viele Techniken und Konzepte von Visual Basic und dem .NET-Framework kennen gelernt. Für das Entwickeln von Anwendungen unter Windows ist das Beherrschen dieser Techniken ein notwendiges Muss, aber sicherlich nicht alles. In diesem Teil geht es darum, vieles des Gelernten in einen sinnvollen Kontext zu setzen, um die Voraussetzung für das Entwickeln regelrechter Windows-Anwendungen bzw. – wie man sie heutzutage so gerne nennt – SmartClient-Anwendungen<sup>1</sup> zu schaffen. Das anschließende Kapitel – Programmieren von Windows Forms-Anwendungen – kümmert sich in erster Linie darum.

Fortgeschrittene Techniken, wie das Entwickeln von Steuerelementen, der Einsatz von GDI+, Threading und ADO.NET, runden diesen letzten Teil des Buches ab.

Übrigens: Für das Verstehen dieses Kapitels hilft es, wenn Sie ► Kapitel 3 durchgearbeitet haben, damit Ihnen der Aufbau von Formularen und der Einsatz der wichtigen Steuerelemente klar ist, sowie ► Kapitel 26, das das Anwendungsframework von Visual Basic 2005 für Windows-Anwendungen erklärt. Das Verständnis von Klassen und Schnittstellen, wie es der Klassenteil dieses Buches vermittelt, ist ebenfalls von großem Vorteil.

---

<sup>1</sup> Wobei »SmartClient« nicht nur impliziert, dass sie – im Gegensatz zu vielen Web-gestützten Anwendungen – über eine eigene Intelligenz verfügen – also »smart« sind – sondern auch mit Datenquellen verbunden werden können, wie beispielsweise Web Services oder SQL Server-Datenbanken, dabei aber auch offline, also unverbunden, ihren Dienst verrichten. Frei interpretiert kann man daher jede Windows Forms-Anwendung, die in irgendeiner Form Daten mit einem entfernten System austauscht, als SmartClient-Anwendung ansehen.



# 27 Programmieren mit Windows Forms

---

- 743 **Strukturieren von Daten in Windows Forms-Anwendungen**
  - 754 **Formulare zur Abfrage von Daten verwenden – Aufruf von Formularen aus Formularen**
  - 766 **Überprüfung auf richtige Benutzereingaben in Formularen**
  - 771 **Anwendungen über die Tastatur bedienbar machen**
  - 775 **Über das »richtige« Schließen von Formularen**
  - 777 **Grundsätzliches zum Darstellen von Daten aus Auflistungsklassen in Steuerelementen**
  - 777 **Darstellen von Daten aus Auflistungen im ListView-Steuerelement**
  - 782 **Verwalten von Daten aus Auflistungen mit dem DataGridView-Steuerelement**
  - 805 **Entwickeln von MDI-Anwendungen**
  - 813 **Vererben von Formularen**
- 

Zentraler Bestandteil einer jeden SmartClient-Anwendung sind Windows Formulare – im .NET-Jargon kurz *WinForms* genannt. Jedes Windows Formular dient als Träger für Steuerelemente, mit denen der Anwender eine SmartClient-Anwendung letzten Endes bedienen kann. Dabei gibt es im Grunde genommen nur drei Arten von Formularen, die in Ihren Programmen zum Einsatz kommen:

- Formulare, die Daten darstellen,
- Formulare, die der Datenerfassung dienen, und
- Formulare, mit der Sie Programmoptionen einstellen.

Und eigentlich sind die Formulare, die unter die letzte Kategorie fallen, auch nichts anderes als Formulare die der Datenerfassung dienen.

Typische SmartClient-Anwendungen unter Windows haben dabei immer den gleichen Aufbau: Sie enthalten ein Hauptanwendungsfenster, das den eigentlichen Arbeitsbereich der Anwendung darstellt. Und von diesem Formular aus verzweigt der Funktionsbereich der Anwendung mithilfe weiterer Formulare, die entweder dazu dienen, weitere Daten (vielleicht anderen Typs) zu erfassen oder die Parameter bestimmter Funktionen einzustellen, die auf die Hauptdaten der Anwendung in irgendeiner Form angewendet werden (doch auch das ist ja auch wieder eine Art von Datenerfassung).

Unterscheiden kann man schließlich noch zwei Grundformen der Datenaufbereitung und Form der Verarbeitung durch den Anwender: Es gibt Anwendungstypen, die ihre Daten in mehrere Dokumente aufteilen, und in so genannten MDI-Anwendungen (*Multi Document Interface* – etwa: Mehrdokumenten-Benutzerschnittstelle) für den Anwender zur Bearbeitung zur Verfügung halten. Verschiedene Datensammlungen gleicher Art (Textdokumente, Bilder, Adresskarten, etc.) können dabei in verschiedenen untergeordneten Fenstern zur gleichen Zeit dargestellt und zur Bearbeitung angeboten werden.

Im Gegensatz dazu gibt es die häufiger vorkommenden SDI-Anwendungen (*Single Document Interface*-Anwendungen), bei denen die Daten eben nicht über mehrere Dokumentenfenster verteilt werden, sondern lediglich ein einzelnes Fenster den durch die Anwendung vorgegebenen Datentyp zur Bearbeitung anbietet.

Für welche Art der Benutzeroberfläche sich ein Entwickler entscheidet, ist eigentlich mehr Geschmackssache als funktionelle Notwendigkeit. Viele Grafikprogramme sind als MDI-Anwendungen ausgelegt – der Anwender kann also »gleichzeitig« mehrere Grafiken als untergeordnete Fenster öffnen; kann er sie aber wirklich gleichzeitig bearbeiten? Moderne Computer ließen das, was die Performance anbelangt, sicherlich zu. Die »Schwachstelle«, wenn man das als solches überhaupt bezeichnen kann, ist dabei eher der Anwender. Ich jedenfalls würde mir es nicht zutrauen, zwei Texte wirklich gleichzeitig zu schreiben, und Microsoft Word ist wahrscheinlich aus ähnlichen Überlegungen bestes Beispiel dafür, wie aus einer MDI-Anwendung (bis Word 97) sinnvollerweise eine SDI-Anwendung (ab Word 2000) werden kann.

Doch ganz gleich welchen Aufbau einer SmartClient-Anwendung Sie auch wählen und welche Art von Daten Ihre Anwendung auch bearbeitet: Entscheiden Sie sich für den Einsatz von Windows Formularen unter .NET, stehen Sie immer wieder vor der Lösung derselben Detailprobleme. Um nur einige zu nennen:

- Wie strukturieren SmartClient-Anwendungen am besten die Daten, die sie verwalten und dem Anwender zur Bearbeitung anbieten?
- Wie kann man Formulare aufrufen, die Parameter für andere Formulare »einholen«?
- Wie kann die Dateneingabe für den Anwender benutzerfreundlich erfolgen?
- Welche Techniken kann man anwenden, um eine Benutzeroberfläche ergonomischer zu gestalten und sie – getreu dem Motto: »das Auge isst mit« – ohne großen Aufwand verschönern?
- Wie geht man vor, um eine Benutzeroberfläche auch in anderen Sprachen anbieten zu können.

Dieses Kapitel setzt sich exakt mit dieser Problematik auseinander, und Sie sehen, dass es einen anderen Ansatz verfolgt, als Sie ihn vielleicht aus vielen anderen Büchern kennen.

Natürlich kann ein Windows Forms-Kapitel hunderte von Seiten füllen, indem es beschreibt, wie das 76. Steuerelement im Detail funktioniert und über welche Eigenschaften, Methoden und Ereignisse es verfügt. Doch damit sind Sie noch keinen Schritt weiter, denn es geht ja schließlich darum, das Gelernte im Kontext der Anwendungsentwicklung umzusetzen. Und außerdem können Sie sich diese Art von Information auch aus der Online-Hilfe von Visual Studio beschaffen.

Viel interessanter ist es doch, Lösungen für verschiedene, häufig auftretende Problemstellungen zu bekommen, und genau diesen Ansatz möchte dieses Kapitel verfolgen. Technische Details, die natürlich für das Verständnis des einen oder anderen Ansatzes notwendig sind, kommen in den jeweiligen Erklärungen natürlich nicht zu kurz.

# Strukturieren von Daten in Windows Forms-Anwendungen

Stellen Sie sich vor, Sie stehen vor der Aufgabe, eine kleine, einfache Adressenverwaltung zu entwickeln. Und um das Szenario wirklich einfach zu halten, gehen wir bei diesem Beispiel für den Moment davon aus, dass Sie die Daten nicht in einer Datenbank sondern in einer einfachen Datei speichern. Die Frage, die sich vielen als erstes stellt, lautet dann oftmals: Wie schaffe ich es, dass meine Anwendung von allen relevanten Stellen aus an die Daten herankommt? Und damit stehen viele bereits vor dem ersten wirklich schwerwiegenden Problem.

Das eigentliche Problem an dieser Stelle ist aber gar nicht die noch nicht bekannte Antwort. Im Grunde genommen ist es nämlich schon die Frage, die falsch gestellt wurde: Auch wenn sie die Klassenprogrammierung bereits aus dem Effeff kennen, fällt es vielen Entwicklern schwer, die OOP-Programmierung auch »im großen Rahmen« umzusetzen. Sie möchten nämlich, dass die Daten irgendwo zentral (und vor allen Dingen: global zugänglich) irgendwo in der Anwendung beherbergt werden. Und wenn, um beim Eingangsbeispiel zu bleiben, eine *Adressen-Bearbeiten*-Funktion aufgerufen wird, dann wird einfach ein neuer Dialog aufgerufen, der dann die Adressen entsprechend bearbeitet und dazu selbst auf den Gesamtdatenbestand der Anwendung zugreift. Soll eine neue Adresse erfasst werden, dann passiert der gleiche Datenzugriff eben aus dem »*Adresse-Neu-Anlegen*-Dialog« heraus. Jede Programmfunction greift also nach Gutdünken auf den Datenbestand zu. Das Chaos ist dabei buchstäblich vorprogrammiert.

Der Ansatz sollte ein anderer sein. In einer gelungenen Windows Forms-Anwendung sollte eine bestimmte Komponente die wesentlichen Daten, die der Anwender bearbeiten können soll, kapseln. Und nur diese Komponente sollte in der Lage sein dürfen, bestimmte Daten zum Bearbeiten »herauszurücken« oder Änderungen bestimmter Daten wieder entgegenzunehmen. Andere Komponenten stellen die Datenstrukturen selbst dar. Und wieder andere Komponenten übernehmen die Kommunikation mit dem Anwender.

Aus diesem Grund ist es bei der OOP-Programmierung gängige Praxis, Anwendungen in verschiedene Ebenen bzw. Schichten – (englische Layer oder Tier) zu unterteilen.

- Eine Schicht stellt dabei die Datenschicht bereit. Das sind in der Regel Klassen, die die eigentlichen Daten beherbergen.
- Eine andere Schicht stellt die Anwendungslogik (auch *Geschäftslogik* oder neudeutsch: *Business Logic*) bereit. Das sind wiederum Klassen, die die Klassen, die von der Datenschicht zur Verfügung gestellt werden, einbinden und kapseln und die Funktionalität zur Verfügung stellen, um die Daten im Sinne des Anwenders aufzubereiten und zu verarbeiten.
- Eine letzte Schicht stellt die Schnittstelle zwischen dem Benutzer und der Anwendungslogik her. Sie hat nur Delegatenfunktion, was in diesem Zusammenhang bedeutet: Sie nimmt Anwenderanforderungen entgegen und leitet sie an die Schicht weiter, die die Anwendungslogik enthält.

Die Einteilung in solche Schichten bringt den Entwicklern dann gleich mehrere Vorteile, die sich vor allen Dingen längerfristig auswirken:

- Verschiedene Ihrer Anwendungen können die einzelnen Komponenten bzw. Schichten, aus denen eine mehrschichtige Anwendung besteht, ebenfalls verwenden.

- Umfangreiche Anwendungen bzw. Anwendungen, die komplexe Datenstrukturen zu verwalten haben, lassen sich einfacher in kleinere Portionen aufteilen.
- Ihr Anwendung kann relativ schnell einem Facelift unterzogen werden: Ohne dass die eigentliche Anwendungslogik wirklich ersetzt werden müsste, können Sie ihr ein neues Make-up verpassen und sie sieht damit in Null Komma nichts neuer und moderner aus.
- Verschiedene Schichten können einfacher im Team entwickelt werden. Durch die Aufteilung einer Anwendung in Schichten bzw. Komponenten lassen sich komplexe Anwendungen viel besser im Team entwickeln.

Wenn Sie große Teile dieses Buches bereits studiert haben, dann sind Sie vielleicht schon einem Ansatz einer kleinen Adressverwaltung hier und da begegnet. Ging es in diesen Kapiteln noch darum, bestimmte Details an plausiblen Beispielen zu erklären (► Kapitel 20, generische Auflistungen, ► Kapitel 22, XML-Serialisierung), so will das folgende Beispiel diese technischen Puzzleteile aufgreifen, neu strukturieren, wieder zusammensetzen und daraus eine kleine, unkomplizierte aber dennoch vollständige Musteranwendung schaffen, die nicht nur prinzipiell und theoretisch zeigt, wie eine saubere OOP-Entwicklung einer SmartClient-Anwendung aussehen sollte.

---

**BEGLEITDATEIEN:** Sie finden das Projekt für das folgende Beispiel im Verzeichnis `.\VB 2005 - Entwicklerbuch\G - SmartClient\Kap27\Adresso01`.

---

So Sie sich die ► Kapitel 20 und 22 schon zu Gemüte geführt haben, werden Sie bei diesem Projekt in Sachen technischen Gegebenheiten zunächst nichts Neues entdecken. Dennoch unterscheiden sich Projekt als auch Code deutlich von der letzten Version dieses Beispiels aus ► Kapitel 22, und das wird bereits beim Betrachten des Projektmappen-Explorers deutlich, denn:



**Abbildung 27.1:** Schon im Projektmappen-Explorer wird deutlich, wie das Programm in drei Schichten eingeteilt ist

- Die Codedatei `AdressDaten.vb` enthält die Datenschicht der Anwendung – die Adresse-Klasse, die eine einzelne Adresse speichert.
- Die Codedatei `AdressLogik.vb` sorgt dafür, dass die einzelnen Adresse-Datensätze in einer Auflistung zusammengefasst und dort funktionell verwaltet werden. Und hier zeigt sich auch das erste Mal der Unterschied zwischen dieser Version von »Adresso« und der, die Sie noch aus ► Kapi-

tel 22 kennen. Die gesamte Funktionalität des Programms – beispielsweise zum Sortieren oder Suchen von Adressen – findet nun an dieser Stelle statt, und nicht mehr, wie in der »Vorgängerverision«, lustig verteilt mal bei der Benutzeroberflächenschicht in `frmMain` und mal, wie das Serialisieren, in der abgeleiteten Auflistung `Adressen`. Alle Aufgaben sind in dieser Version, so, wie es geplant war, sauber voneinander getrennt (wovon Sie sich im anschließenden Codelisting auch selber überzeugen können).

- In `frmMain.vb` schließlich, der dritten Schicht, finden nur noch Delegationsaufgaben statt. Funktionen, die der Anwender beispielsweise aus dem Pulldown-Menü abruft, werden einfach an die Adresslogik weitergereicht. Das Komplizierteste, was diese Benutzeroberflächenschicht noch leisten muss, ist die Darstellung der durch die Adresslogik verwalteten Adresse-Objekte im ListView-Steuerelement.

Übrigens: Das Umgestalten des Codes vom Beispiel aus ► Kapitel 22 zu der Version, wie Sie sie hier sehen, ist ein Vorgang, den man unter dem Begriff *Refactoring* versteht.

Die Codedateien sehen in dieser Version des Beispiels nun folgendermaßen aus:

### Datenschicht: AdressDaten.vb

`AdressDaten.vb` enthält nur noch die Datenstruktur zur Speicherung einer einzelnen Adresse in Form von Adresse-Klasse:

```
<Serializable()>
Public Class Adresse

    'Membervariablen, die die Daten halten:
    Protected myMatchcode As String
    Protected myName As String
    Protected myVorname As String
    Protected myStraße As String
    Protected myPLZ As String
    Protected myOrt As String
    Protected myGeburtsdatum As Date

    ''' <summary>
    ''' Erstellt eine neue Instanz dieser Klasse.
    ''' </summary>
    ''' <remarks>Ein parameterloser Konstruktor wird benötigt,
    ''' um XML-Serialisierung zu ermöglichen.</remarks>
    Sub New()
        MyBase.New()
    End Sub

    'Konstruktor - legt eine neue Instanz an
    Sub New(ByVal Matchcode As String, ByVal Name As String, ByVal Vorname As String, _
            ByVal Straße As String, ByVal Plz As String, ByVal Ort As String, _
            ByVal Geburtsdatum As Date)
        myMatchcode = Matchcode
        myName = Name
        myVorname = Vorname
        myStraße = Straße
        myPLZ = Plz
    End Sub

```

```

myOrt = Ort
myGeburtsdatum = Geburtsdatum
End Sub

Public Overridable Property Matchcode() As String
    Get
        Return myMatchcode
    End Get
    Set(ByVal Value As String)
        myMatchcode = Value
    End Set
End Property

Public Overridable Property Name() As String
    Get
        Return myName
    End Get
    Set(ByVal Value As String)
        myName = Value
    End Set
End Property

Public Overridable Property Vorname() As String
    Get
        Return myVorname
    End Get
    Set(ByVal Value As String)
        myVorname = Value
    End Set
End Property

Public Overridable Property Straße() As String
    Get
        Return myStraße
    End Get
    Set(ByVal value As String)
        myStraße = value
    End Set
End Property

Public Overridable Property PLZ() As String
    Get
        Return myPLZ
    End Get
    Set(ByVal Value As String)
        myPLZ = Value
    End Set
End Property

Public Overridable Property Ort() As String
    Get
        Return myOrt
    End Get

```

```

    Set(ByVal Value As String)
        myOrt = Value
    End Set
End Property

Public Overridable Property Geburtsdatum() As Date
    Get
        Return myGeburtsdatum
    End Get
    Set(ByVal Value As Date)
        myGeburtsdatum = Value
    End Set
End Property

Public Overrides Function ToString() As String
    Return Matchcode & ":" & Name & ", " & Vorname
End Function
End Class

```

Sie werden übrigens feststellen, jedenfalls wenn Sie das Beispiel aus ► Kapitel 22 kennen, dass in dieser Version ein weiteres Adressdatenfeld hinzugekommen ist – auch wenn es nicht in der Adressenliste des Formulars zu sehen ist. Das Geburtsdatum ist nicht nur ein wesentlicher Bestandteil jeder noch so kleinen Adressverwaltung – dieses Feld eignet sich insbesondere auch zur Demonstration von Eingabeüberprüfungen bei neuen Adressen, mit dem sich der nächste Abschnitt beschäftigt.

---

**TIPP:** Wenn Sie wissen wollen, wie im Programm die dazugehörigen »Zufallsdatums« erzeugt werden, werfen Sie einen Blick in die Codedatei *AdressLogik.vb* und dort in die Funktion Zufallsadressen. Die Erklärung finden Sie in den Kommentaren am Ende dieser Prozedur.

---

### Anwendungslogik: AdressLogik.vb

Alle Funktionen der Anwendung sind in dieser Codedatei zusammengefasst. Sie bildet damit den Code, der am intensivsten refaktoriert wurde. Die Änderungen im Vergleich zum Beispiel aus ► Kapitel 22 sind fett hervorgehoben.

```

<Serializable()>
Public Class Adressen
    Inherits List(Of Adresse)

    Private mySortierenNach As AdressenSortierenNach

    ''' <summary>
    ''' Erstellt eine neue Instanz dieser Klasse.
    ''' </summary>
    ''' <remarks></remarks>
    Sub New()
        MyBase.New()
    End Sub

```

```

'<summary>
'<summary>
'<param name="collection">Die generische Adresse-Auflistung, deren Elemente
'<param name="Text">Suchbegriff, nachdem diese Auflistung durchsucht werden soll.</param>
'<remarks></remarks>
Sub New(ByVal collection As ICollection(Of Adresse))
    MyBase.New(collection)
End Sub

'<summary>
'<summary>
'<param name="Text">Suchbegriff, nachdem diese Auflistung durchsucht werden soll.</param>
'<returns></returns>
'<remarks></remarks>
Public Function Suchen(ByVal Text As String) As Adressen

    'In dieser Adressen-Auflistung wird das Suchergebnis gesammelt.
    Dim locAdressen As New Adressen()

    For Each locAdresse As Adresse In Me
        If locAdresse.Name Like Text Then
            locAdressen.Add(locAdresse)
            'Direkt zum 'Next' springen.
            Continue For
        ElseIf locAdresse.Vorname Like Text Then
            locAdressen.Add(locAdresse)
            Continue For
        ElseIf locAdresse.Straße Like Text Then
            locAdressen.Add(locAdresse)
            Continue For
        ElseIf locAdresse.Ort Like Text Then
            locAdressen.Add(locAdresse)
            Continue For
        ElseIf locAdresse.PLZ.ToString Like Text Then
            locAdressen.Add(locAdresse)
        End If
    Next

    Return locAdressen
End Function

```

Die Suchenfunktion wurde im Gegensatz zum Beispiel aus ► Kapitel 22 auch funktionell abgeändert: Sie findet nun nicht nur die erste Adresse, auf die der Suchbegriff zutraf, sondern liefert eine Adressen-Auflistung zurück, die alle Adressen enthält, in denen der Suchbegriff vorkam. Darüber hinaus findet es den Suchbegriff nicht nur in einer bestimmten einstellbaren Datenspalte, sondern in der gesamten Adresszeile.

```

Public Sub Sortieren(ByVal SortierenNach As AdressenSortierenNach)
    'Das Array mithilfe eines Comparison-Delegaten sortieren
    mySortierenNach = SortierenNach
    Me.Sort(New Comparison(Of Adresse)(AddressOf AdressenVergleicher))

End Sub

'Der Comparison-Delegat zum Vergleichen zweier Elemente.
Private Function AdressenVergleicher(ByVal x As Adresse, ByVal y As Adresse) As Integer

    Try
        'Sortierung in Abhängigkeit zur Suchspalte durchführen
        Select Case mySortierenNach
            Case AdressenSortierenNach.Matchcode
                Return x.Matchcode.CompareTo(y.Matchcode)
            Case AdressenSortierenNach.Name
                Return x.Name.CompareTo(y.Name)
            Case AdressenSortierenNach.Ort
                Return x.Ort.CompareTo(y.Ort)
            Case AdressenSortierenNach.PLZ
                Return x.PLZ.CompareTo(y.PLZ)
            Case AdressenSortierenNach.Straße
                Return x.Straße.CompareTo(y.Straße)
            Case Else
                Return x.Vorname.CompareTo(y.Vorname)
        End Select
        Catch ex As Exception
            'Afangen, dass einer der Suchstrings Nothing ist.
            'Der ist dann immer kleiner als alle anderen!
            Return -1
        End Try
    End Function

''' <summary>
''' Serialisiert alle Elemente dieser Auflistung in eine XML-Datei.
''' </summary>
''' <param name="Dateiname">Der Dateiname der XML-Datei.</param>
''' <remarks></remarks>
Sub XMLSerialize(ByVal Dateiname As String)
    Dim locXmlWriter As New XmlSerializer(GetType(Adressen))
    Dim locXmlDatei As New StreamWriter(Dateiname)
    locXmlWriter.Serialize(locXmlDatei, Me)
    locXmlDatei.Flush()
    locXmlDatei.Close()
End Sub

''' <summary>
''' Generiert aus einer XML-Datei eine neue Instanz dieser Auflistungsklasse.
''' </summary>
''' <param name="Dateiname">Name der XML-Datei, aus der die Daten für
''' diese Auflistungsinstanz entnommen werden sollen.</param>
''' <returns></returns>
''' <remarks></remarks>

```

```

Public Shared Function XmlDeserialize(ByVal Dateiname As String) As Adressen
    Dim locXmlLeser As New XmlSerializer(GetType(Adressen))
    Dim locXmlDatei As New StreamReader(Dateiname)
    Return CType(locXmlLeser.Deserialize(locXmlDatei), Adressen)
End Function

''' <summary>
''' Liefert eine Instanz dieser Klasse mit Zufallsadressen zurück.
''' </summary>
''' <param name="Anzahl">Anzahl der Zufallsadressen, die erzeugt werden sollen.</param>
''' <returns></returns>
''' <remarks></remarks>
Public Shared Function ZufallsAdressen(ByVal Anzahl As Integer) As List(Of Adresse)
    'Aus Platzgründen ausgelassen
End Function
End Class

Public Enum AdressenSortierenNach
    Matchcode
    Name
    Vorname
    Straße
    PLZ
    Ort
End Enum

```

### Benutzeroberflächenschicht: frmMain.vb

Und schließlich gibt es die letzte Schicht der Anwendung, die, wie schon gesagt, nur noch Delegationsaufgaben erfüllt. Alle Funktionen, die das Formular durch entsprechende Ereignisse vom Benutzer entgegennimmt, leitet es direkt an die Klasse Adressen weiter, in der dann die eigentliche Problemlösung vollzogen wird.

Die Stellen, an denen es die Programmkontrolle zur eigentlichen Durchführung der »Geschäftslogik« an die Anwendungslogikschicht übergibt, sind im folgenden Listing fett markiert. Nicht relevante Codeauszüge sind aus Platzgründen ausgeblendet.

Auf einen funktionellen Unterschied zum Beispiel aus ► Kapitel 22 möchte ich in diesem Zusammenhang noch besonders hinweisen: Das Anklicken einer Adresszeile in der Liste bewirkt die Anzeige einiger Detaildaten in der Statuszeile des Formulars. Der entsprechende Code, der dieses Verhalten steuert, befindet sich ganz am Ende des dokumentierten Listings, das Sie im Folgenden finden.

```

Public Class frmMain

    Private myAdressen As Adressen          ' Alle Adressen

    'Dieses hier nicht als Ereignis einbinden - wäre von hinten, durch die Brust, ins Auge...
    'Besser überschreiben - frmListenForm ist schließlich von Form abgeleitet!
    Protected Overrides Sub OnLoad(ByVal e As System.EventArgs)
        'Basismethode aufrufen, damit das Formular machen kann,
        'was es machen muss, um alles einzurichten.
        MyBase.OnLoad(e)
    End Sub

```

```

'Instanzieren.
myAdressen = New Adressen

'Die Listview-Spalten einrichten.
Vorbereitungen()

'Die Adressenliste darstellen.
ElementeDarstellen()
End Sub

'Richtet lediglich die Listview ein.
Sub Vorbereitungen()
    ' Aus Platzgründen ausgelassen
End Sub

Sub ElementeDarstellen()

    'Unterdrückt Neuzeichnen-Ereignisse bis zum
    'nächsten EndUpdate; dadurch geht der Aufbau
    'der Elemente schneller und wackelt nicht.
    Me.lvAdressen.BeginUpdate()

    'Alle Elemente der ListView löschen.
    Me.lvAdressen.Items.Clear()

    'Für jedes Element der Liste wird ElementInListe aufgerufen.
    myAdressen.ForEach(New Action(Of Adresse)(AddressOf ElementInListe))

    'So werden die Spaltenbreiten optimal angepasst.
    For Each locCol As ColumnHeader In Me.lvAdressen.Columns
        locCol.Width = -2
    Next

    'Aufbau der ListView ist beendet.
    Me.lvAdressen.EndUpdate()
End Sub

'Der Action-Delegat: für jedes Element der Liste wird diese Aktion durchgeführt.
Sub ElementInListe(ByVal Element As Adresse)
    'Neues ListView-Element - Matchcode kommt zuerst.
    Dim locLvwItem As New ListViewItem(Element.Matchcode)

    'Die Untereinträge setzen
    With locLvwItem.SubItems
        .Add(Element.Name)
        .Add(Element.Vorname)
        .Add(Element.Straße)
        .Add(Element.PLZ)
        .Add(Element.Ort)
    End With
End Sub

```

```

'Zum Wiederfinden Referenz in Tag
locLvwItem.Tag = Element

'Zur Listview hinzufügen
lvwAdressen.Items.Add(locLvwItem)
End Sub

'Wird aufgerufen, wenn eine der Spalten angeklickt wird.
Private Sub lvwAdressen_ColumnClick(ByVal sender As Object, ByVal e As
System.Windows.Forms.ColumnEventArgs) Handles lvwAdressen.ColumnClick

'Spaltennummer, die in e.Column steht, in AdressenSortierenNach konvertieren
Dim locNach As AdressenSortierenNach = CType(e.Column, AdressenSortierenNach)

'Sortieren
myAdressen.Sortieren(locNach)

'Die Elemente neu sortiert darstellen
ElementeDarstellen()
End Sub

Private Sub tsmAdresseSuchen_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
Handles tsmAdresseSuchen.Click
Dim locSuchFormular As New frmSuchen
Dim locErsterGefundener As Boolean

'Den Suchbegriff merken, damit der Predicate-Delegat darauf
'zugreifen kann.
Dim locSuchbegriff As String = locSuchFormular.Suchbegriff
If String.IsNullOrEmpty(locSuchbegriff) Then
    Return
End If

'Alle gefundenen Elemente durchlaufen, und...
For Each locAdresse As Adresse In myAdressen.Suchen(locSuchbegriff)

    'jeweils alle ListView-Elemente durchsuchen und überprüfen, ob ...
    For Each locLvwItem As ListViewItem In Me.lvwAdressen.Items

        '... die Tag-Referenz der Referenz des gesuchten Objekts entspricht.
        If locLvwItem.Tag Is locAdresse Then

            'Gefunden! ListView-Element markieren,
            locLvwItem.Selected = True

            'und dafür sorgen, dass es der erste gefundene
            'Eintrag im sichtbaren Bereich liegt.
            If Not locErsterGefundener Then
                locLvwItem.EnsureVisible()
                locErsterGefundener = True
            End If
        End If
    Next
Next

```

```

        'Gefunden, wir müssen in der ListView
        'nicht weitersuchen!
        Exit For
    End If
    Next
End Sub

Private Sub tsmAdressenLöschen_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
Handles tsmAdressenLöschen.Click

If lvwAdressen.SelectedItems IsNot Nothing AndAlso lvwAdressen.SelectedItems.Count > 0 Then

    Dim locDr As DialogResult = MessageBox.Show(My.Resources.MB_Nachfrage_Löschen_Body, _
                                                My.Resources.MB_Nachfrage_Löschen_Titel, MessageBoxButtons.YesNo, _
                                                MessageBoxIcon.Question)
    If locDr = Windows.Forms.DialogResult.Yes Then
        For Each lwi As ListViewItem In lvwAdressen.SelectedItems
            myAdressen.Remove(DirectCast(lwi.Tag, Adresse))
        Next
        ElementeDarstellen()
    End If
End If
End Sub

Private Sub tsmZufallsadressenAnfügen_Click(ByVal sender As System.Object, _
                                              ByVal e As System.EventArgs) Handles tsmZufallsadressenAnfügen.Click
    ' Aus Platzgründen ausgelassen
End Sub

'Wird aufgerufen, wenn Anwender Datei/Adressliste speichern wählt.
Private Sub tsmAdresslisteSpeichern_Click(ByVal sender As System.Object, _
                                            ByVal e As System.EventArgs) Handles tsmAdresslisteSpeichern.Click

    Dim dateiSpeichernDialog As New SaveFileDialog

    With dateiSpeichernDialog
        'Ermitteln des Dateinamens aus Platzgründen ausgelassen

        'Adressen serialisieren
        myAdressen.Serialize(.FileName)
    End With
End Sub

'Wird ausgelöst, wenn der Anwender einen Eintrag oder mehrere Einträge der Liste selektiert
Private Sub lvwAdressen_SelectedIndexChanged(ByVal sender As System.Object, _
                                         ByVal e As System.EventArgs) Handles lvwAdressen.SelectedIndexChanged

If lvwAdressen.SelectedItems.Count = 0 Then

    'Kein Eintrag selektiert
    tsslAusgewählteAdresse.Text = "Keine Adresse selektiert"

```

```

ElseIf lvwAdressen.SelectedIndices.Count > 1 Then

    'Mehrere Einträge selektiert
    tsslAusgewählteAdresse.Text = "Mehrere Adressen selektiert"
Else

    'Genau ein Eintrag selektiert, dann die Grunddaten und das Geburtsdatum darstellen
    Dim locAdresse As Adresse

    'Das ursprüngliche Adresse-Objekt aus der Liste holen.
    'Dieses wurde beim Erstellen der ListViewItems in deren
    'Tag-Eigenschaften gespeichert, und auf diese Weise wird
    'die Relation zwischen Listeneintrag und eigentlichem Objekt hergestellt.
    locAdresse = DirectCast(lvwAdressen.SelectedItems(0).Tag, Adresse)

    'Text für die Darstellung in der Statuszeile zusammenbasteln:
    Dim locStatusText As String
    locStatusText = locAdresse.Matchcode & ": "
    locStatusText &= locAdresse.Name & ", " & locAdresse.Vorname & " - Geboren am "
    locStatusText &= locAdresse.Geburtsdatum.ToString("dd/MM/yyyy")

    'Text in der Statuszeile darstellen.
    tsslAusgewählteAdresse.Text = locStatusText
End If
End Sub

End Class

```

Wie Sie sehen, ist das Komplizierteste, was diese Schicht noch übernehmen muss, die Darstellung der Daten im ListView-Steuerelement. Für den vorhandenen Funktionsumfang ist der reine Formularcode aber nunmehr vergleichsweise kompakt. Genaueres über die Darstellung von Auflistungsklassen im ListView-Steuerelement erfahren Sie im ► Abschnitt »Darstellen von Daten aus Auflistungen im ListView-Steuerelement« ab Seite 777.

## Formulare zur Abfrage von Daten verwenden – Aufruf von Formularen aus Formularen

Die seit Erscheinen von Visual Basic .NET wohl häufigste Frage zu Windows Forms-Anwendungen, der man in den verschiedensten Newsgroups begegnet, lautet: »Wie rufe ich aus einem Formular ein Formular auf – beispielsweise um Daten für die Hauptanwendung vom Anwender zu erfragen?«

Zweifelsohne gibt es auf diese Frage nicht *die eine* Antwort; es gibt natürlich mehrere Wege, Formulare zu realisieren, die zum Abfragen von bestimmten Parametern dienen und diese an das aufrufende Formular zurückliefern.

Die Technik, die ich Ihnen im Folgenden vorstellen möchte, ist in Absprache mit vielen Entwicklerkollegen aber wohl eine der gängigsten. Sie hat den Vorteil, dass sie ein Muster darstellt, das Sie in Anwendungen, die diese Technik des »Dateneinsammelns« erfordern, immer wieder einsetzen können.

Bei den Realisierungsüberlegungen und zum späteren besseren Verständnis des Codes, der diese Aufgabe übernimmt, sollte man sich die folgenden Punkte vor Augen führen:

Formulare, die zum Einlesen bestimmter Parameter dienen (im schon bekannten Beispielprojekt könnte das zum Beispiel das Erfassen einer neuen Adresse oder das Editieren einer vorhandenen sein), sollten ...

- ... als modale Dialoge dargestellt werden (siehe nächster Abschnitt).
- ... Daten, mit denen das Formular vorbelegt werden muss, vom aufrufenden Formular entgegen nehmen können, und die durch den Anwender veränderten Daten wieder zurückliefern.
- ... möglichst als Funktion aufgerufen werden können, und selbst dafür sorgen, dass sie sich darstellen und schließlich wieder schließen.
- ... vor ihrem Schließen die Richtigkeit der durch den Anwender eingegebenen Daten überprüfen.

## Der Umgang mit modalen Formularen

Bevor wir uns mit dem Erstellen eines grundsätzlichen Musters für das Erfragen von Daten mit modalen Formularen beschäftigen, sollten wir uns für ein besseres Verständnis zunächst mit modalen Dialogen an sich beschäftigen. Es gibt hier nämlich einiges zu entdecken, was nicht allgemeinhin bekannt ist.

Modale Dialoge verhalten sich wie Meldungsfelder (MessageBox-Dialöge), was bedeutet, dass Sie die darunter liegenden Dialoge nicht erreichen können, solange der modale Dialog aktiv ist. Ein Anwender muss also den Dialog erst ausfüllen und bestätigen (oder alternativ abbrechen, wenn er den Dialog versehentlich aufgerufen hat), um mit der Anwendung weiterarbeiten zu können. Im Gegensatz dazu gibt es übrigens nicht-modale Dialoge, die sich auf gleicher Ebene mit anderen Fenstern befinden, also parallel zu anderen Fenstern bedient werden können.

### Modale Formulare haben eine eigene Nachrichtenwarteschlange

In Windows selbst funktioniert die Kommunikation zwischen verschiedenen Komponenten auf Basis so genannter Nachrichtenwarteschlangen (das nächste Kapitel hält dazu mehr bereit, falls Sie sich für die Details interessieren). Wenn Sie eine Windows Forms-Anwendung mit einem Hauptformular starten, dann richtet das Anwendungsframework für diese Anwendung genau eine dieser Nachrichtenwarteschlangen ein. Sobald ein Ereignis eintritt, wird dieses Ereignis windows-intern dieser Nachrichtenschlange hinzugefügt und letzten Endes von entsprechenden .NET-Klassen des Anwendungsframework bzw. des Hauptformulars der Anwendung bearbeitet.

Das ändert sich in dem Moment, in dem Sie einen modalen Dialog ins Leben rufen. In diesem Fall wird eine neue Nachrichtenwarteschlange für den modal dazustellenden Dialog eingerichtet. Der Programmablauf stoppt dann an der Stelle, an der Sie den entsprechenden modalen Dialog aufrufen:

```
Dim locFrm As New IrgeneinFormular  
locFrm.ShowDialog()  
'Hier geht es erst weiter, wenn der Dialog beendet wurde.
```

## Steuern von modalen Dialogen mit der DialogResult-Eigenschaft

Da sich modale Dialoge wie Meldungsfelder verhalten (oder besser: es sind), kommt einer ihrer Eigenschaften eine besondere Funktion zu: der Eigenschaft DialogResult. Das Formular bestimmt durch Setzen dieser Eigenschaft, wie das »Ergebnis« der Formularbedienung aussah. Doch das Setzen dieser Eigenschaft im Formular bewirkt noch mehr: Sobald Sie diese Eigenschaft im Rahmen eines Ereignisses auf einen anderen Wert als None setzen, wird der Dialog beendet!

Das bedeutet: Verfügt Ihr Dialog, den Sie modal aufrufen möchten, beispielsweise über eine OK-Schaltfläche, dann genügt eine einzige Zeile Code, nicht nur um das Dialogergebnis zu setzen, sondern auch, um den Dialog zu schließen – etwa auf die folgende Weise:

```
'Wird aufgerufen beim Auslösen von OK-Schaltfläche.  
Private Sub btnOK_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnOK.Click  
  
    'Das Zuordnen eines Wertes für DialogResult beendet den Dialog.  
    Me.DialogResult = Windows.Forms.DialogResult.OK  
  
End Sub
```

---

**HINWEIS:** Wichtig dabei ist es zu wissen, dass beim bloßen Setzen von DialogResult die Ressourcen des Dialoges nicht implizit wieder freigegeben werden. Wenn das Formular kritische Ressourcen blockiert, sollte es also in der Anwendung eine Instanz geben, die für das explizite Aufrufen von Dispose des Formulars sorgt. Eine gute Technik ist es, das Formular in einen Using...End Using-Block einzuschließen. Mehr zum Thema Dispose erfahren Sie in ► Kapitel 12. Der Umgang mit Using wird in ► Kapitel 6 beschrieben (Abschnitt: »Gezieltes Freigeben von Objekten mit Using«).

---

## Ein Muster für modale Formulare implementieren, die sich selber verwalten können

Nachdem diese wichtigen Fakten bekannt sind, schreiten wir zur eigentlichen Aufgabe. Ziel ist es, ein Muster für einen Dialog zu schaffen, der, wie eingangs erwähnt, ...

- ... eine Funktion zur Verfügung stellt, die Parameter zur Bearbeitung entgegennimmt, den eigentlichen Dialog darstellt und die vom Anwender eingegebenen und geprüften Daten zurückliefert,
- ... OK und Abbrechen implementiert, mit denen der Dialog geschlossen wird,
- ... eine zentrale Routine anbietet, in der die Daten überprüft werden und
- ... dafür sorgt, dass die Ressourcen des Dialogs wieder freigegeben werden.

Beginnen wir mit dem ersten Teil, einer Funktion, die wir im Dialog als Member-Methode implementieren und aufrufen können, sobald uns eine Instanz des Dialogs vorliegt. Die Funktion innerhalb des Formularcodes sieht in etwa wie folgt aus:

```
Public Class frmFürDatenabfrage  
  
    'Datenmember, auf die FormClosing zugreifen und die eine Funktion zurückliefern kann  
    Private myDatentyp As Datentyp  
  
    'Member-Funktion des Formulars, die das Bearbeiten von Daten regelt.  
    Public Function DatenBearbeiten(ByVal adr As Datentyp) As Datentyp  
        'Damit wird dafür gesorgt, dass die Ressourcen für das Formular  
        'nach Aufruf sofort wieder freigegeben werden!
```

```

Using Me
    'TODO: Daten in die Maske kopieren.

    'Dialog modal darstellen. Das Programm "hält" an dieser
    'Stelle quasi an, und nur noch Ereignisse innerhalb
    'dieses Formulars werden verarbeitet.
    Me.ShowDialog()

    'Hier geht's erst weiter, wenn OK oder Abbrechen geklickt wurde.
    Return myDatentyp
End Using
End Function
.
.
.
End Class

```

Diese Funktion deckt den ersten und den letzten Punkt unserer Todo-Liste bereits ab. Mit der Anweisung

```
Me.ShowDialog()
```

sorgt sie dafür, dass der Dialog als modaler Dialog dargestellt wird. Wenn die Funktion aufgerufen wird, nachdem eine Instanz des Formulars, das sie beherbergt, eingerichtet wurde, hält die Programmausführung an exakt dieser Stelle an und übergibt die Kontrolle der Nachrichtenwarteschlange des Formulars. Alle Ereignisse, die durch den Anwender bei der Bedienung des Dialogs auftreten können, werden von diesem Zeitpunkt an vom Formular verarbeitet, die entsprechenden Ereignisbehandlungs routinen auch aufgerufen. Erst wenn der Dialog auf irgendeine Art und Weise wieder geschlossen wird, geht es an dieser Stelle hinter dem ShowDialog-Aufruf weiter, und das Formular kann schließlich die Daten, die die aufrufende Instanz angefordert hat (oder Nothing, wenn der Dialog abgebrochen wurde) zurückliefern.

Damit die Ressourcen des Formulars freigegeben werden, wenn die Funktion beendet ist, klammern wir den gesamten Funktionscode in Using mit Bezug auf das eigene Formular (Me) ein. Dieses Konstrukt sorgt dafür, dass jedes Objekt, das das Formular selbst als Member-Variable speichert, einerseits noch zurückgegeben werden kann, die Formularinstanz anschließend aber (und *erst* dann) noch »entsorgt« wird.

Implementieren müsste diese Formulkarklasse nun noch lediglich eine Methode, die die übergebenen Daten, die bearbeitet werden sollen, in die entsprechenden Steuerelemente des Formulars schreibt.

Die aufrufende Instanz würde diese Funktion nun wie folgt verwenden:

```

Dim locFrm As New frmFürDatenabfrage
Dim locDatentypZumBearbeiten As Datentyp = IrgendwasWasBearbeitetWerdenSoll
Dim locDatentypZurück As Datentyp = locFrm.DatenBearbeiten(locDatentypZumBearbeiten)

```

Auffällig ist die Objektvariable myDatentyp, die zu Beginn des Formulars als Klassen-Member deklariert wurde; sie global für das ganze Formular zu deklarieren, scheint auf den ersten Blick unnötig zu sein. Klarer wird ihr Gültigkeitsbereich, wenn wir den fehlenden Formularcode, der die Datenüberprüfung und das Schließen (bzw. Abbrechen) des Formulars regelt, im Kontext betrachten:

```

'Wird aufgerufen beim Auslösen von OK-Schaltfläche.
Private Sub btnOK_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnOK.Click

    'Das Zuordnen eines Wertes für DialogResult beendet den Dialog.
    Me.DialogResult = Windows.Forms.DialogResult.OK
End Sub

'Wird aufgerufen beim Auslösen von Abbrechen-Schaltfläche.
Private Sub btnAbbrechen_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnAbbrechen.Click

    'Das Zuordnen eines Wertes für DialogResult beendet den Dialog.
    Me.DialogResult = Windows.Forms.DialogResult.Cancel
End Sub

'Wird aufgerufen, wenn das Formular geschlossen werden soll.
Protected Overrides Sub OnClosing(ByVal e As System.ComponentModel.CancelEventArgs)
    MyBase.OnClosing(e)
    'Überprüfung des Formulars nur, wenn OK geklickt wurde:
    If Me.DialogResult = Windows.Forms.DialogResult.OK Then

        myDatentyp = DatenDerMaskeÜberprüfenUndAlsObjektErmitteln

        'Wenn die ermittelten Daten Nothing und damit NICHT in Ordnung war, ...
        If myDatentyp Is Nothing Then
            '... bleibt der Dialog, der ja eigentlich geschlossen werden soll,
            'offen - und das erreichen wir durch Setzen von
            e.Cancel = True
        End If
    Else
        'Dieser Zeile kann nur erreicht werden, wenn Abbrechen
        'ausgelöst wurde. Dann wird auf jeden Fall Nothing als
        'Funktionsergebnis zurückgegeben.
        myDatentyp = Nothing
    End If
End Sub

```

Das Beenden des Dialogs wird eingeleitet, wenn eine der vorhandenen Schaltflächen *OK* oder *Abbrechen* des Formulars betätigt wird. Die jeweiligen Ereignisbehandlungsroutinen machen dazu nichts weiter, als die *DialogResult*-Eigenschaft des Formulars zu ändern, und damit das Schließen des Formulars anzustoßen.

Das Schließen durch *OK* darf allerdings nur dann erfolgen, wenn die Daten, die der Anwender in der Maske zuvor eingegeben hat, auch validiert werden konnten, anderenfalls muss nicht nur eine entsprechende Fehlermeldung ausgegeben werden oder eine Kennzeichnung der falsch eingegebenen Datenfelder erfolgen – das Formular darf vor allen Dingen nicht geschlossen werden.

Das *Closing*-Ereignis des Formulars verfügt über die Fähigkeit, das Schließen des Formulars zu verhindern (wenn es nicht gerade mit *Form.Dispose* sozusagen mit Brachialgewalt abgeschossen wurde, doch das sollte eh die Ausnahme bleiben). Also ergibt es Sinn, die Prüfung auf Richtigkeit der Anwendereingaben in exakt diese Ereignisbehandlungsroutine zu verlegen.

Übrigens: Denken Sie daran, dass wir uns in der OOP-Programmierung befinden, und dass Formularen im Grunde genommen auch nichts anderes als Klassen sind. Es wäre also von hinten durch die Brust ins Auge, wenn wir innerhalb eines Formulars ein Formularereignis einbänden. Stattdessen ist es guter Programmierstil, die Methode zu überschreiben, die das Ereignis auslöst – damit bekommt Ihr Formularcode das Ereignis natürlich schon ein paar Schritte vorher zu Gesicht. Im Codeauszug selbst passiert exakt das mit Protected Overrides Sub OnClosing.

Der Code dieser Methode ist nun auch von entscheidender Bedeutung: Nur wenn das Schließen des Formulars mit OK eingeleitet wurde (DialogResult war OK), muss eine Überprüfung (und am besten auch noch die gleichzeitige Erstellung der Instanz einer Datentypklasse aus den Formulardatenfeldern) erfolgen. Im Ereignisbehandlungscode wird das durch die Zeilen

```
If Me.DialogResult = Windows.Forms.DialogResult.OK Then  
    myDatentyp = DatenDerMaskeÜberprüfenUndAlsObjektErmitteln
```

erledigt. Und hier zeigt sich auch, wieso das Vorhalten einer Objektvariablen für die Datenrückgabe als Klassen-Member so wichtig ist: OnClosing muss in diese Objektvariable das Ergebnis mit den vom Anwender eingegebenen Daten ablegen, und damit der Ausgangscode der Formularaufruffunktion DatenBearbeiten diese Objektvariable als Funktionsergebnis zurückliefern kann, muss auch ihr der Zugriff auf die Objektvariable gestattet sein.

Hat die Datenüberprüfungsfunktion Fehler in der Anwendereingabe gefunden, dann sollte sie zu diesem Zeitpunkt schon entsprechende Hinweismeldungen im Formular gezeigt haben. Darüber hinaus gibt sie OnClosing in diesem Fall Nothing zurück, und OnClosing weiß anhand dessen, dass ein Fehler aufgetreten ist. Es verwendet nun folgendes Verfahren, um das Schließen des Formulars zu verhindern:

```
If myDatentyp Is Nothing Then  
    '... bleibt der Dialog, der ja eigentlich geschlossen werden soll,  
    'offen - und das erreichen wir durch Setzen von  
    e.Cancel = True  
End If
```

Durch Setzen der Cancel-Eigenschaft im Ereignisparameter von OnClosing lässt sich das Schließen des Formulars an dieser Stelle verhindern.

---

**TIPP:** Falls Sie übrigens mehr zur Kommunikation zwischen Komponenten in .NET-Anwendungen in Form von Ereignissen erfahren wollen, werfen Sie einen Blick in ► Kapitel 15.

---

### Implementierung des Musters in einer Anwendung

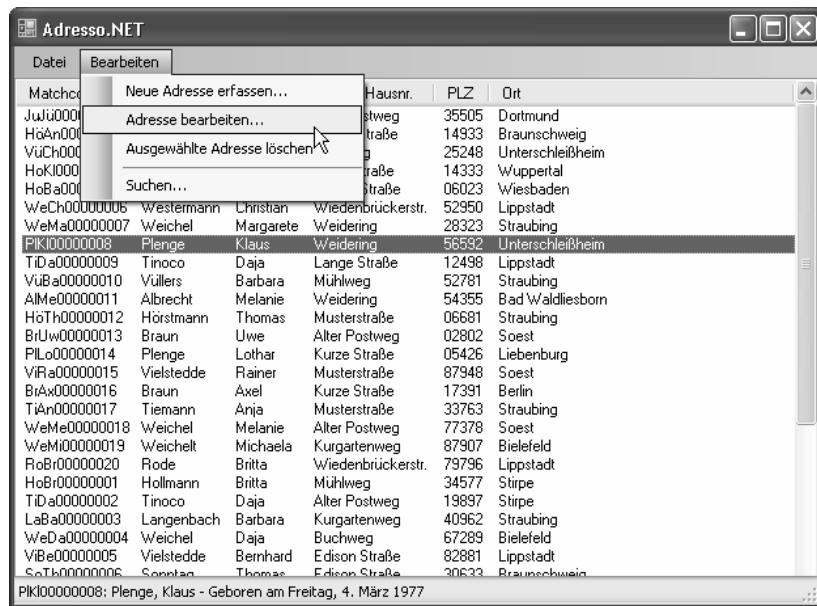
Theorie ist das eine, Praxis das andere. Wie sähe ein solcher Code nun aus, den wir in unsere schon bekannte Adresso-Anwendung implementieren wollten. Werfen wir dazu zunächst einen Blick auf die nächste Implementierungsstufe dieses Beispiels.

---

**BEGLEITDATEIEN:** Sie finden das Projekt für das folgende Beispiel im Verzeichnis .\VB 2005 - Entwicklerbuch\G - SmartClient\Kap27\Adresso02\.

---

Sie entdecken nach dem Programmstart in diesem Beispielprojekt zwei neue Funktionen im Menü **Bearbeiten**: *Neue Adresse eingeben* und *Adresse bearbeiten* (Siehe Abbildung 27.2). Die letzte Funktion lässt sich natürlich nur dann aufrufen, wenn Sie zuvor eine Adresse in der Liste selektiert haben. Ein und dasselbe Formular in der Anwendung deckt letzten Endes beide Funktionen ab.



**Abbildung 27.2:** In dieser Ausbaustufe von Adresso können Sie die vorhandene Liste der Adressen bearbeiten und um neue ergänzen

Dennoch unterscheidet sich die Funktionsweise des Formulars bei beiden Funktionen marginal: Beim Bearbeiten einer vorhandenen Adresse lässt sich der Matchcode nicht ändern. Beim Neuanlegen muss er mit eingegeben werden. Der modale Dialog zum Ändern bzw. Erfassen einer Adresse sieht wie folgt aus:



**Abbildung 27.3:** Mit diesem modal dargestellten Dialog werden Adressdaten in Adresso bearbeitet oder neu angelegt

Damit das in ► Abschnitt »Strukturieren von Daten in Windows Forms-Anwendungen« (ab Seite 743) postulierte Prinzip der mehrschichtigen Programmierung gewahrt bleibt, befinden sich die Aufrufe des Formulars nicht im Code des Hauptformulars sondern im Logikteil der Anwendung.

Der Aufruf des Formulars zum Bearbeiten gestaltet sich wie folgt:

```
''' <summary>
''' Ruft den Dialog zum Bearbeiten einer Adresse auf.
''' Liefert bei erfolgreicher Bearbeitung True zurück.
''' </summary>
''' <param name="adr">Adresse, die in dieser Instanz vorhanden sein muss
''' und bearbeitet werden soll.</param>
''' <returns></returns>
''' <remarks></remarks>
Public Function AdresseBearbeiten(ByVal adr As Adresse) As Boolean
    'Zu suchende Adresse merken, damit der Find-Predicate den
    'richtigen Index ausfindig machen kann.
    myAktuelleAdresse = adr

    'Index ermitteln
    Dim locIndex As Integer = Me.FindIndex(New Predicate(Of Adresse)(AddressOf IndexFinder))

    'Adresse ist nur bei Index > -1 in Auflistung vorhanden.
    'Die Nummer dieser Adresse merken wir uns in locIndex.
    If locIndex > -1 Then

        'Adresse bearbeiten.
        Dim locFrm As New frmAdresseNeuUndBearbeiten
        Dim locAdresse As Adresse = locFrm.AdresseBearbeiten(adr)

        'Die Adresse wurde geändert...
        If locAdresse IsNot Nothing Then
            '...und kann gegen die ursprüngliche ausgetauscht werden.
            Me(locIndex) = locAdresse

            'True zurückliefern, damit die aufrufende Instanz weiß, dass
            'es die Liste aktualisieren muss, um die Änderungen widerzuspiegeln.
            Return True
        End If
    End If
    'Keine Änderung
    Return False
End Function
```

Der relevante Codeteil ist hier fett hervorgehoben. Sie werden sehen, dass das Muster zum Aufruf des Dialogs prinzipiell dem entspricht, wie wir ihn im vorherigen Abschnitt (ab Seite 756) besprochen haben.

Übrigens: Um den Index des in der Liste markierten Objektes zu finden, bedient sich die Prozedur der FindIndex-Methode, die, angewendet auf die generische List(Of )-Auflistung, genauso wie deren Find-Methode zusammen mit einer Predicate-Instanz arbeitet. Genaueres zu diesem Thema finden Sie in ► Kapitel 20.

Ein wenig komplizierter ist das Neuanlegen einer Adresse, da sich hier ein besonderes Problem stellt: Wenn eine neue Adresse angelegt wird, darf der Matchcode, den der Anwender bestimmt, natürlich keinem schon vorhandenen Matchcode der Adressenliste entsprechen. Aus diesem Grund muss es eine Möglichkeit geben, dass der Dialog, in dem die Eingabe einer neuen Adresse erfolgt, die Adres-

sen auf einen bereits vorhandenen gleichen Matchcode kontrolliert. Nun entspricht es aber nicht der Aufteilung einer Anwendung in Schichten, wenn ein Erfassungsdialog Zugriff auf die komplette Datenschicht nehmen darf. Abermals helfen uns Delegaten (besprochen in ► Kapitel 15) aus dem Dilemma. Anstatt dem Dialog eine Referenz auf die kompletten Adressendaten zu geben, mit denen er anschließend eine Kontrolle auf einen vorhandenen Matchcode in der Liste durchführen könnte, übergeben wir ihm einfach die *Adresse einer Funktion*, die das für ihn erledigt. Zu gegebener Zeit ruft er über den ihm übergebenden Delegaten einfach eine Prozedur in der Anwendungslogikschicht auf, die diese Überprüfung für ihn vornimmt – das Prinzip der strikten Schichttrennung bleibt damit gewahrt.

Der Code in der Anwendungslogik, der das Neuanlegen einer Adresse einleitet, sieht damit wie folgt aus:

```

'<summary>
'<param name="Matchcode">Matchcode, der überprüft werden soll.</param>
'<returns>True, wenn der Matchcode bereits existierte.</returns>
'<remarks></remarks>
Public Delegate Function MatchcodeCheckenDelegate(ByVal Matchcode As String) As Boolean

'<summary>
'<param name="Matchcode">Matchcode, der überprüft werden soll.
'<returns>True, wenn die Adresse dieser Auflistung hinzugefügt wurde.</returns>
'<remarks></remarks>
Public Function NeueAdresse() As Boolean
    'Formularinstanz erstellen
    Dim locFrm As New frmAdresseNeuUndBearbeiten

    'Neue Adresse ermitteln. Dazu die Routine als Delegat übergeben,
    'die überprüft, ob der Matchcode in dieser Auflistung bereits vorhanden ist.
    Dim locAdresse As Adresse = locFrm.NeueAdresse(AddressOf MatchcodeChecken)

    'Nur wenn eine gültige Adresse ermittelt wurde...
    If locAdresse IsNot Nothing Then

        '...diese dieser Auflistung hinzufügen.
        Me.Add(locAdresse)

        'True zurückliefern, damit die aufrufende Instanz weiß, dass
        'es die Liste aktualisieren muss, um die Änderungen widerzuspiegeln.
        Return True
    End If
    'Keine Änderung
    Return False
End Function
```

```

'<summary>
'< Überprüft, ob ein vorhandener Matchcode bereits in der Auflistung vorhanden ist.
'</summary>
'<param name="Matchcode"></param>
'<returns></returns>
'<remarks></remarks>
Public Function MatchcodeChecken(ByVal Matchcode As String) As Boolean
    For Each locItem As Adresse In Me
        If locItem.Matchcode = Matchcode Then
            Return True
        End If
    Next
    Return False
End Function

```

Delegat und korrelierende Funktion entsprechen der ersten und letzten fett hervorgehobenen Listingzeile im oben zu sehenden Codeausschnitt. Der »mittlere Teil« entspricht fast wörtlich dem im vorherigen Abschnitt besprochenen Muster.

Mit dieser Kapselung der Funktionalität beschränkten sich die Änderungen, die an der Benutzeroberfläche in *frmMain.vb* nötig sind, auf folgende beiden Ereignisbehandlungsmethoden:

```

'Wird aufgerufen, wenn der Anwender im Menü Bearbeiten
'den Menüpunkt Neue Adresse erfassen anklickt.
Private Sub tsmNeueAdresseErfassen_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles tsmNeueAdresseErfassen.Click

    'Nur aktualisieren, wenn das Neuanlegen erfolgreich war.
    If myAdressen.NeueAdresse() Then
        ElementeDarstellen()
    End If

End Sub

'Wird aufgerufen, wenn der Anwender im Menü Bearbeiten
'den Menüpunkt Adresse bearbeiten anklickt.
Private Sub tsmAdresseBearbeiten_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles tsmAdresseBearbeiten.Click

    'Herausfinden, ob es überhaupt eine selektierte Adresse gibt
    If lvwAdressen.SelectedItems.Count > 0 Then

        'Erste selektierte Adresse wird bearbeitet
        Dim locAdresse As Adresse = DirectCast(lvwAdressen.SelectedItems(0).Tag, Adresse)

        'Nur aktualisieren, wenn das Bearbeiten erfolgreich war.
        If myAdressen.AdresseBearbeiten(locAdresse) Then
            ElementeDarstellen()
        End If
    End If

End Sub

```

Nun ist die eigentliche Realisierungsweise des Formularcodes zum Erfassen einer Adresse natürlich von besonderem Interesse. Abgesehen von der Überprüfung der Formulardaten, auf die der nächste Abschnitt eingehen will, entspricht dieser Code ebenfalls zu 99 % dem vorgestellten Muster:

```
Public Class frmAdresseNeuUndBearbeiten

    'Hier steht das Ergebnis der beiden Funktionen
    Private myAdresse As Adresse

    'Delegat, mit dessen Hilfe ein doppelter Matchcode überprüft wird.
    Private myMatchcodeCheckenProc As Adressen.MatchcodeCheckenDelegate

    ''' <summary>
    ''' Stellt diesen Dialog dar, und ermittelt das Objekt einer neuen Adresse.
    ''' </summary>
    ''' <param name="matchcodeChecken">Delegat, der auf eine Prozedur verweist, die doppelte
    ''' Matchcodes in der Auflistung überprüft.</param>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Public Function NeueAdresse(ByVal matchcodeChecken As Adressen.MatchcodeCheckenDelegate) As Adresse

        'Damit wird dafür gesorgt, dass die Ressourcen für das Formular
        'nach Aufruf sofort wieder freigegeben werden!
        Using Me
            'Dialogtitel anpassen
            Me.Text = "Adresse bearbeiten"

            'Delegaten zuordnen - den brauchen wir später, um auf doppelte Matchcodes zu prüfen.
            myMatchcodeCheckenProc = matchcodeChecken

            'Dialog modal darstellen. Das Programm "hält" an dieser
            'Stelle quasi an, und nur noch Ereignisse innerhalb
            'dieses Formulars werden verarbeitet.
            Me.ShowDialog()

            'Hier geht's erst weiter, wenn OK oder Abbrechen geklickt wurde.
            Return myAdresse
        End Using
    End Function

    ''' <summary>
    ''' Stellt diesen Dialog dar, lässt den Anwender eine Adresse bearbeiten und liefert
    ''' ein neues Adresse-Objekt zurück, das die Änderungen widerspiegelt.
    ''' </summary>
    ''' <param name="adr">Adresse-Objekt, das bearbeitet wird.
    ''' ACHTUNG! Das Rückgabeobjekt stellt eine neuen Instanz dar!</param>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Public Function AdresseBearbeiten(ByVal adr As Adresse) As Adresse

        'Damit wird dafür gesorgt, dass die Ressourcen für das Formular
        'nach Aufruf sofort wieder freigegeben werden!
        Using Me
```

```

'Dialogtitel anpassen
Me.Text = "Adresse bearbeiten"

'Daten in die Maske kopieren.
DatenInMaske(adr)

'Matchcode darf nicht editiert werden:
txtMatchcode.ReadOnly = True

'Dialog modal darstellen. Das Programm "hält" an dieser
'Stelle quasi an, und nur noch Ereignisse innerhalb
'dieses Formulars werden verarbeitet.
Me.ShowDialog()

'Hier geht's erst weiter, wenn OK oder Abbrechen geklickt wurde.
Return myAdresse
End Using

End Function

'Wird aufgerufen beim Auslösen von OK-Schaltfläche.
Private Sub btnOK_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnOK.Click

'Das Zuordnen eines Wertes für DialogResult beendet den Dialog.
Me.DialogResult = Windows.Forms.DialogResult.OK

End Sub

'Wird aufgerufen beim Auslösen von Abbrechen-Schaltfläche.
Private Sub btnAbbrechen_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
Handles btnAbbrechen.Click

'Das Zuordnen eines Wertes für DialogResult beendet den Dialog.
Me.DialogResult = Windows.Forms.DialogResult.Cancel

End Sub

'Wird aufgerufen, wenn das Formular geschlossen werden soll.
Protected Overrides Sub OnClosing(ByVal e As System.ComponentModel.CancelEventArgs)
 MyBase.OnClosing(e)
 'Überprüfung des Formulars nur, wenn OK geklickt wurde:
 If Me.DialogResult = Windows.Forms.DialogResult.OK Then

'Daten aus Maske gibt ein Adresse-Objekt nur dann zurück,
'wenn die Daten in Ordnung waren.
myAdresse = DatenAusMaske()

'Wenn myAdresse also Nothing und damit NICHT in Ordnung war, ...
If myAdresse Is Nothing Then

'... bleibt der Dialog, der ja eigentlich geschlossen werden soll,
'offen - und das erreichen wir durch Setzen von
e.Cancel = True

```

```

    End If
Else
    'Dieser Zeile kann nur erreicht werden, wenn Abbrechen
    'ausgelöst wurde. Dann wird auf jeden Fall Nothing als
    'Funktionsergebnis zurückgegeben.
    myAdresse = Nothing
End If
End Sub

'Überprüft die Eingaben im Formular, und liefert
'im Erfolgsfall ein fix-und-fertiges Adresse-Objekt
'aus den Eingabefeldern zurück.
Private Function DatenAusMaske() As Adresse
    'Wird im nächsten Abschnitt beschrieben – daher hier ausgelassen.
End Function

'Kopiert ein vorhandenes Adresse-Objekt in die Maske
'für das weitere Bearbeiten.
Private Sub DatenInMaske(ByVal adr As Adresse)
    txtMatchcode.Text = adr.Matchcode
    txtVorname.Text = adr.Vorname
    txtNachname.Text = adr.Name
    txtStraße.Text = adr.Straße
    mtbPlz.Text = adr.PLZ
    txtOrt.Text = adr.Ort
    mtbGeburtsdatum.Text = adr.Geburtsdatum.ToShortDateString
    lblWochentag.Text = adr.Geburtsdatum.ToString("dddd")
End Sub

Private Sub mtbGeburtsdatum_LostFocus(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles mtbGeburtsdatum.LostFocus
    Dim locGebDatum As Date
    If Date.TryParse(mtbGeburtsdatum.Text, locGebDatum) Then
        lblWochentag.Text = locGebDatum.ToString("dddd")
    End If
End Sub
End Class

```

## Überprüfung auf richtige Benutzereingaben in Formularen

Die Qualität Ihres Arbeitsalltags fällt und wächst mit dem Aufwand, den Sie zu Zeiten der Softwareentwicklung in das Thema Eingabeverprüfungen stecken. Wenn Sie professionell Softwareentwicklung betreiben, dann können Sie mir glauben, dass Sie, wenn Sie sich morgens an Ihren Entwicklungskomputer setzen, Sie kein Fax mit Inhalt dessen auf dem Schreibtisch liegen haben möchten, was Sie auch in Abbildung 27.4 sehen können.

Gerade bei der Datenerfassung, also wenn es um die Schnittstelle zwischen Ihrer Software und dem Anwender geht, können bei der Entwicklung viele Fehler passieren. Und Sie sollten nach Möglichkeit extrem kreativ werden, um Vorahnungen zu entwickeln, welche Kombinationen unglücklicher Umstände dazu führen können, dass Ihre Software jahrelang funktioniert, und plötzlich nicht mehr.

## Anekdoten aus der Praxis

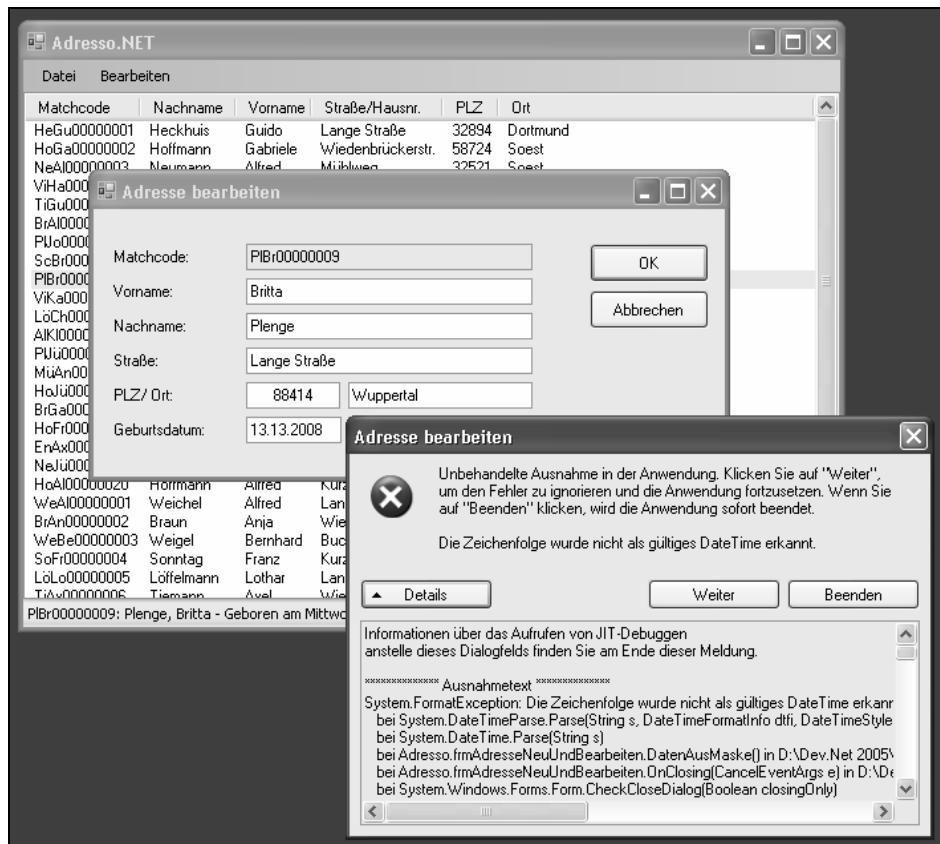
So wurde ich als Berater zur Analyse eines Phänomens einer Software bestellt – einer Erfassung von Mitarbeiterzeiten in Produktionsbetrieben – die jahrelang anstandslos ihren Dienst verrichtet hatte. Die Frage an den Kunden, ob in den vergangenen Tagen etwas Ungewöhnliches passiert sei, beantwortete er mit einem klaren Nein.

Nach einigem Recherchieren stellte sich allerdings heraus, dass doch etwas Außergewöhnliches vorgefallen war – wenn auch nicht im Betrieb vor Ort: Der Schwesterbetrieb dieses Berliner Unternehmens, das seinen Sitz in Magdeburg hatte, war nämlich tragischerweise direkt vom Elbe-Hochwasser 2002 betroffen.

Der Konzern bewies aber gute Führungsqualitäten in Form von schneller Reaktion und gutem Improvisationstalent, und so verlegte man kurzerhand die Produktion der wichtigsten Produkte von Magdeburg nach Berlin; die Berliner Mitarbeiter und der Betriebsrat erklärten sich sofort bereit, eine noch nie da gewesene 3. Schicht zu fahren, um den betroffenen Magdeburgern den Vortritt zu lassen.

Damit trat eine Konstellation ein, die der Software das erste Mal abverlangte, datumsübergreifende Buchungen durchführen zu müssen, und da das Tagesdatum nicht bei der Erfassung von Start- und Endzeit sondern nur als Buchungsdatum berücksichtigt wurde, sah es aus Sicht der Software so aus, als ob die Anfangszeit (20:00, 24.8.2002) zeitlich *hinter* der Endzeit (2:30, aber 25.8.2002) lag.

Die Software quittierte das mit einem Absturz bei den anschließenden Auswertungen; alle zuvor ermittelten Zeiten konnten aber glücklicherweise mit angepasstem Filter abermals aus den Zeiterfassungsterminals übernommen werden.



**Abbildung 27.4:** Wenn ein Anwender Ihrer Software einen solchen »Programmzustand« provozieren kann, liegt der Fehler leider in Ihrer Software ...

Es spielt keine Rolle, woher Ihre Software Ihre Daten bezieht – ob nun von Maschinen oder von Menschen. Sie sollten nicht zuletzt im eigenen Interesse (ruhiger schlafen, bessere Reputation) sehr großen Wert darauf legen, das größtmögliche Augenmerk auf das Sichern von Datenschnittstellen gegen Fahrlässigkeit und groben Unfug zu legen.

Was die manuelle Eingabe von Daten anbelangt, bietet das .NET-Framework eine Komponente an, die dem Anwender Falscheingaben auf eine – meiner Meinung nach – recht eigenwillige aber dennoch nicht minder plausible Weise vor Augen führt.

Abbildung 27.5 zeigt dieses Verfahren im Einsatz: Durch die `ErrorProvider`-Komponente lassen sich beim Auftreten von Fehlern in Eingabefeldern von Formularen diese betroffenen Felder mit einem roten Ausrufungszeichen markieren. Tritt der Fehler auf, blinken diese Ausrufungszeichen sogar für eine bestimmte (einstellbare) Weile.



**Abbildung 27.5:** Durch die *ErrorProvider*-Komponente können Sie Anwender auf Fehler in Eingabefeldern gezielt aufmerksam machen

Fährt der Anwender anschließend mit dem Mauszeiger auf eines der Ausrufungszeichen, zeigt der ErrorProvider eine zuvor definierbare Fehlermeldung als Tooltip an.



**Abbildung 27.6:** Eine Komponente (ein zur Entwurfzeit nicht sichtbares Steuerelement) ziehen Sie bei Bedarf ebenfalls ins Formular; der Designer legt es dann im Komponentenfach ab

Um ein Formular mit einer ErrorProvider-Komponente auszustatten, brauchen Sie sie lediglich aus der Toolbox ins Formular zu ziehen. Der Visual Basic-Designer legt dann – so dies die erste Komponente war – ein Komponentenfach unterhalb des Formulars an und legt die Komponente dort ab.

Der ErrorProvider verfügt nämlich über keine eigene Benutzeroberfläche. Er erweitert einfach die Steuerelemente, die sich auf dem Container (dem Formular in diesem Fall) befinden, um die entsprechende Funktionalität.

Zum Anzeigen eines Eingabefehlers neben einem Steuerelement des Formulars genügt anschließend ein einziger Methodenaufruf, etwa in dieser Form:

```
ErrorProvider.SetError(steuerelementName, "Fehlermeldungstext für den Tooltip!")
```

Wenn Sie möchten, dass alle Fehlermarkierungen wieder aufgehoben werden, verwenden Sie die folgende Methode:

```
ErrorProvider.Clear()
```

---

**BEGLEITDATEIEN:** Sie finden das Projekt zur Veranschaulichung dieses Abschnittes ebenfalls im Verzeichnis .\VB 2005 - Entwicklerbuch\G - SmartClient\Kap27\Adresse02\.

---

Das Adresseo-Beispiel verwendet diese Vorgehensweise, und der entsprechende Code, der von OnFormClosing (siehe Beispiele der vorherigen Abschnitte) aus aufgerufen wird, sieht folgendermaßen aus:

```
'Überprüft die Eingaben im Formular, und liefert
'im Erfolgsfall ein fix-und-fertiges Adresse-Objekt
'aus den Eingabefeldern zurück.
Private Function DatenAusMaske() As Adresse

    Dim locFehler As Boolean

    'Alle möglichen vorherigen Fehler zurücksetzen
    ErrProv.Clear()

    'Wenn keine Eingabe im Feld gemacht wurde,
    If String.IsNullOrEmpty(txtMatchcode.Text) Then

        'Fehlermeldung setzen.
        ErrProv.SetError(txtMatchcode, "Fehlende Eingabe!")
        locFehler = locFehler Or True

    Else

        'Der Delegat ist nur beim Neuanlegen einer Adresse vorhanden,
        'deswegen auf Vorhandensein überprüfen!
        If myMatchcodeCheckenProc IsNot Nothing Then
            'Feststellen, ob der Matchcode in der Auflistung schon existiert!
            'Das passiert in unserem Fall über einen Delegaten.
            If myMatchcodeCheckenProc.Invoke(txtMatchcode.Text) Then
                ErrProv.SetError(txtMatchcode, "Dieser Matchcode ist schon vorhanden!")
                locFehler = locFehler Or True
            End If

            End If
        End If
        'Ähnlich bei allen anderen vorgehen.
        If String.IsNullOrEmpty(txtVorname.Text) Then
            ErrProv.SetError(txtVorname, "Fehlende Eingabe!")
            locFehler = locFehler Or True
        End If

        If String.IsNullOrEmpty(txtNachname.Text) Then
            ErrProv.SetError(txtNachname, "Fehlende Eingabe!")
            locFehler = locFehler Or True
        End If
```

```

If String.IsNullOrEmpty(txtStraße.Text) Then
    ErrProv.SetError(txtStraße, "Fehlende Eingabe!")
    locFehler = locFehler Or True
End If

Dim locGebDate As Date
If Not Date.TryParse(mtbGeburtsdatum.Text, locGebDate) Then
    ErrProv.SetError(mtbGeburtsdatum, "Falsches Datumsformat!")
    locFehler = locFehler Or True
End If

If String.IsNullOrEmpty(txtOrt.Text) Then
    ErrProv.SetError(txtOrt, "Fehlende Eingabe!")
    locFehler = locFehler Or True
End If

'Fehler war vorhanden - Nothing zurückliefern
If locFehler Then Return Nothing

'Alles war OK, es gibt eine neue Adresse.
Return New Adresse(txtMatchcode.Text, txtNachname.Text, _
                   txtVorname.Text, txtStraße.Text, _
                   mtbPlz.Text, txtOrt.Text, locGebDate)
End Function

```

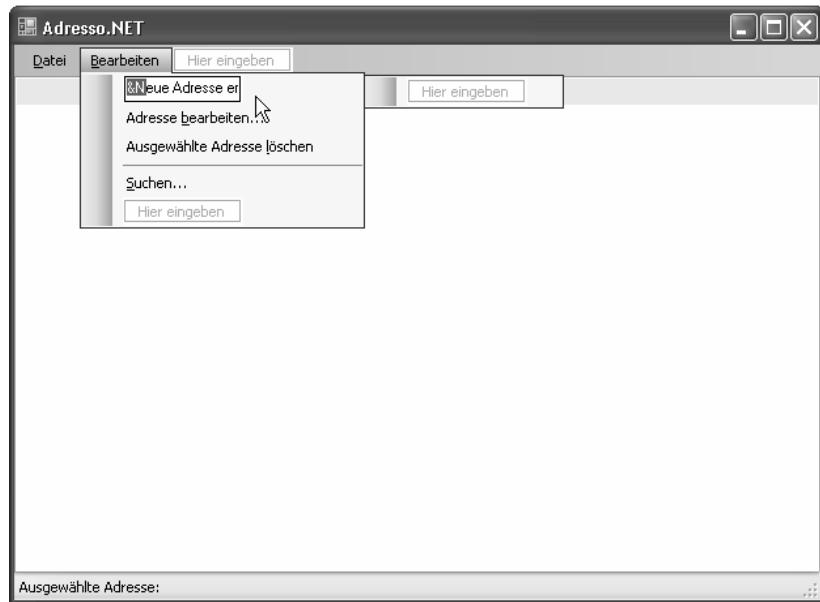
## Anwendungen über die Tastatur bedienbar machen

Dieses Buch wäre schätzungsweise einen Monat später fertig geworden, könnte man die Funktionen von Microsoft Word ausschließlich über die Maus bedienen. Wenn ich selbst den typografischen Satz eines Buches vornehme, kann ich die Batterien aus meiner Funkmaus herausnehmen – ich brauche sie oft über mehrere Stunden gar nicht. Zwar ist Windows eine grafische Benutzeroberfläche, doch ist es vielen Softwareentwicklern gar nicht bewusst, wie wichtig es für einen Anwender ist, der täglich mit einer Software arbeiten muss, diese nur ausschließlich über die Maus bedienen zu können. Es kostet wahnsinnig viel Zeit, von der Tastatur, die man natürlich für Texteingaben benötigt, zur Maus umzugreifen, weil man nur so an eine bestimmte Programmefunktion herankommen kann.

Dabei bedarf es nicht viel, um Dialoge oder Menüs auch über die Tastatur bedienbar zu machen. Es gibt unter Windows definierbare Schnellzugriffstasten, deren Implementierung in die eigene Software im Handumdrehen erledigt ist.

### Definition von Schnellzugriffstasten per »&«-Zeichen

Beim Erstellen von Menüs genügt ein Voranstellen des »&«-Zeichens, um dieses Menü mit dem dahinter stehenden Buchstaben über die Tastatur (in Verbindung mit **Alt**) aktivieren zu können (siehe Abbildung 27.7).

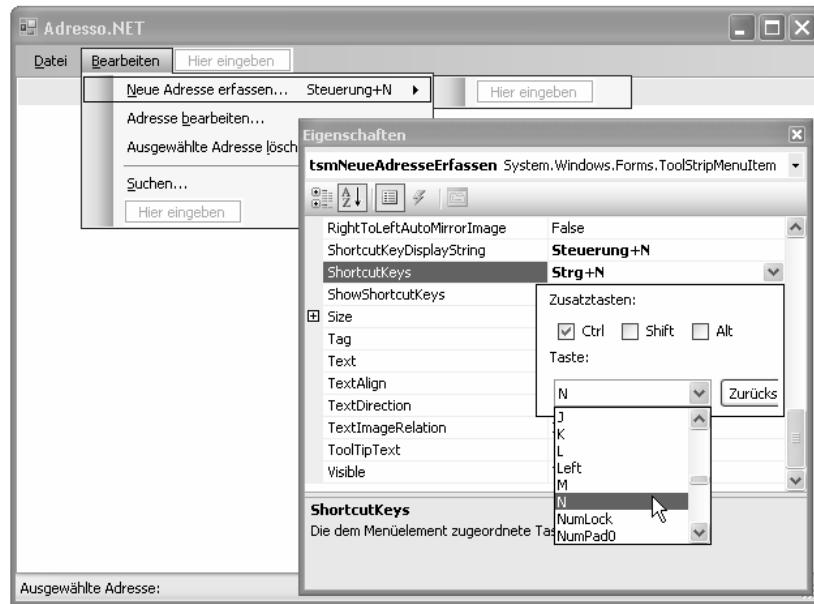


**Abbildung 27.7:** Mit dem Kaufmannsund (»&«) definieren Sie Schnellzugriffstasten in Menüs ...

Für jeden Menüeintrag können Sie im Bedarfsfall zudem eine Funktionstaste definieren, mit der Sie den Menübefehl (und damit die dahinter stehende Funktion) direkt auslösen können. Dazu klicken Sie einfach auf den entsprechenden Menüeintrag und stellen im Eigenschaftenfenster seine ShortcutKeyes-Eigenschaft ein. Standardmäßig blendet dann der Menüeintrag neben dem eigentlichen Menütext einen Hinweis auf diese Tastenkombination für den Anwender ein. Sollte Ihnen diese Tastenkombinationsbeschreibung nicht passen, können Sie sie mit dem ShortcutKeyDisplayString nach Gutdünken anpassen.<sup>2</sup> Und falls Sie es überhaupt nicht wünschen, dass die Taste oder Tastenkombination neben dem Menütext angezeigt wird, setzen Sie die ShowShortcutKeyes-Eigenschaft des entsprechenden Eintrags auf False.

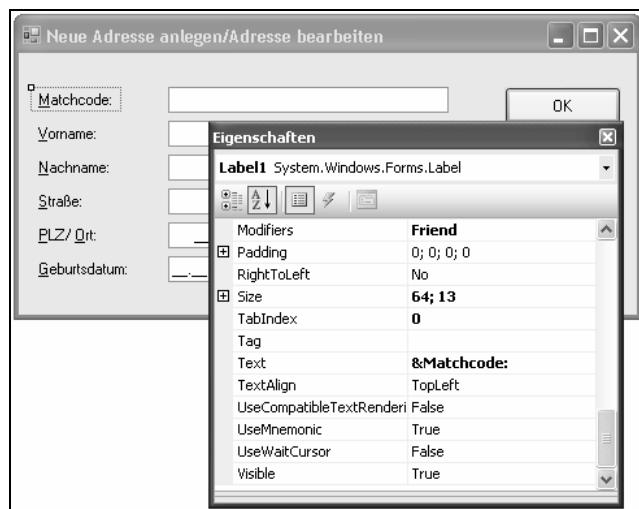
---

<sup>2</sup> Und in diesem Zusammenhang sei mir die bissige Bemerkung erlaubt: »Strg« steht auf deutschen Tastaturen nicht für »String«, nicht für »Strong«, nicht für »Struck« oder sonst einen Blödsinn. Es ist die vielleicht nicht ganz gelungene Abkürzung von »S t e r u n g«. Und falls Sie wissen, welche Geschichte sich hinter der Abkürzung »S-Abfr« auf älteren Tastaturen verbirgt (unterhalb der Druck-Taste) – für eine kurze E-Mail wäre ich dankbar ... ;-)



**Abbildung 27.8:** Neben den Schnellzugriffstasten lassen sich für jeden Menübefehl auch Funktionstasten (»Abkürzungstasten«) einrichten – einfach und schnell über das Eigenschaftenfenster ...

Bei Textfeldern in Formularen beispielsweise kommen zwei Faktoren ins Spiel, um das Eingabefeld per Tastatur zur fokussieren: Ein beschreibendes Label-Steuerelement, das sich vor dem eigentlich zu aktivierendem Steuerelement befindet, und eine richtig eingestellte Aktivierungsreihenfolge.



**Abbildung 27.9:** ... und mithilfe davor stehender Beschriftungen (Label) auch für Steuerelemente, die über keine selbst beschreibende Beschriftung verfügen – wie beispielsweise Textbox-Steuerelemente

Da das Label vor dem Eingabesteuerelement (z.B. TextBox) nur eine beschreibende Funktion hat, selbst aber nicht aktiviert werden kann, wird beim Auslösen der Zugriffstaste das in der Aktivierungsreihenfolge hinter dem Label-Steuerelement stehende Steuerelement fokussiert.

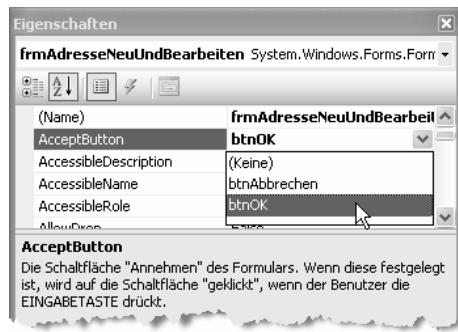
Die Aktivierungsreihenfolgen können Sie übrigens einstellen, indem Sie das Formular durch Anklicken im Designer selektieren, und anschließend aus dem Menü *Ansicht* den Menüpunkt *Aktivierungsreihenfolge* auswählen. Der Visual Studio-Designer schaltet anschließend in einen besonderen Bearbeitungsmodus, der das Bestimmen der Aktivierungsreihenfolge (englisch: *Tab Ordner* für *Tabulatorreihenfolge*) durch simples Nacheinander-Anklicken der Steuerelemente auf dem Formular erlaubt. Eine geschickte Auswahl der Aktivierungsreihenfolge ermöglicht, wie in Abbildung 27.10 zu sehen, dabei auch Konstellationen, in denen eine Beschriftung für zwei Eingabefelder gilt.



**Abbildung 27.10:** Mit der Funktion *Aktivierungsreihenfolge* bestimmen Sie die Zugriffsreihenfolge für das Fokussieren von Steuerelementen per **Tabulator**-Taste durch simples Nacheinander-Anklicken

## Accept- und Cancel-Schaltflächen in Formularen

Viele modale Dialoge lassen sich mit den Tasten **Eingabe** und **Esc** beenden.



**Abbildung 27.11:** Die Einstellung der Schaltflächen für **Eingabe** und **Esc** nehmen Sie über *AcceptButton* und *CancelButton* am Formular vor

Für .NET-Formulare richten Sie diese Tasten für Schaltflächen (und ausschließlich für Schaltflächen) über die Formulareigenschaften *AcceptButton* (für Schaltflächen, die mit **Eingabe** auszulösen sind) und *CancelButton* (für Schaltflächen, die mit **Esc** auszulösen sind) ein.

---

**HINWEIS:** Dies ist eine für VB6-Entwickler unübliche Vorgehensweise, da sich diese Art der Formularsteuerung in VB6 über Eigenschaften der Schaltflächen und nicht über die des Formulars selbst einstellen ließen. Achten Sie also darauf, die beschriebenen Eigenschaften am Formular und nicht an den Schaltflächen im Eigenschaftenfenster zu suchen!

---

Im Normalfall entspricht `AcceptButton` der *OK*-Schaltfläche, `CancelButton` der *Abbrechen*-Schaltfläche eines Formulars. Sinn ergibt das Setzen dieser Eigenschaften übrigens nur in modal darzustellenden Formularen.

## Über das »richtige« Schließen von Formularen

Es mag auf den ersten Blick nicht eines ganzen Abschnittes wert sein; besorgte Nachfragen im Usenet machen mich aber glauben, dass ein paar Sätze über das richtige Schließen eines Formulars nicht schaden können.

Um ein Formular programmtechnisch zu schließen, gibt es vier Möglichkeiten:

- `formInstance.Hide`
- `formInstance.Close`
- `formInstance.Dispose`
- `formInstance.DialogResult = [einWert]>>DialogResult.None]` (nur in modalen Dialogen – lesen Sie dazu bitte den ► Abschnitt »Der Umgang mit modalen Formularen« ab Seite 755).

### Unsichtbarmachen eines Formulars mit Hide

Die `Hide`-Methode macht nichts weiter, als die `Visible`-Eigenschaft eines Formulars auf `False` zu setzen. Damit gibt es die eigentliche Instanz eines Formulars zwar noch, das Formular befindet sich nur nicht mehr sichtbar auf dem Bildschirm.

Allerdings gibt es etwas zu beachten, wenn Sie Formulare modal darstellen:

Das nämlich bedeutet für das Anwenden der `Hide`-Methode (oder auch für das Setzen der `Visible`-Eigenschaft des Formulars auf `False`): Wenn die Darstellung des Formulars zuvor modal erfolgte, wird in diesem Moment die Warteschlange beendet und die Kontrolle an die aufrufende Instanz zurückgegeben, die Ressourcen des Formulars werden aber nicht sofort freigegeben.

### Schließen des Formulars mit Close

Das Schließen des Formulars mit `Close`, »emuliert« sozusagen das Schließen des Formulars durch den Anwender. Dabei hat die das Formular einbindende Instanz (oder das Formular selbst) die Möglichkeit, den Vorgang des Schließens zu verhindern.

Entweder durch Überschreiben von `OnClosing` (soweit es das Formular selbst betrifft) oder durch Einbinden des `Closing`-Ereignisses (soweit es die das Formular einbindende Instanz betrifft) besteht die Möglichkeit, durch die `CancelEventArgs` des Ereignisses den kompletten Schließenvorgang abzubrechen:

```

Private Sub frmMain_Closing(ByVal sender As Object, ByVal e As System.ComponentModel.CancelEventArgs) _
    Handles MyBase.Closing
    'Formular soll nicht geschlossen werden
    e.Cancel = True
End Sub

```

Allerdings: Wenn nicht programmtechnisch interveniert wurde, das Formular zu schließen, ist es nicht nur unsichtbar geworden, sondern wird durch den Garbage Collector bei der nächsten Gelegenheit entsorgt. Die Close-Methode entspricht also quasi einem intervenierbaren Dispose (siehe nächster Abschnitt).

### **Was passiert bei Form.Close intern:**

Für die Puristen unter Ihnen hier eine Ereignisabfolge, die beschreibt, in welcher Reihenfolge Dinge beim Schließen eines Formulars mit Close passieren (lesen Sie im Bedarfsfalls zunächst die entsprechenden Abschnitte im nachfolgenden Kapitel, um mehr Grundsätzliches über die interne Verwaltung und Verarbeitung von Nachrichten bei Windows-Anwendungen zu erfahren).

- Close wird aufgerufen, das Formular sendet windows-intern die Nachricht WM\_Close mit seinem eigenen Window Handle an die Nachrichtenwarteschlange.
- Wenn die Nachrichtenschlange verarbeitet wird, schickt diese die Nachricht weiter an WndProc des Formulars.
- WndProc ruft die Formular-interne Methode WMClose auf.
- WMClose ruft die Dispose-Methode des Formulars auf, nachdem OnClosing und OnClose aufgerufen wurden, die ihrerseits die entsprechenden Ereignisse ausgelöst haben. Dispose wird aber nur dann aufgerufen, wenn OnClosing den Schließvorgang nicht abgebrochen hat.

---

**TIPP:** Wenn sich Ihr Formular selber schließen soll, es dies aber in einer Methode macht, die einen Rückgabewert an die aufrufende Instanz zurückliefern muss, der aus einem Member der Formulkarklasse hervorgeht, schließen Sie es mit Close, und wenden Sie dann Return an. Nur so ist gewährleistet, dass es den zurückzugebenden Member noch gibt, wenn Return ausgeführt wird. Dispose würde das Formular sofort zur Entsorgung freigeben; Sie könnten nicht sicher sein, dass es den zurückzugebenden Wert beim Return noch gibt.

Da diese Anforderungen in der Regel nur für modale Dialoge bestehen, verwenden Sie noch besser das Programmiermuster, über das der ► Abschnitt »Der Umgang mit modalen Formularen« ab Seite 755 genau Auskunft gibt.

---

## **Entsorgen des Formulars mit Dispose**

Dispose macht mit einem Formular, was es auch mit jedem anderen Objekt macht: es zur Entsorgung durch den Garbage Collector freigeben. Das bedeutet: Dispose lässt ein Formular unaufhaltsam und für immer verschwinden. Das Closing-Ereignis wird *nicht* aufgerufen, und weder die das Formular einbindende Instanz noch das Formular selbst haben die Möglichkeit, etwas dagegen zu tun. Das gilt natürlich auch für alle Member des Formulars. Lesen Sie deswegen bitte auch die Ausführungen des vorherigen Abschnittes.

# Grundsätzliches zum Darstellen von Daten aus Auflistungsklassen in Steuerelementen

Fast alle denkbaren Anwendungen verlangen es, dass Datenaufzählungen in irgendeiner Form visualisiert werden, und das geschieht in der Regel in Form von Listen. Das **ListBox**-Steuerelement eignet sich dazu nur bedingt, da es im Prinzip nur eine Eigenschaft eines Datenelements darstellen kann. Wenn es darum geht, umfangreiche Mengen von Daten in SmartClient-Anwendungen übersichtlich darzustellen, bieten sich dazu zwei Steuerelemente an:

- Das **ListView**-Steuerelement.
- Das **DataGridView**-Steuerelement.

Das **ListView**-Steuerelement eignet sich hervorragend zur reinen Textdarstellung von tabellarischen Daten, wenn Sie seine detaillierte Listendarstellung verwenden. Sie kennen es aus dem Windows-Explorer und der Dateianzeige: Es ist ein unglaublich mächtiges Steuerelement, das die verschiedensten Darstellungsformen kennt. So können Sie sich auch im Windows-Explorer für eine Miniaturansicht von Daten, für die gekachelte Darstellung, die symbolische Darstellung, die einfache Listendarstellung in die detaillierte Listendarstellung entscheiden. In den anschließenden Beispielen werden wir uns auf diesen letzten Darstellungsstil beschränken.

Das **DataGridView**-Steuerelement ist ein komplett neues Steuerelement im .NET-Framework 2.0 und dann interessant, wenn nicht nur reine Textdaten, sondern komplexere Datenvisualisierungen in verschiedenen Spalten umgesetzt werden müssen. Im Gegensatz zum **ListView**-Steuerelement erlaubt es auch das Bearbeiten von Daten direkt in Tabellenform. Dafür ist der Programmieraufwand zur Verwaltung der Daten im **DataGridView**-Steuerelement auch größer, um Daten abzubilden und dem Anwender die Möglichkeit zu geben, diese zu bearbeiten.

Die folgenden Abschnitte beschreiben die grundsätzliche Vorgehensweise beim Abbilden von Auflistungsklassen in diesen Steuerelementen.

## Darstellen von Daten aus Auflistungen im ListView-Steuerelement

Wenn Sie sich für die tabellarische Darstellung der Daten von Auflistungsklassen entscheiden, ist die detaillierte Listendarstellung des **ListView**-Steuerelements genau das Richtige. Um eine Instanz des **ListView**-Steuerelements für diese Form der Darstellung vorzubereiten, verwenden Sie die folgenden Eigenschaften (`IvwAdressen` sei dabei im Folgenden der Name eines **ListView**-Steuerelements im Formular):

```
IvwAdressen.View = Details      'Einstellung des Steuerelements auf detaillierte Listendarstellung
IvwAdressen.FullRowSelect = True 'Selektierung der gesamten Zeile, nicht nur des ersten Elements
IvwAdressen.HideSelection = False 'Beim Verlieren des Fokus bleibt die Selektierung erhalten
IvwAdressen.GridLines = True     'Zwischen den Zeilen werden kleine Abtrennungslinien eingezeichnet
```

Diese Einstellungen können Sie natürlich auch mit dem Eigenschaftenfenster zur Entwurfszeit einstellen, so, wie es in den vergangenen und zukünftigen Beispielen dieses Kapitels auch der Fall ist.

## Spaltenköpfe

Die Details-Ansicht des ListView-Steuerelements zeichnet sich dadurch aus, dass Daten in mehreren Spalten pro Zeile angeordnet werden können. Die Spalten selbst werden durch die ColumnHeaderCollection-Auflistung<sup>3</sup> des Steuerelements eingerichtet und gesteuert. Um verschiedene Spalten dem Steuerelement hinzuzufügen, verfahren Sie beispielsweise wie folgt:

```
With Me.lvwAdressen
  With .Columns
    'Alle Spaltenköpfe löschen.
    .Clear()
    'Spaltenüberschriften einrichten.
    .Add("Spalte1", -2, HorizontalAlignment.Left)      'Linksbündige Darstellung in der Spalte
    .Add("Spalte2", -2, HorizontalAlignment.Right)     'Rechtsbündige Darstellung
    .Add("Spalte3", -2, HorizontalAlignment.Center)    'Zentrierte Darstellung
    .Add("Spalte4", -2)                                'Ohne Angabe: Standard ist linksbündig
  End With
End With
```

Mit der Columns-Eigenschaft erhalten Sie Zugriff auf die ColumnHeaderCollection-Auflistung, durch die Sie dann die einzelnen Spaltenköpfe wie bei einer »normalen« Auflistung mit der Add-Methode hinzufügen können. Die einfachste Methode, das zu erreichen, wird hier demonstriert: Als ersten Parameter bestimmen Sie den Text eines Spaltenkopfes, als zweiten seine Breite. Wenn Sie möchten, dass die Spaltenbreite automatisch an die Breite des Textes angepasst wird, übergeben Sie als Spaltenbreite den Wert -2. Wichtig dabei ist zu wissen, dass Sie die Spaltenbreiten abermals auf -2 setzen müssen (obwohl sich ja eigentlich dieser Wert bereits in der Eigenschaft befindet), nachdem Sie die Liste mit Daten gefüllt haben. Dann wird die Layout-Logik abermals für jede Spalte aufgerufen; dieses Mal werden die Breiten der Spalten dann jedoch aufgrund der Breite der Spaltenköpfe *und* aufgrund der Breite der einzelnen Listeneinträge neu berechnet und visuell angepasst.

Schließlich können Sie einen dritten Parameter übergeben, der die Ausrichtung der Texte in den Spalten bestimmt.

---

**HINWEIS:** Beachten Sie, dass die erste Spalte Texte grundsätzlich nur linksbündig darstellen kann, egal, welchen Parameter Sie hier angeben. Wenn Sie keinen Parameter übergeben, wird die linksbündige Darstellung der Texte in der entsprechenden Spalte angenommen.

---

## Hinzufügen von Daten

Jede Zeile eines ListView-Steuerelements wird durch eine ListViewItem-Instanz repräsentiert. Ein ListViewItem-Objekt enthält also die Texte jeder Spalte einer Zeile, die im ListView-Steuerelement dargestellt werden sollen. Den Text, der in der ersten Spalte dargestellt wird, bestimmt die Text-Eigenschaft eines ListViewItem-Objekts.

---

<sup>3</sup> Kleine Anmerkung am Rande: Die Benennung einer Auflistung mit dem endenden Wortteil »Collection« widerspricht übrigens Microsofts Richtlinien zur Namensvergabe von Auflistungen – Microsoft verstößt bei der Namensgebung von ColumnHeaderCollection also gegen seine eigenen Richtlinien. Am besten fahren Sie mit Begriffen, bei denen Sie die Datenklasse im Singular benennen (Adresse), eine entsprechende Auflistung, die diese Daten als Menge verwaltet, im Plural (Adressen).

Diesen Text können Sie direkt beim Instanzieren einer Instanz dieses Objektes angeben. Die Texte aller weiteren Spalten einer Zeile werden durch die SubItems-Elemente des ListViewItem-Objekts festgelegt. Da es sich bei SubItems wieder um eine Auflistung handelt, können Sie die Texte der einzelnen Spalten ebenfalls mit deren Add-Methode einrichten, etwa wie im Folgenden zu sehen:

```
Dim locLvwItem As New ListViewItem("Text, Spalte 1")

'Die Untereinträge setzen
With locLvwItem.SubItems
    .Add("Text, Spalte 2")
    .Add("Text, Spalte 3")
    .Add("Text, Spalte 4")
End With
```

Wenn Sie eine ListViewItem-Instanz auf diese Weise aufbereitet haben, stellen Sie die komplette Zeile im ListView-Steuerelement dar, indem Sie diese Instanz der Items-Auflistung des ListView-Steuerelements hinzufügen, etwa mit:

```
'Zur Listview hinzufügen
lvwAdressen.Items.Add(locLvwItem)
```

## Beschleunigen des Hinzufügens von Elementen

Das Hinzufügen der Elemente auf die so beschriebene Weise erfolgt sichtbar, und das heißt: Sobald Sie ein Element hinzugefügt haben, wird der komplette Inhalt des ListView-Steuerelements neu gezeichnet. Bei hunderten von Elementen kann das sehr lange dauern – unnötigerweise.

Sie können mit zwei Methoden aufrufen – BeginUpdate sowie EndUpdate – festlegen, dass das Neuzeichnen der Elemente für die Dauer ihres Hinzufügens zur Liste ausgesetzt wird, und das funktioniert folgendermaßen:

```
'Unterdrückt Neuzeichnen-Ereignisse bis zum nächsten EndUpdate;
'dadurch geht der Aufbau der Elemente schneller und "wackelt" nicht.
Me.lvwAdressen.BeginUpdate()
'Hier fügen Sie alle Elemente der Liste hinzu:

'Aufbau der ListView ist beendet.
Me.lvwAdressen.EndUpdate()
```

## Automatisches Anpassen der Spaltenbreiten nach dem Hinzufügen aller Elemente

Nachdem alle Elemente der Liste hinzugefügt wurden, können Sie mit einem kleinen Trick dafür sorgen, dass sich die Spaltenbreiten automatisch an die Breiten der Texte bzw. der Spaltenköpfe anpassen (die längsten Texte in den Köpfen und Datenzeilen einer jeden Spalte werden als Maß verwendet). Dazu setzen Sie die Breiten aller Spalten einfach komplett nochmals auf den Wert -2. In Form von Code sieht das so aus:

```
'So werden die Spaltenbreiten optimal angepasst.
For Each locCol As ColumnHeader In Me.lvwAdressen.Columns
    locCol.Width = -2
Next
```

Sie sollten diese Aktion auch zwischen `BeginUpdate` und `EndUpdate` durchführen, damit ein Neuzeichnen des gesamten Steuerelements erst anschließend stattfindet.

## Zuordnen von ListViewItems und Objekten, die sie repräsentieren

Anders als bei der einfachen `ListBox`, der Sie eine komplette Objektreferenz als Listeneintrag selbst übergeben können, deren Textdarstellung anschließend über die `ToString`-Funktion des Objektes geregelt wird, und bei der eine selektierte Zeile auch gleichzeitig einem selektierten Objekt entspricht, gibt es beim `ListView`-Steuerelement keine automatische Zuordnung zwischen einem Eintrag in der Liste und einem Objekt, das diese Zeile visuell darstellen soll. Die Frage lautet also: Wenn der Anwender eine Zeile angeklickt hat, wie kann ich dann herausfinden, welchem Objekt sie ursprünglich entsprach?

Um diesem Problem zu entgehen, verfügt jedes `ListViewItem`-Objekt, das eine Zeile der `ListView` darstellt, über eine `Tag`-Eigenschaft (sprich: »Tähg«, etwa: Kennzeichnung, Markierung). Diese Eigenschaft kann beliebige Objekte, sprich: »alles« aufnehmen. Der Trick besteht also darin, aus einem Objekt Texte für die Darstellung über eine `ListView`-Instanz zu generieren und die Objektreferenz zusätzlich noch in der `Tag`-Eigenschaft eines `ListViewItem` zu speichern.

Wenn Sie später eine bestimmte Zeile der `ListView` als `ListViewItem` über die `Items`-Auflistung abrufen, steht Ihnen so auch das Ursprungsobjekt, aus dem die Texte für die `ListViewItem`-Instanz entstanden sind, wieder zur Verfügung.

Eine Zuordnung eines beliebigen Objekts sieht in etwa wie folgt aus:

```
'Zum Wiederfinden: Referenz in Tag speichern
locLvwItem.Tag = locElement
```

Wenn Sie später wieder an das Ausgangsobjekt herankommen wollen, brauchen Sie es nur aus der `Tag`-Eigenschaft wieder auszulesen (und im Bedarfsfall in den Ursprungstyp zurückzucasten). Das folgende Codeschnipsel macht genau das (Voraussetzung dafür ist natürlich, dass mindestens ein Element in der `ListView` selektiert wurde):

```
'Das ursprüngliche Adresse-Objekt aus der Liste holen.
locElement = DirectCast(lvwAdressen.SelectedItems(0).Tag, ElementType)
```

Übrigens: Auch bei komplexen Objekten bedeutet diese Vorgehensweise natürlich kein Aasen mit Arbeitsspeicher, weil, wie Sie im Klassenteil dieses Buches erfahren konnten, natürlich nur eine *Referenz* auf das eigentliche Objekt gespeichert wird. Da es die `Tag`-Eigenschaft sowieso gibt (und der entsprechende Speicher für eine Referenz sowieso reserviert wird, auch wenn sie standardmäßig auf `Nothing` »zeigt«), kostet das Speichern jedes Objektes in der `Tag`-Eigenschaft jedes `ListViewItem` der Liste kein einziges Byte an zusätzlichen Speicher.

## Feststellen, dass ein Element der Liste selektiert wurde

Wenn Sie eine ListView verwenden, wird der Anwender irgendwann einmal ein Element dieser Liste auswählen, um es beispielsweise bearbeiten zu können. In diesem Fall muss Ihre Anwendung natürlich nicht nur wissen, welches Element angeklickt wurde, sondern unter Umständen auch, wann dieses Ereignis auftrat, um zum Beispiel eine bestimmte Statusinformation (etwa in einer Statuszeile des Fensters) zu aktualisieren.

Mit dem SelectedIndexChanged-Ereignis informiert Sie ein ListView-Steuerelement, dass sich die Elementselektierungen in der Liste geändert haben. Das Ereignis wird immer dann ausgelöst, wenn ...

- ... ein Element selektiert wurde und zuvor keines selektiert war
- ... kein Element mehr selektiert ist, vorher aber mindestens ein Element selektiert war
- ... ein weiteres Element selektiert wurde, wenn das ListView-Steuerelement zuvor mithilfe der MultiSelect-Eigenschaft so eingestellt wurde (True), dass mehrere Elemente in der Liste selektiert werden konnten.

---

**HINWEIS:** Wenn Sie sehr zeitintensive Benutzeroberflächenaktualisierungen durchführen müssen, sobald sich die Selektierung in der Liste ändert, beachten Sie dabei, dass dieses Ereignis beim Wechsel einer Elementselektierung zweimal ausgelöst wird: Einmal für die Deselektierung des ersten Elements und ein anderes Mal für die Selektierung des neuen Elements.

---

Mit dem ItemSelectionChanged-Ereignis (neu im Framework 2.0) werden Sie darüber hinaus informiert, wenn sich der Auswahlzustand eines Elementes ändert.

Die Feststellung, welche Elemente im ListView-Steuerelement selektiert sind, kann mithilfe zweier Eigenschaften erfolgen:

- Die SelectedItems-Eigenschaft enthält eine Auflistung aller Elemente, die zurzeit im ListView-Steuerelement selektiert sind.
- Die SelectedIndices-Eigenschaft enthält eine Auflistung aller Indizes der Elemente, die zurzeit im ListView-Steuerelement selektiert sind.

Der folgende Code implementiert eine Routine zum Löschen aller selektierten Elemente im ListView-Steuerelement aus einer korrelierenden Auflistung, die die Ursprungsobjekte zur Darstellung im ListView-Steuerelement bereithält.

```
If lvwAdressen.SelectedItems IsNot Nothing AndAlso lvwAdressen.SelectedItems.Count > 0 Then
    For Each lvwItem As ListViewItem In lvwAdressen.SelectedItems
        myAdressen.Remove(DirectCast(lvwItem.Tag, Adresse))
    Next
    'Hier wird die Liste neu aufgebaut:
    ElementeDarstellen()
End If
End If
```

## Selektieren von Elementen in einem ListView-Steuerelement

Wichtig für Anwendungen ist es nicht nur, herausfinden zu können, welche Elemente eines ListView-Steuerelements selektiert sind. Hier und da müssen auch Elemente programmtechnisch selektiert werden.

---

**HINWEIS:** Hierfür können Sie die SelectedItems-Eigenschaft *nicht* verwenden, da sie nur die selektierten Elemente eines ListView-Steuerelements *widerspiegelt*. Wenn Elemente eines ListView-Steuerelements selektiert werden sollen, erreichen Sie das ausschließlich über die Selected-Eigenschaft eines ListViewItem-Elements der Liste.

---

Der Code, um Elemente eines ListView-Steuerelements zu selektieren, in deren Tag-Eigenschaft sich Objekte einer korrelierenden Auflistung befinden, lautet folgendermaßen:

```
'Alle zu selektierenden Elemente durchlaufen, und...
For Each locAdresse As Adresse In zuSelektierendeAdressenObjekte

    'jeweils alle ListView-Elemente durchsuchen und überprüfen, ob ...
    For Each locLvwItem As ListViewItem In Me.lvAdressen.Items

        '... die Tag-Referenz der Referenz des gesuchten Objekts entspricht.
        If locLvwItem.Tag Is locAdresse Then

            'Gefunden! ListView-Element markieren,
            locLvwItem.Selected = True

            'und wir müssen in der ListView
            'nicht weitersuchen!
            Exit For
        End If
    Next
Next
```

---

**TIPP:** Wenn Sie möchten, dass ein bestimmtes Element nicht nur selektiert wird, sondern es sich darüber hinaus auch auf jeden Fall im sichtbaren Bereich des ListView-Steuerelements befinden soll, verwenden Sie die EnsureVisible-Methode des entsprechenden ListViewItem-Objekts.

---

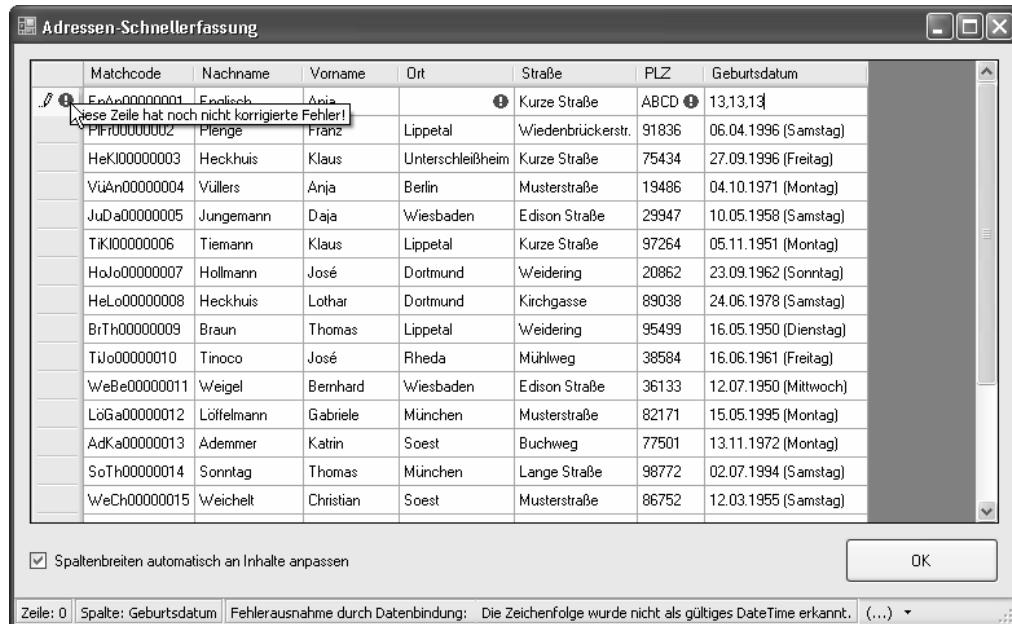
## Verwalten von Daten aus Auflistungen mit dem DataGridView-Steuerelement

Wenn es um die reine Darstellung von Daten einer Auflistung geht, verrichtet das ListView-Steuerelement sicherlich gute Dienste, zumal es bei der Anzeige auch vieler Daten eine gute Performance an den Tag legt.

Flexibler für die Darstellung von Daten, wenn auch nicht so performant, ist ein Steuerelement, das neu mit dem Framework 2.0 eingeführt wurde: das DataGridView-Steuerelement. Der Name dieses Steuerelements ist vielleicht ein wenig irreführend, weil es durch die Silbe »View« (etwa: *Ansicht, Darstellung, Ausblick*) im Namen die Vorstellung erweckt, es könne Daten lediglich *anzeigen*. Doch das ist überhaupt nicht der Fall.

Ganz im Gegenteil: Mit dem DataGridView-Steuerelement legt Ihnen Microsoft eine visuelle Komponente in die Hände, die an Mächtigkeit in Sachen Datendarstellung und Datenerfassung eigentlich nicht mehr zu übertreffen ist.

Doch es ist wie bei allen Dingen des Lebens: Wo viel Licht ist, ist auch viel Schatten. Um bei dieser Analogie zu bleiben: Das DataGridView-Steuerelement ist ein 10.000-Watt-Suchscheinwerfer; deswegen gibt es auch eine Menge Falltüren, die es clever zu umschiffen gilt, wenn Sie für den Komfort des Anwenders das Letzte aus diesem Steuerelement herausholen wollen.



**Abbildung 27.12:** So komfortabel sollte eine Datenerfassungsanwendung mindestens sein, die Sie den Anwendern Ihrer Software über das DataGridView-Steuerelement zur Verfügung stellen

Um es vorweg zu sagen: Dem DataGridView-Steuerelement könnte man locker nicht nur ein eigenes Kapitel, sondern fast schon ein kleines Buch widmen, und wenn ich persönlich auch dieses Thema extrem spannend finde, so muss ich mich aus Platzgründen dennoch auf Wesentliches zu diesem Thema beschränken. Neben den grundlegendsten Dingen, die Sie für den Start benötigen, habe ich deswegen versucht, möglichst viele der Themen herauszuarbeiten, die Sie nicht ohnehin aus der Online-Hilfe erfahren könnten.

Und das impliziert meinen ersten Vorschlag: Die Lektüre der Online-Hilfe zum Thema DataGridView enthält wirklich umfangreiches und gar nicht so trocken geschriebenes Material. Wenn Sie sich viel mit Datenbindung und Datenaufbereitung in Ihren Anwendungen beschäftigen müssen, sollten Sie sich wirklich einen halben Tag Zeit nehmen und die Online-Hilfe von Visual Basic zu diesem Thema gründlich studieren.

Um Daten in einem DataGridView-Steuerelement anzeigen oder bearbeiten zu können, stehen Ihnen folgende Vorgehensweisen zur Verfügung:

- Sie sorgen komplett per Programmcode für das Einrichten der verschiedenen Spalten, für die Datentypen, die in den Zellen der einzelnen Spalten verarbeitet werden, für das Anzeigen der Daten in den Zellen und für die Aufbereitung und spätere Übernahme in den Datenklassen ihrer Anwendung.
- Sie binden das DataGridView-Steuerelement an eine Datenquelle, sorgen sich nicht um Tabelleneinrichtung, Datendarstellung und Datenüberprüfung und programmieren gar nicht.
- Sie finden den goldenen Mittelweg, verwenden die Datenbindung, aber greifen bei entscheidenden Prozessen per Programmcode in das Geschehen ein. Die folgenden Abschnitte und darin enthaltenen Beispiele gehen diesen Mittelweg, da er den meiner Meinung nach besten Kosten-/Nutzen-Faktor darstellt.

## Über Datenbindung, Bindungsquellen, die BindungSource-Komponente und warum Currency nicht unbedingt Währung heißt

Ganz simpel ausgedrückt: Bei der Datenbindung von Framework-Komponenten wird die Eigenschaft einer konsumierenden Datenquelle mit der Eigenschaft einer Daten vorgebenden Komponente verknüpft. Ändert sich der Wert der Eigenschaft der letzteren, ändert sich gleichzeitig der Wert der ersten – im Bedarfsfall auch umgekehrt.

Bei größeren Bindungsszenarien, die nicht nur auf Eigenschaften (also Datenfeldebene) beschränkt sind, bedeutet das beispielsweise für ein DataGridView-Steuerelement: Man kann auch eine ganze DataGridView-Instanz an eine Datenquelle binden. Die Datenquelle liefert *Datenlisten*, die DataGridView zeigt sie an, erlaubt das Bearbeiten, und die Änderungen laufen im Bedarfsfall wieder zurück in die Datenquelle. Eine Datenquelle kann dabei eine Komponente einer Datenbank sein. Beispielsweise eine Access-Datentabelle. Oder besser: Eine Tabelle oder ein Resultset<sup>4</sup> einer SQL Server-Datenbank. Oder, und das ist das Tolle, auch etwas, was man im Microsoft-Jargon als *Business Object* (Geschäftsobjekt) bezeichnet, und dabei handelt es sich in der Regel um eine Auflistungsklasse, wie Sie sie beispielsweise in unserer ständig wachsenden Adresso-Anwendung kennen gelernt haben. Nicht nur einzelne Eigenschaften werden also dabei gebunden, sondern mehrere Objekte, die ihrerseits über mehrere Eigenschaften verfügen. Eine Tabellenzeile stellt also genau ein Objekt einer Auflistung dar (oder, im Falle von Datenbanken, einen Datensatz). Eine Tabellenzelle entspricht einer Eigenschaft eines Objektes der Auflistung (oder dem Inhalt eines Datenfeldes eines Datensatzes im Fall von Datenbanken). Die ganze DataGridView-Tabelle entspricht der gesamten Auflistung (oder – bei Datenbanken – einem kompletten Resultset). Beim Binden von Auflistungen spricht man übrigens nur von Datenbindungen, beim Binden von Datenbanktabellen von *komplexen* Datenbindungen.

Doch egal, welches Objekt an welches andere Objekt gebunden wird, sie benötigen immer eine vermittelnde Instanz. Im Fall von simplen Eigenschaftsverknüpfungen übernimmt diese Aufgabe das so genannte *PropertyManager*-Objekt; im Falle von Listen das *CurrencyManager*-Objekt.

---

<sup>4</sup> ... was soviel wie *Abfrageergebnissatz* bedeutet und einer wie auch immer gearteten Datentabelle entspricht.

---

**HINWEIS:** Lassen Sie sich dabei nicht von dem englischen Begriff *Currency* ins Bockshorn jagen, für den Sie vielleicht nur die deutsche Übersetzung *Währung* parat haben. *Currency* bedeutet nämlich auch *Zeitnähe* (von *current*, etwa: *aktuell*); ein *CurrencyManager* ist also eine überwachende Instanz, die dafür sorgt, dass Dinge *zeitnah* passieren – im .NET-Framework die *zeitnahe* Umsetzung der Datenbindung.

---

Bis zur Version 1.1 verlief das Binden einer Komponente an eine Datenquelle, die mehrere Listenelemente anbot, dubios und irgendwie im Hintergrund, ohne dass man so recht wusste, was genau passiert. Was wirklich passierte, war: Beim Zuweisen einer Datenquelle an eine Komponente – beispielsweise für eine Listendarstellung dieser Elemente – wurde implizit ein *CurrencyManager*-Objekt erstellt. Dieses *CurrencyManager*-Objekt sorgte dann dafür, dass – sehr vereinfacht ausgedrückt – beim Editieren einer Zeile auch das korrelierende Objekt in der Auflistung »getroffen« wurde.

Im .NET 2.0-Framework ist dieser Vorgang nicht nur offensichtlicher geworden – er wurde auch um Designer-Unterstützung ergänzt: Sie können mithilfe des Designers eine Datenquelle erstellen, oder besser: dem Designer mitteilen, welches Objekt (Datenbank, Auflistungsklasse) eine Datenquelle *sein soll*. Und dann können Sie interaktiv diese Datenquelle an eine Komponente wie beispielsweise eine *DataGridView*-Instanz binden. Der Designer erstellt dabei eine so genannte *BindingSource*-Komponente (auf deutsch etwa *Bindungsquelle*, eine Art *CurrencyManager XXL*), und legt diese Komponente im Komponentenfach ab.

Diese *BindingSource*-Komponente ist also im .NET-Framework 2.0 das Bindeglied zwischen der die Daten anbietenden und der die Daten konsumierenden Komponente. Damit das reibungslos funktioniert, stellt sie (ganz ähnlich wie früher *CurrencyManager*) Methoden wie *MoveNext* (*zum Nächsten bewegen*), *MovePrevious* (*zum Vorherigen bewegen*), *AddNew* (*Neuen anlegen*) und Ähnliches bereit. Und wozu? Ganz einfach:

Wenn Sie später beim Bearbeiten von Daten in Ihrer *DataGridView*-Instanz mit dem Cursor eine Zeile nach unten fahren, dann ruft die *DataGridView* die *MoveNext*-Methode der *BindingSource* auf. Die *BindingSource* sorgt dann dafür, dass ein interner Zeiger auf die einzelnen Elemente der Datenquelle ebenfalls weitergerückt wird. Die *DataGridView* kann sich also das nächste (*next*) Objekt aus der Datenquelle holen – im Falle einer Auflistung eben das nächste Objekt in der Liste.

Fahren Sie mit dem Cursor in der Tabelle eine Zeile nach oben, ruft die *DataGridView*-Instanz *MovePrevious* der *BindingSource* auf, und sie bekommt, um beim Beispiel zu bleiben, wieder über den Vermittler *BindingSource*, das vorherige (*previous*) Objekt der Auflistung.

Dieser Fall geht auch umgekehrt. Wenn Sie eine Objektauflistung als Datenquelle über eine *BindingSource*-Instanz an eine *DataGridView*-Instanz gebunden haben, können Sie auch selber *MoveNext* der *BindingSource* aufrufen, um einerseits den Zeiger zum nächsten Objekt der Datenaufstellung, andererseits den Cursor der *DataGridView*-Instanz eine Zeile nach unten zu verschieben. Und ein Aufruf von *MovePrevious* macht das Gleiche bei beiden involvierten Komponenten, nur in die andere Richtung. Übrigens: Eine weitere im .NET-Framework 2.0 neue Komponente macht von dieser Methode regen Gebrauch, und sie nennt sich *BindingNavigator*. ► Kapitel 32 stellt sie im Rahmen von ADO.NET-Datenbindungen am einfachen Beispiel vor.

---

**HINWEIS:** Das ListView-Steuerelement sieht das Binden von Auflistungsdaten über eine BindingSource-Komponente übrigens nicht vor.

---

Wie das interaktive Erstellen einer solchen Korrelation in der Praxis ausschaut, und wie einfach das vonstatten geht, zeigt der folgende Abschnitt.

## Erstellen einer gebundenen DataGridView mit dem Visual Basic-Designer

Für das folgende Beispiel können Sie zwei Beispielprojekte der Begleitdateien verwenden – ein fix und fertiges und eines, mit dem Sie die folgenden Schritte nachvollziehen können.

---

**BEGLEITDATEIEN:** Sie finden letzteres im Verzeichnis `.\VB 2005 - Entwicklerbuch\G - SmartClient\Kap27\Adresse03 - zum Experimentieren`. Das fertige Beispiel befindet sich im Verzeichnis `\VB 2005 - Entwicklerbuch\G - SmartClient\Kap27\Adresse03`.

---

- Öffnen Sie das Projekt und rufen Sie das schon teilweise vorbereitete Formular `frmSchnellerfassung.vb` im Designer auf. Ihnen bietet sich anschließend ein Bild, wie Sie es in etwa auch in Abbildung 27.13 sehen können.

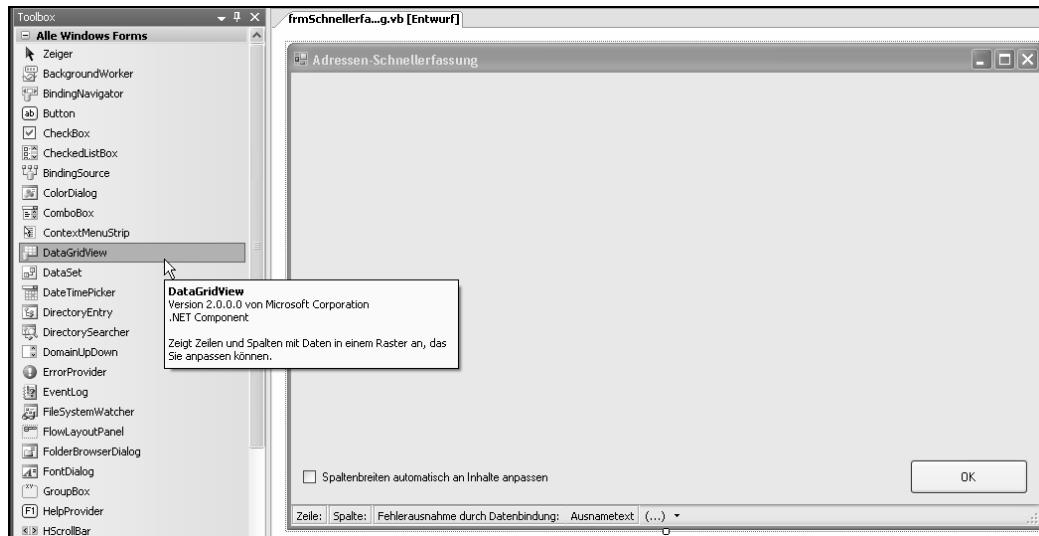
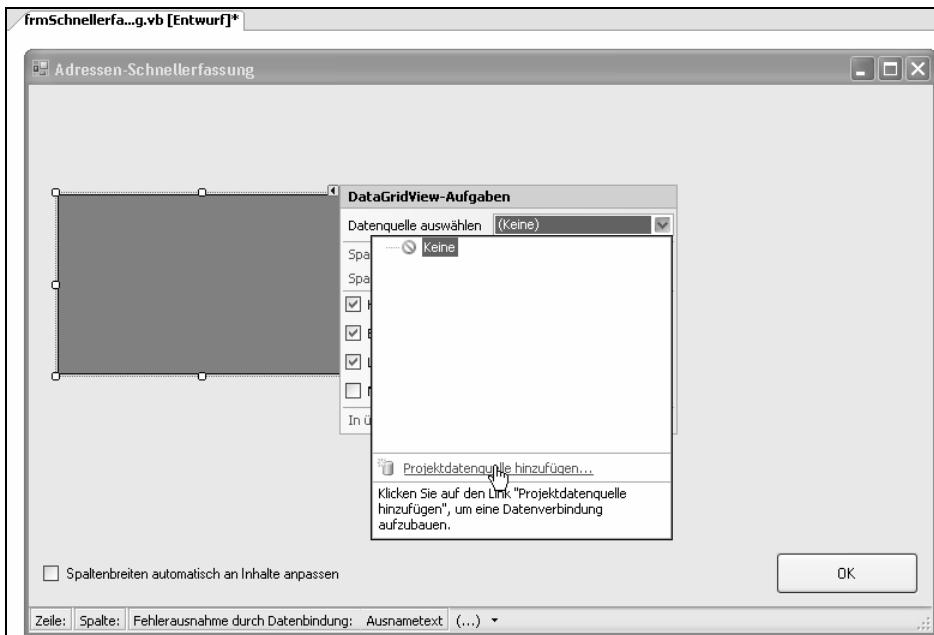
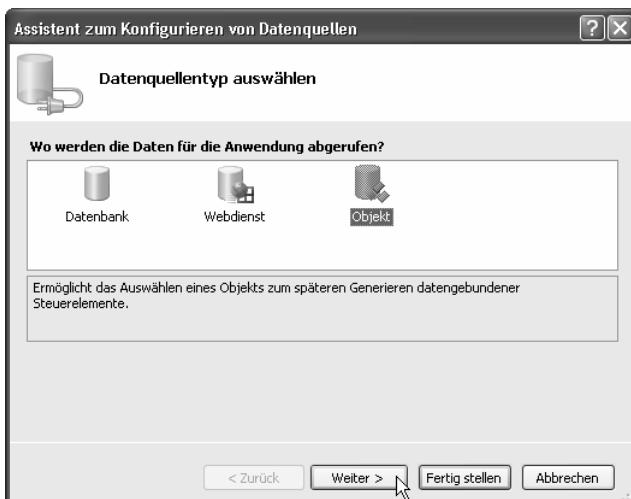


Abbildung 27.13: Suchen Sie das DataGridView-Steuerelement in der Toolbox, und ziehen Sie es ins Formular



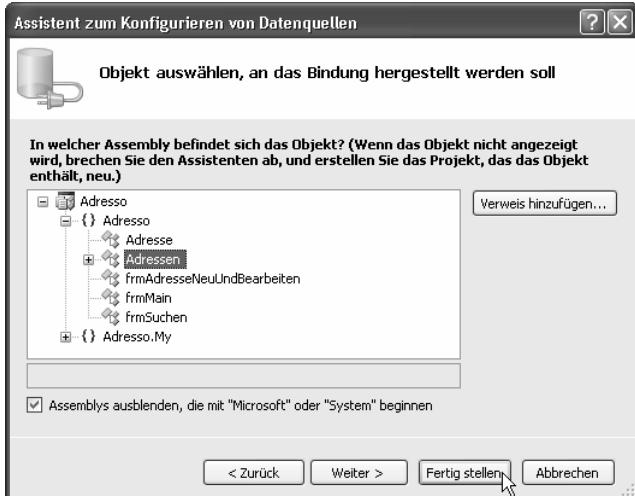
**Abbildung 27.14:** Wählen Sie aus der Aufgabenliste der DataGridView als Erstes *Datenquelle auswählen*, und klicken Sie – da es noch keine definierten Datenquellen gibt – auf den Link *Projektdatenquelle hinzufügen*



**Abbildung 27.15:** Der Assistent zum Konfigurieren von Datenquellen hilft Ihnen beim Erstellen einer neuen Datenquelle auf Auflistungsklassenbasis. Dazu wählen Sie im Assistentendialog das *Objekt*-Element.

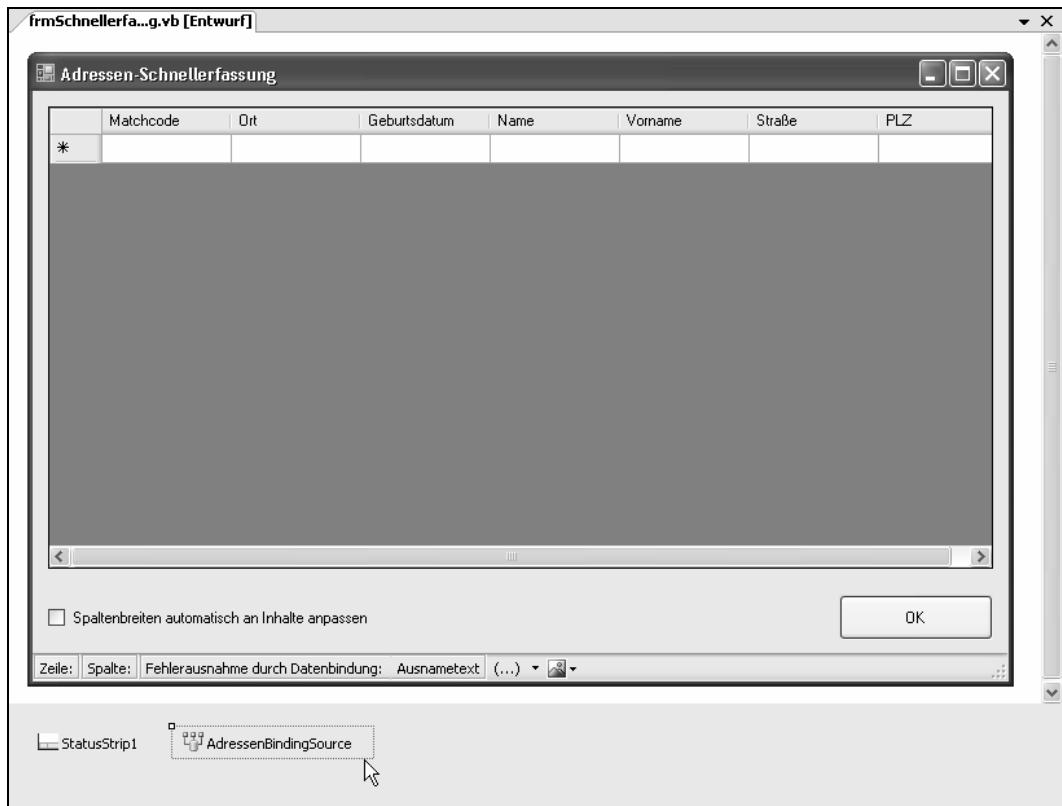
- Suchen Sie in der Toolbox das DataGridView-Steuerelement. Ziehen Sie es anschließend ins Formular. Die Aufgabenliste des DataGridView-Steuerelements wird jetzt angezeigt.
- Klappen Sie die Liste *Datenquelle auswählen* auf.

- Da es noch keine Projektdatenquellen gibt, klicken Sie auf den Link *Projektdatenquelle hinzufügen*, um eine neue Datenquelle einzurichten. Ziel dabei wird es in den folgenden Schritten sein, die Adressen-Klasse, die die Auflistungsklasse für alle Adressen unserer kleinen Adressverwaltung darstellt, zur Datenquelle für das DataGridView-Steuerelement zu promovieren.
- Der Visual Basic-Designer zeigt nun einen Assistenten an, mit dem Sie Datenquellen konfigurieren können. Im ersten Assistentenschritt, etwa wie in Abbildung 27.15 zu sehen, wählen Sie als Datenquellentyp *Objekt*.



**Abbildung 27.16:** Bestimmen Sie in der TreeView, wo in der Objekt- und Namespace-Hierarchie Ihres Projektes die Klasse, die als Datenquelle fungieren soll, zu finden ist

- Bestätigen Sie diesen Schritt des Assistenten mit *Weiter*.
- Im nächsten und auch schon letzten Schritt (Abbildung 27.16) bestimmen Sie, wo innerhalb Ihrer Projektmappe sich das Objekt (sprich: die Auflistungsklasse) befindet, dessen Instanz als Datenquelle verwendet werden soll. Klappen Sie die entsprechenden Zweige in der Treeview auf, und selektieren Sie die *Adressen*-Auflistungsklasse.
- Klicken Sie anschließend auf *Fertig stellen*, um den Assistenten zu beenden.
- Wenn Sie das DataGridView-Steuerelement anschließend vergrößern, und es mehr oder weniger flächenfüllend auf dem Formular anordnen, erkennen Sie, dass ein Prozess stattgefunden haben muss, der die Adressenklasse analysiert und aus den verschiedenen Eigenschaften eines Elementes (Adresse) die Grobdefinition für die Tabelle erstellt hat (Abbildung 27.17).



**Abbildung 27.17:** Nach dem Vergrößern des *DataGridView*-Steuerelements können Sie sehen, dass die Schemainformationen der *Adresse*-Klasse als Basis für das Einrichten der Tabellenspalten diente

In dieser Abbildung sehen Sie ebenfalls, dass der Designer die *BindingSource*-Komponente *AdressenBindingSource* erstellt und im Komponentenfach des Formulars untergebracht hat. Und diese stellt ab sofort das Bindeglied zwischen der Auflistungsklasse *Adressen* und dem *DataGridView*-Steuerelement dar.

Das Beste ist: Mit einer einzigen Zeile Code können Sie nun dafür sorgen, dass die Daten im *DataGridView*-Steuerelement angezeigt werden können.

- Dazu wechseln Sie zum Hauptformular des Projektes *frmMain.vb*, und öffnen Sie dieses im Designer.
- Klicken Sie im *MenuStrip*-Steuerelement des Formulars zunächst auf *Bearbeiten*, und doppelklicken Sie anschließend auf *Schnellerfassung von Adressen*.

Im Codeeditor können Sie nun die folgenden Zeilen eingeben, sodass sich für die Ereignisbehandlungsroutine Folgendes ergibt:

```

Private Sub tsmSchnellerfassung_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles tsmSchnellerfassung.Click

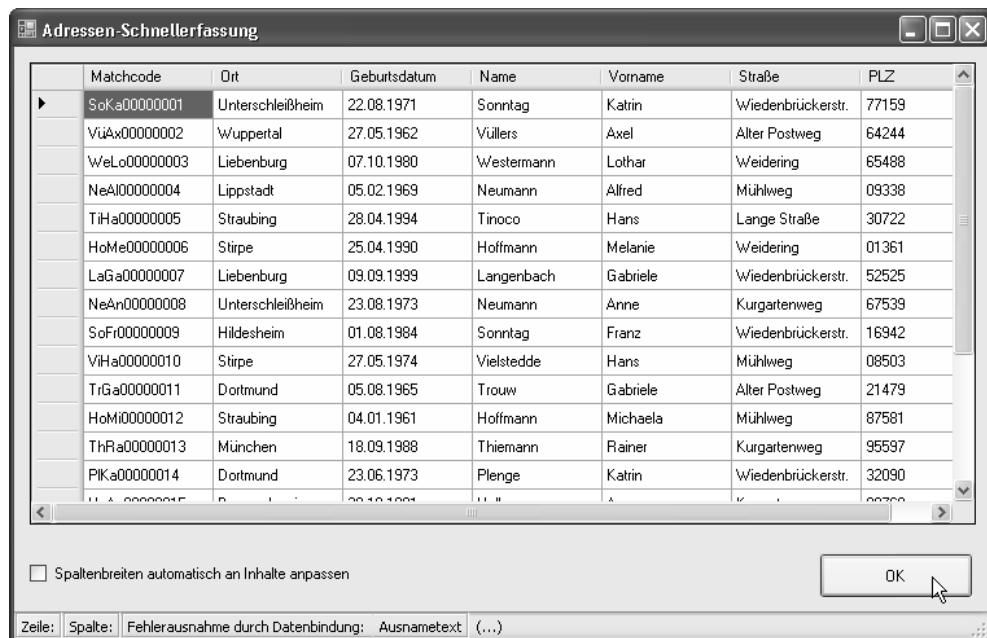
    'Neue Instanz des Dialogs
    Dim locfrm As New frmSchnellerfassung

    'Eine Zeile reicht aus, um die Adressen darzustellen!
    Locfrm.AdressenBindingSource.DataSource = myAdressen

    'Dialog modal darstellen
    Locfrm.ShowDialog()
End Sub

```

Wenn Sie diese Zeilen eingegeben haben und das Programm anschließend starten, wählen Sie zuerst aus dem Menü *Datei* den Menüpunkt *Zufallsadressen einfügen*. Wählen Sie anschließend aus dem Menü *Bearbeiten* den Menüpunkt *Schnellerfassung von Adressen*. Was Sie anschließend sehen, sollte nicht nur Abbildung 27.18 entsprechen, sondern Sie vor Begeisterung auch umhauen ...

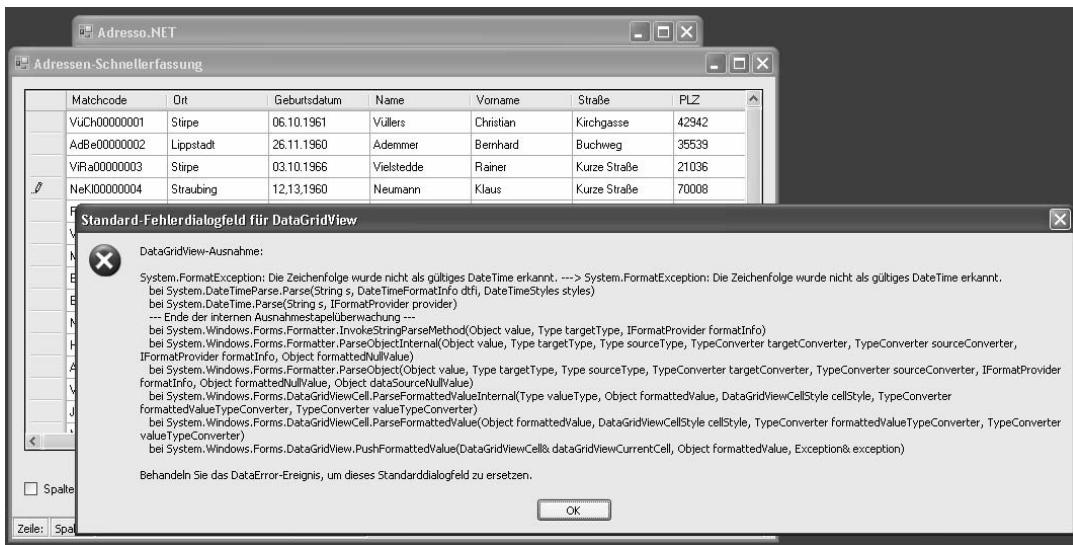


**Abbildung 27.18:** Mit einer einzigen Codezeile steht Ihnen ein komfortabler Editor zur Bearbeitung des Adress-Pools zur Verfügung

Doch der Teufel steckt wie immer im Detail, denn es gibt noch ein paar Punkte, die keinesfalls dem entsprechen, was man unter einer komfortablen Benutzeroberfläche verstehen könnte:

- Die Datenspalten sind willkürlich angeordnet; wir sollten versuchen, eine plausible Ordnung in die Datenspalten zu bekommen.
- Die Statuszeile sollte als Navigationshilfe die Position des Cursors im DataGridView-Steuerelement anzeigen.

- Sie können in jedem Feld jede beliebige Eingabe durchführen – Eingaben werden überhaupt nicht kontrolliert, und bei Feldern, die datentypbedingt ein bestimmtes Eingabeformat verlangen, kann man dabei auf die Nase fallen (siehe Abbildung 27.19).



**Abbildung 27.19:** Solche Fehler sollten in der Release-Version Ihrer Anwendung nach Möglichkeit nicht mehr vorkommen

- Bei unserem Beispiel sollte der Matchcode nicht editierbar, auf jeden Fall aber neu eingebbar sein. Und: Auch hier dürfen doppelte Matchcodes nicht vorkommen und müssen im Rahmen einer Eingabeprüfung bei Neueingaben von Adressen ausgeschlossen werden.
- Und schließlich gibt es auch noch eine kleinen, aber gemeinen Störfaktor bei der Eingabe: Wenn Sie eine Zelle editiert haben und Ihre Änderungen mit **Eingabe** bestätigen, springt der Cursor nicht in das nächste Feld rechts des gerade bearbeiteten Feldes, sondern in das darunter. Damit ist der Anwender gar nicht in der Lage, Adressen eine nach der anderen (also zeilenweise) neu zu erfassen.

Die nächsten Abschnitte beschreiben, wie Sie mit gar nicht soviel Codierungsaufwand diese Ziele erreichen können.

## Sortieren und Benennen der Spalten eines DataGridView-Steuerelements

Wenn Sie ein DataGridView-Steuerelement an eine Datenquelle gebunden haben, werden die Schema-informationen (welche Datenspalten mit welchen Datentypen gibt es) aus der Datenquelle ermittelt. Auf Basis dieser Schemainformationen werden die Spaltenköpfe eingerichtet. Programmtechnisch rufen Sie die Spaltendefinitionen einer Tabelle mithilfe der `Columns`-Eigenschaft (vom Typ `DataGridViewColumnCollection`) ab, die eine Auflistung mit `DataGridViewColumn`-Objekten enthält. Ein einzelnes `DataGridViewColumn`-Objekt dieser Auflistung bestimmt letzten Endes die Beschaffenheit einer Spalte, indem sie Spaltennamen, Beschriftung, darzustellenden Typ und Weiteres definiert.

Die Einrichtung bzw. Modifizierung nehmen Sie am einfachsten mit dem Designer vor. Die folgende Anleitung beschreibt, wie Sie ...

- ... definieren, welche Operationen (*Neuer Datensatz*, *Datensatz bearbeiten*, *Datensatz löschen*, *Umsortieren von Spalten*) mit dem DataGridView-Steuerelement durchgeführt werden dürfen,
- ... Spaltenreihenfolgen anordnen,
- ... die Beschriftung und Benennung von Spalten ändern,
- ... die Datentypen von Spalten einrichten bzw. verändern können.

---

**BEGLEITDATEIEN:** Um die Schritte der folgenden Unterabschnitte nachzuvollziehen, die zunächst noch den Visual Basic-Designer involvieren, verwenden Sie das bisher verwendete Beispiel im Verzeichnis *.\VB 2005 - Entwicklerbuch\G - SmartClient\Kap27\Adresso03 - zum Experimentieren*. Dabei wird davon ausgegangen, dass Sie die Bindung der Datenquelle, wie im vorherigen Abschnitt beschrieben, bereits vorgenommen haben.

---

### Öffnen der Liste der häufigen Aufgaben

Das DataGridView-Steuerelement lässt sich natürlich wie jedes andere Steuerelement über das Eigenschaftenfenster des Visual Basic-Designers konfigurieren. Einfacher ist es allerdings, bestimmte Einstellungen über die Liste der häufigen Aufgaben auszuführen. Um diese Liste zu öffnen, verfahren Sie wie folgt:



**Abbildung 27.20:** Mit dem Smarttag des Steuerelements erreichen Sie die Liste der häufigen Aufgaben

- Klicken Sie auf das DataGridView-Steuerelement im Formular, um es zu selektieren.
- Klicken Sie auf den Smarttag des Steuerelements (der kleine nach rechts weisende Pfeil an der rechten oberen Ecke), um die Liste häufiger Aufgaben des Steuerelements zu öffnen.

### Bestimmen, welche Aufgaben mit dem DataGridView-Steuerelement erledigt werden dürfen

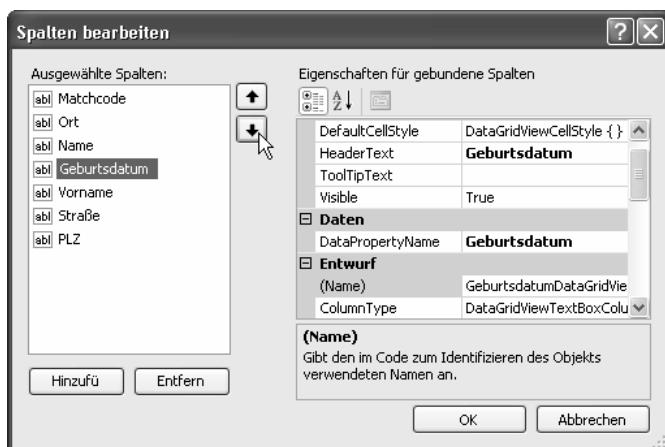
- Öffnen Sie die Liste häufiger Aufgaben durch Klick auf das Smarttag des zuvor selektierten DataGridView-Steuerelements (siehe Abbildung 27.20).
- Wählen Sie im Dialogfeld durch An- oder Abwählen der entsprechenden Kontrollkästchen, welche Funktionen das DataGridView-Steuerelement zu Verfügung stellen soll.

**HINWEIS:** Auch wenn Sie hier das Hinzufügen von Datensätzen erlauben, wird diese Funktion zur Laufzeit nur dann zur Verfügung stehen, wenn die AllowNew-Eigenschaft der entsprechenden BindingSource-Komponente (oder – bei programmtechnischer Verknüpfung – auch die CurrencyManager-Komponente) auf True gesetzt wurde.

Wenn Sie später zur Laufzeit feststellen, dass beide Eigenschaften auf True gesetzt wurden, Sie aber dennoch die Funktionalität zum Hinzufügen von Datensätzen vermissen, stellen Sie sicher, ob Sie die Daten einer Auflistung nicht versehentlich mithilfe ihrer DataSource-Eigenschaft *direkt* an das DataGridView-Steuerelement anstatt an die verknüpfte BindingSource-Komponente zugewiesen haben.<sup>5</sup>

## Anordnen der Spalten eines DataGridView-Steuerelements mit Designer-Unterstützung

- Öffnen Sie die Liste häufiger Aufgaben durch Klick auf das Smarttag des zuvor selektierten DataGridView-Steuerelements (siehe Abbildung 27.20).
- Klicken Sie auf Spalten bearbeiten. Sie sehen einen Dialog, etwa wie auch in Abbildung 27.21 zu sehen.



**Abbildung 27.21:** Mit den Pfeilschaltflächen (hier durch den Mauszeiger gekennzeichnet) ordnen Sie die Reihenfolge der Spalten neu an

- Mit den Pfeilschaltflächen ordnen Sie, wie in der Abbildung zu sehen, die Reihenfolge der Spalten des DataGridView-Steuerelements neu an. Der hier in der Liste ganz oben stehende Eintrag entspricht der ganz links im Steuerelement erscheinenden Spalte.

## Die Beschriftung und Benennung von Spalten ändern

Insgesamt drei Eigenschaften eines DataGridViewColumn-Objekts, das für die Eigenschaften einer DataGridView-Spalte zuständig ist, haben etwas mit Benennungen bzw. Beschriftungen zu tun. Gerade wenn die Spaltendefinitionen aus Schemainformationen einer Datenquelle stammen, kann es anfangs verwirren, welche Eigenschaften für welche Aufgabe zuständig sind.

Um diese Einstellungen mit dem Designer durchzuführen, verfahren Sie wie folgt:

<sup>5</sup> Und nun raten Sie mal, wieso dieses Kapitel einen halben Tag eher fertig gestellt sein könnten...

- Öffnen Sie die Liste häufiger Aufgaben durch Klick auf das Smarttag des zuvor selektierten DataGridView-Steuerelements (siehe Abbildung 27.20).
- Klicken Sie auf *Spalten bearbeiten*. Sie sehen einen Dialog, etwa wie auch in Abbildung 27.21 zu sehen.
- In der Eigenschaftenliste bestimmen Sie folgende Eigenschaften:
- **Headertext:** Legen Sie mit dieser Zeichenfolge fest, welche Überschrift die entsprechende Spalte im DataGridView-Steuerelement aufweisen soll.
- **DataPropertyName:** Bestimmen Sie mit dieser Zeichenfolge, welcher Objekteigenschaft der Datenquelle diese Spalte zugeordnet sein soll. Sie sollten diese Eigenschaft nur dann ändern, wenn sich die Eigenschaft eines Objektes der als Datenquelle fungierenden Auflistung geändert hat. Belassen Sie anderenfalls diese Eigenschafteneinstellung so, wie sie ist.
- **Name:** Bestimmen Sie mit dieser Zeichenfolge, mit welchem Textschlüssel Sie auf ein Element der **DataGridViewColumnCollection** oder der **DataGridViewCellCollection** (einer **DataGridViewRow**-Auflistung) zugreifen. Diese Eigenschafteneinstellung ist also dann wichtig, wenn Sie programmtechnisch entweder auf eine Spaltedefinition oder eine Zelle einer DataGridView-Instanz zugreifen wollen.

#### **Zugriff auf ein Element der DataGridViewColumnCollection (eine Spaltendefinition):**

```
'Index einer Spalte aus dem Spaltennamen ermitteln
myGebDatColumnIndex = dgvAdressen.Columns("GeburtsdatumDataGridViewTextBoxColumn").Index
```

#### **Zugriff auf ein Element der DataGridViewCellCollection (einer Zelle der Tabelle):**

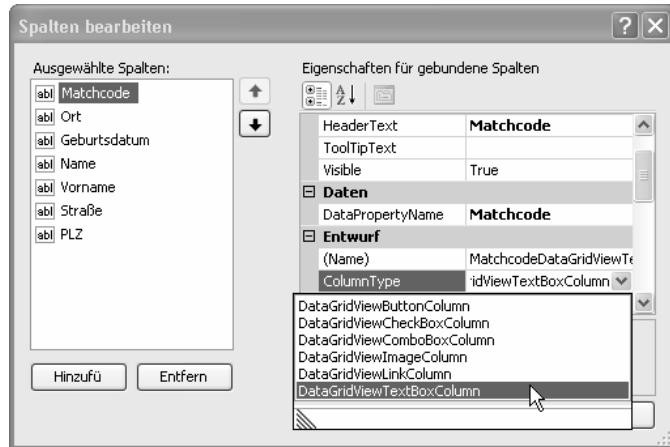
```
'Eine bestimmte Zelle aus Zeilennummer und Spaltennamen ermitteln
Dim locRow As DataGridViewRow = dgvAdressen.Rows(Zeilennummer)
Dim locCell As DataGridViewCell = locCurrentRow.Cells("MatchcodeDataGridViewTextBoxColumn")
```

#### **Bestimmen der Spaltentypen einer DataGridView-Instanz**

Mit der **ColumnType**-Eigenschaft des *Spalten bearbeiten*-Dialogs bestimmen Sie, mit welchem Steuer-elementtyp die Spalte ausgestattet werden soll.

---

**HINWEIS:** Die **ColumnType**-Eigenschaft bestimmt dadurch *nicht* notwendigerweise, welcher Datentyp in einer Spalte verarbeitet wird. So wählen Sie für numerische Werte, Datumswerte oder Texte als **ColumnType**-Eigenschaft wahrscheinlich immer **DataGridViewTextBoxColumn** (**DataGridViewComboBoxColumn** würden Sie dabei wählen, um aus einer vorgegebenen Liste mit Werten nur bestimmte zur Auswahl zuzulassen, oder wenn Sie eine Verknüpfung mit einer anderen Datentabelle herstellen wollen).



**Abbildung 27.22:** Mit der *ColumnType*-Eigenschaft bestimmen Sie den Steuerelementtyp (nicht Datentyp!), den eine *DataGridView*-Spalte verarbeiten soll

Insgesamt 6 verschiedene Spaltentypen stehen standardmäßig zur Verfügung:

ColumnType	Aufgabe
DataGridViewButtonColumn	Implementiert Schaltflächen in den Datenzeilen für die entsprechende Spalte
DataGridViewCheckBoxColumn	Implementiert Kontrollkästchen in den Datenzeilen für die entsprechende Spalte
DataGridViewComboBoxColumn	Implementiert Aufklapplisten in den Datenzeilen für die entsprechende Spalte
DataGridViewImageColumn	Implementiert Bilddarstellungen in den Datenzeilen für die entsprechende Spalte
DataGridViewLinkColumn	Implementiert einen Web-Link in den Datenzeilen für die entsprechende Spalte
DataGridViewTextBoxColumn	Implementiert ein Texteingabefeld in den Datenzeilen für die entsprechende Spalte

**Tabelle 27.1:** Die standardmäßig vorhandenen Steuerelemente, die als Darstellungs- und Anzeigeeinstanzen innerhalb von *DataGridView*-Tabellen fungieren können

**HINWEIS:** Sie werden die *ColumnType*-Eigenschaft vergeblich im *DataGridViewColumn*-Objekt suchen – es handelt sich nämlich um eine nennen wir sie mal: »virtuelle« Designereigenschaft, die sich nur auf die Erstellung des Designer-Codes (das nächste Kapitel erklärt mehr dazu) bezieht. Wenn Sie für ein gebundenes *DataGridView*-Steuerelement eine Spaltentypumstellung vornehmen wollen, platzieren Sie den entsprechenden Code dazu am besten direkt unterhalb von *InitializeComponent* im Standardkonstruktor des Formulars, das die *DataGridView* beinhaltet.

Programmtechnisch legen Sie das zu verwendende Steuerelement für die entsprechende Spalte mit der *CellTemplate*-Eigenschaft fest. Das Umdefinieren einer Spalte führen Sie am besten im Standardkonstruktor des Formulars durch, etwa wie folgt:

```
Public Class frmSchnellerfassung
    Sub New()
        ' Dieser Aufruf ist für den Windows Form-Designer erforderlich.
        InitializeComponent()
        ' Fügen Sie Initialisierungen nach dem InitializeComponent()-Aufruf hinzu.
    End Sub

```

```

'Eines der möglichen Steuerelemente für die Spalte manuell zuordnen.
dgvAdressen.Columns("NameDataGridViewTextBoxColumn").CellTemplate = New DataGridViewTextBoxColumn()
End Sub
.
.
.
End Class

```

**TIPP:** Wenn Sie eigene Steuerelemente für Tabellenzellen mit individuellen Verhaltensweisen implementieren wollen, leiten Sie entweder eines der vorhandenen Steuerelemente der oben stehenden Tabelle ab, erweitern es und binden es auf die hier beschriebene Weise ein. Oder Sie leiten es von der Basis aller Steuerelemente ab, die in Tabellenzeilen einer DataGridView zur Anwendung kommen können. Verwenden Sie dazu die abstrakte Basisklasse DataGridViewCell. Wichtig: Achten Sie dabei darauf, die Clone-Methode zu überschreiben und neu zu implementieren, da die Definition für alle Zellen einer Spalte aus der ersten Instanz erstellt wird, die man über die CellTemplate-Eigenschaft definiert. Dies gilt nur dann, wenn Sie einem vorhandenen Steuerelement weitere Eigenschaften hinzufügen, deren Inhalte natürlich zusätzlich zu den bereits vorhandenen kopiert werden müssen. Dies gilt aber auf jeden Fall, wenn Sie das Steuerelement auf Basis von DataGridViewCell komplett neu implementieren.

## Programmtechnisches Abrufen der aktuellen Zeile und aktuellen Spalte

Die aktuelle Zeile und die aktuelle Spalte, also die Position innerhalb der DataGridView-Tabelle, in der sich der Cursor gerade befindet, rufen Sie mit der Eigenschaft CurrentCell ab. Diese Eigenschaft liefert ein Objekt auf Basis der DataGridViewCell-Klasse zurück.

Um die aktuelle Zeile zu ermitteln, verwenden Sie die RowIndex-Eigenschaft dieses Objektes. Die aktuelle Spalte ermitteln Sie mit ColumnIndex.

### Feststellen, dass sich die aktuelle Zelle geändert hat

Wenn Sie informiert werden möchten, wenn sich die aktuelle Position des Cursors in der Tabelle geändert hat, binden Sie das CurrentCellChanged-Ereignis des DataGridView-Steuerelements ein. Das folgende Beispiel demonstriert, wie Sie Zeile und Spalte in einer Statuszeile anzeigen lassen können; diese Anzeige wird aktualisiert, sobald sich die Cursorposition im DataGridView-Steuerelement verändert.

**BEGLEITDATEIEN:** Sie finden diese Prozedur als Teil des Projektes im Verzeichnis .\VB 2005 - Entwicklerbuch\G - SmartClient\Kap27\Adresse03 | in der Codedatei frmSchnellerfassung.vb.

```

'Wird aufgerufen, wenn eine neue Zelle zur aktuellen Zelle wird.
Private Sub dgvAdressen_CurrentCellChanged(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles dgvAdressen.CurrentCellChanged
    With dgvAdressen
        ts1Zeile.Text = "Zeile: " & .CurrentCell.RowIndex.ToString
        ts1Spalte.Text = "Spalte: " & .Columns(.CurrentCell.ColumnIndex).HeaderText
    End With
End Sub

```

## Formatieren von Zellen

Damit Daten in einem DataGridView-Steuerelement übersichtlich aufbereitet werden können, lassen sie sich durch eine Formatzeichenfolge formatieren (mehr zum generellen Umgang mit Formatzeichenfolgen lesen Sie in ► Kapitel 17). Möchten Sie zum Beispiel für eine Spalte, die Datumswerte anzeigen, bestimmen, dass diese im Format »ddd, dd. MMM yyyy« dargestellt werden (was für den 24.2. des Jahres 2006 die Zeichenfolge »Fr, 24 Feb 2006« ergäbe), dann stellen Sie dieses Format global für alle Zellen der entsprechenden Spalte wie folgt ein:

```
dgvAdressen.Columns("GeburtsdatumDataGridViewTextBoxColumn").DefaultCellStyle.Format = "ddd, dd. MMM yyyy"
```

Wenn Ihr DataGridView-Steuerelement Zellen lediglich darstellen aber nicht bearbeiten muss, dann genügt diese Art der Formatierungsfestlegung für die entsprechende(n) Spalten(n) im Regelfall.

Wenn sich die Formatierung dabei nicht über eine Formatzeichenfolge realisieren lässt, können Sie das CellFormatting-Ereignis des DataGridView-Steuerelements behandeln, um die Formatierung dort vorzunehmen. Zwei Möglichkeiten stehen Ihnen nun zur Formatierung zur Verfügung:

- Bestimmen Sie über die Eigenschaftenkombination `CellStyle.Format` des Ereignisparameters `e` die individuell anzuwendende Formatzeichenfolge. Oder:
- Lesen Sie den zu formatierenden Wert über die `Value`-Eigenschaft des Ereignisparameters `e` aus, konvertieren Sie ihn im Bedarfsfall in den entsprechenden Datentyp, formatieren Sie ihn (in der Regel in Form einer Zeichenkette) und schreiben Sie ihn anschließend zurück in die `Value`-Eigenschaft. Setzen Sie in diesem Fall die `FormattingApplied`-Eigenschaft des Ereignisparameters `e` auf `True`, um dem DataGridView-Steuerelement anzusehen, dass Sie die Formatierung der Zelle komplett selbst übernommen haben.

---

**HINWEIS:** Denken Sie bitte daran, dass das `CellFormatting`-Ereignis für jede darzustellende Zelle ausgelöst wird. Damit Sie die richtige zu formatierende Zelle »treffen«, testen Sie mit dem Ereignisparameter `RowIndex`, ob sich das gerade ausgelöste Ereignis auf die entsprechende Spalte bezieht. Ein Beispiel dafür finden Sie im nächsten Abschnitt.

---

Wenn Ihre individuell formatierte Zelle auch die Bearbeitung von Werten zulässt, lesen Sie bitte auch den nächsten Abschnitt.

## Problemlösung für das Parsen eines Zellenwerts mit einem komplexen Anzeigeformat

Bei der Datenerfassung von anderen Typen als reine Zeichenketten stellt sich immer das Problem der Datenkonvertierung. Für die wichtigsten Datentypen implementiert das DataGridView-Steuerelement so genannte Parsing-Algorithmen, die den meisten Konvertierungsansprüchen Genüge tun.

Das Problem bei der Verarbeitung anderer Typen als reine Zeichenketten ist, dass der Datentyp für die Darstellung in eine Zeichenkette umgewandelt werden muss. Bearbeitet der Anwender eine Zelle, in der ein anderer Datentyp als eine Zeichenkette erfasst werden soll, editiert er natürlich zunächst einmal ebenfalls eine Zeichenkette, die anschließend wieder zurück in den eigentlichen Datentyp konvertiert werden muss.

Solange wie Sie ein sehr einfaches Standardformat für die Darstellung des Datentyps verwenden, treten dabei keine Probleme auf. Handelt es sich aber um eine sehr informative Aufbereitung eines Datentyps, dann wird der Standardparser des DataGridView-Steuerelements dieses Formats nicht mehr hergeben. Oder, um ein Beispiel zu bemühen:

Sie stellen in einer Spalte, in der Datumswerte angezeigt und bearbeitet werden sollen, die Werte im Format »dd.MM.yyyy« dar. Die Darstellung ist zwar sehr simpel, aber der Parser hat mit dieser Darstellung keine Probleme – er kann also einen in diesem Format editierten Wert ohne Probleme wieder zurück in den eigentlichen Datentyp umwandeln.

Entscheiden Sie sich allerdings für die Darstellung von »dd.MM.yyyy (dddd)« – der Wochentag wird hier also in Klammern neben dem eigentlichen Datum angezeigt –, und Sie lassen den Anwender die Zelle auch in diesem Format editieren, wird der Parser mit dieser komplexen Darstellung des Datums nicht mehr zurechtkommen.

Nun gibt es zwei generelle Möglichkeiten, dieses Problem zu umgehen: Sie implementieren für die entsprechende Datenspalte eine Ereignisbehandlungsroutine für das CellParsing-Ereignis und sorgen dort selbst dafür, dass der Parsing-Vorgang auch mit dem komplex formatierten Datentyp nicht fehlschlägt. Das kann aber unter Umständen eine ganze Menge Programmieraufwand bedeuten.

Oder, und das wäre meine empfohlene Vorgehensweise, sie formatieren zum Editieren die entsprechende Zelle anders als für die reine Anzeige. Und das funktioniert, weil sowohl beim Anzeigen als auch kurz vor dem Editieren das CellFormatting-Ereignis ausgelöst wird. Das folgende Beispiel zeigt, wie es funktioniert:

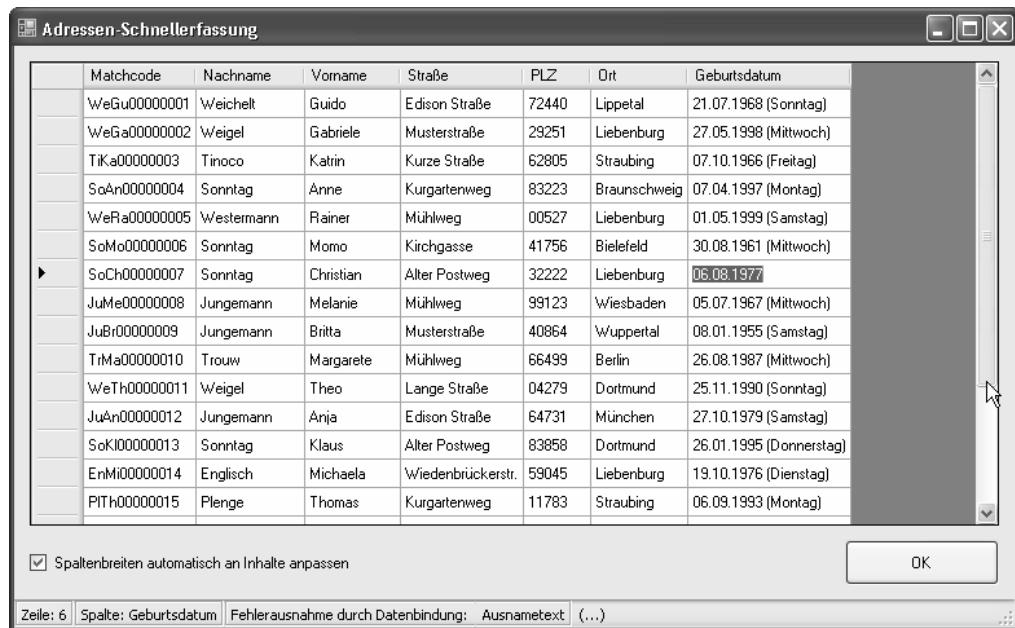
---

**BEGLEITDATEIEN:** Sie finden diese Prozedur als Teil des Projektes im Verzeichnis .\VB 2005 - Entwicklerbuch\G - SmartClient\Kap27\Adresse03 | in der Codedatei frmSchnellerfassung.vb.

---

```
'Wird für jede Zelle aufgerufen, wenn der Inhalt dieser formatiert wird.  
Private Sub dgvAdressen_CellFormatting(ByVal sender As Object,  
    ByVal e As System.Windows.Forms.DataGridViewCellFormattingEventArgs) Handles dgvAdressen.CellFormatting  
  
    'Nur die Geburtsdatums-Spalte wird besonders formatiert.  
    If e.ColumnIndex = myGebDatColumnIndex Then  
        'Die Zelle ermitteln. Aus Geschwindigkeitsgründen sollte das in einem  
        'Rutsch erfolgen; hier getrennt, damit es deutlicher wird.  
        Dim locCurrentRow As DataGridViewRow = dgvAdressen.Rows(e.RowIndex)  
        Dim locCurrentCell As DataGridViewCell = locCurrentRow.Cells(e.ColumnIndex)  
  
        'Wenn ein Geburtsdatum zum Bearbeiten formatiert wird,  
        'dann ein "parse"-bares Format darstellen.  
        If locCurrentCell.IsInEditMode Then  
            eCellStyle.Format = "dd.MM.yyyy"  
        Else  
            'Nur für die Anzeige ein "mehr informatives Format" darstellen.  
            eCellStyle.Format = "dd.MM.yyyy (dddd)"  
        End If  
    End If  
End Sub
```

Das Ergebnis dieses Aufwands können Sie betrachten, wenn Sie das Beispielprojekt starten, ein paar Zufallsadressen erzeugen und diese zum Schnellbearbeiten darstellen lassen. Wenn Sie anschließend das Geburtsdatum eines Datensatzes bearbeiten, sehen Sie, dass sich das Format zur Eingabe ändert.



**Abbildung 27.23:** Durch geschickte Behandlung des *CellFormatting*-Ereignisses wird für die Bearbeitung ein anderes Format als für die Anzeige verwendet

**HINWEIS:** Da das *CellFormatting*-Ereignis für jede einzelne darzustellende Zelle im Anzeigebereich des DataGridView-Steuerelements aufgerufen wird, sollten Sie sehr auf eine effiziente Programmierung achten und Code formulieren, der so wenig Zeit wie möglich benötigt.

## Verhindern des Editierens bestimmter Spalten aber Gestatten ihrer Neueingabe

Falls Sie das Adresso-Szenario bis hierher verfolgt haben, dann wissen Sie, dass das Beispiel das Erfassen eines Matchcodes beim Neuerfassen einer Adresse erlaubt, es aber verbietet, den Matchcode einer vorhandenen Adresse zu editieren.

Mit dem Setzen simpler Eigenschaften können Sie das DataGridView-Steuerelement nicht dazu bringen, solche Teilreglementierungen umzusetzen.

Zwar ist es möglich, beispielsweise eine Spalte mit der Anweisung

```
dgvAdressen.Columns("MatchcodeDataGridViewTextBoxColumn").ReadOnly = True
```

für die Bearbeitung ganz zu sperren. Doch das entspricht nicht dem Sinne des Erfinders, jedenfalls nicht bei unserem Beispiel.

Mit dem CellBeginEdit-Ereignis können Sie die Kontrolle über das Editieren einer Zelle in die eigenen Hände nehmen. Innerhalb dieses Ereignisses brauchen Sie nämlich nur festzustellen, ob eine neue Zeile eingegeben oder eine vorhandene editiert wird, und um welche Datenspalte es sich dabei handelt.

```
'Wird aufgerufen, bevor das Bearbeiten einer Zelle startet.  
Private Sub dgvAdressen_CellBeginEdit(ByVal sender As Object, ByVal e As _  
    System.Windows.Forms.DataGridViewCellCancelEventArgs) Handles dgvAdressen.CellBeginEdit  
  
'Verhindern, dass der Matchcode bearbeitet werden kann.  
If e.ColumnIndex = dgvAdressen.Columns("MatchcodeDataGridViewTextBoxColumn").Index Then  
  
'Neue Matchcodes können zwar bearbeitet werden, aber keine vorhandenen.  
'IsNewRow zeigt ein, ob es sich um die "Neue-DataGridView-Zeile" handelt.  
If Not dgvAdressen.CurrentRow.IsNewRow Then  
    'Wie gesagt: NICHT neue Zeile, dann Bearbeitung nicht zulassen.  
    e.Cancel = True  
End If  
End If  
End Sub
```

Schlüssel bei diesem Beispiel sind zwei Komponenten: Zum einen die Eigenschaft IsNewRow des CurrentRow-Objektes des DataGridView-Steuerelements, die Auskunft darüber gibt, ob eine vorhandene Zeile bearbeitet oder eine neue eingegeben wird. Zum anderen die Cancel-Eigenschaft des Ereignisparameters e des CellBeginEdit-Ereignisses, mit dem Sie durch Setzen auf True verhindern können, dass die Zeile editiert wird.

## Überprüfen der Richtigkeit von Eingaben auf Zellen- und Zeilenebene

Ein Blick auf Abbildung 27.12 offenbart, dass es beim DataGridView-Steuerelement so etwas wie ErrorProvider auf Zellen- und Zeilenebene gibt (falls Sie diese ErrorProvider-Klasse nicht kennen, der ► Abschnitt »Überprüfung auf richtige Benutzereingaben in Formularen« ab Seite 766 weiß mehr dazu).

Und in der Tat: Idealerweise erlauben Sie Falscheingaben zunächst zwar auf Zellenbasis, machen den Anwender aber im Moment des Verlassens einer »falsch« bearbeiteten Zelle auf diesen Missstand aufmerksam. Sie erlauben aber nicht das Verlassen einer Zeile, solange diese noch nicht korrigierte Fehler enthält.

Wenn der Anwender eine Eingabe durchgeführt hat, muss die Richtigkeit seiner Eingabe überprüft werden. Diesen Vorgang nennt man Validieren. Dementsprechend nennen sich die Ereignisse des DataGridView-Steuerelements, die ausgelöst werden, wenn Eingabeüberprüfungen auf Zellen- bzw. Zeilenebene anstehen CellValidating und RowValidating.

---

**BEGLEITDATEIEN:** Sie finden diese Prozedur als Teil des Projektes im Verzeichnis .\VB 2005 - Entwicklerbuch\G - SmartClient\Kap27\Adresse03 | in der Codedatei frmSchnellerfassung.vb.

---

Die folgenden beiden Ereignisbehandlungsroutine implementieren die Eingabereglementierungen für unsere Adresso-Anwendung auf bekannte Weise:

- Es dürfen keine doppelten Matchcodes vorhanden sein.
- Jedes Feld muss ausgefüllt werden.
- Postleitzahlen müssen mindestens 4-stellig und dürfen höchstens 5-stellig sein, und sie müssen aus Ziffern bestehen.
- Nur gültige Datumsformate dürfen eingegeben werden.

Die prinzipielle Arbeitsweise beider Routinen ist wie folgt:

- Als erstes wird das `CellValidating`-Ereignis aufgerufen, wenn der Cursor das Eingabefeld verlässt.  
**Wichtig:** Dieses Ereignis wird auch dann aufgerufen, wenn keine Eingabe oder Änderung in einem Feld erfolgte, also auch dann, wenn der Cursor nur über das Feld hinweg bewegt wurde. Die Validierung wird in der unten stehenden Ereignisbehandlungsroutine deswegen nur dann ausgeführt, wenn die Zelle mit ihrer `IsInEditMode`-Eigenschaft angezeigt, dass sie tatsächlich bearbeitet wurde (oder besser: *wird* – denn zum Zeitpunkt der Validierung läuft die Bearbeitung noch und das Beenden der Bearbeitung könnte auch mit `e.Cancel = True` verhindert werden!).
- Wurde ein Fehler in einem der Eingabefelder festgestellt, wird die Zelle durch Zuweisen eines Textes an ihre `ErrorText`-Eigenschaft als fehlerhaft markiert. Das bewirkt, dass ein kleines rotes Ausrufungszeichen an das Ende der Zelle gesetzt wird. Den Fehlertext bekommt der Anwender als Tooltip angezeigt, wenn er mit dem Mauszeiger auf das Ausrufungszeichen fährt. War das Eingabefeld okay, wird die `ErrorText`-Eigenschaft auf `Nothing` gesetzt. Ein etwaiger vorhandener Fehler-Text aus einer vorherigen Falscheingabe wird auf diese Weise zurückgesetzt. An dieser Stelle wird das Verlassen einer Zelle nach rechts oder links allerdings noch nicht verhindert.
- Versucht der Anwender eine neue Zeile »zu betreten«, tritt das `RowValidating`-Ereignis ein. Jetzt hat das Programm die Möglichkeit, die Richtigkeit der kompletten Zeile zu überprüfen. Das ist dann der Fall, wenn keine einzelne Zelle der betroffenen Zeile mehr über eine gesetzte `ErrorText`-Eigenschaft verfügt, und auf genau das überprüft der Code in dieser Ereignisbehandlungsroutine. Sollte er allerdings dabei auf noch nicht korrigierte Fehler stoßen, verhindert er einerseits das Verlassen der Zeile durch Setzen von `e.Cancel = True` und setzt ebenfalls einen Fehlerhinweis (dieses Mal auf Zeilenebene), dass es in der Zeile noch nicht korrigierte Fehler gibt.

```
'Wird aufgerufen, wenn der Inhalt einer Zelle überprüft werden soll.  
Private Sub dgvAdressen_CellValidating(ByVal sender As System.Object, ByVal e As  
    System.Windows.Forms.DataGridViewCellValidatingEventArgs) Handles dgvAdressen.CellValidating  
  
    'Aktuell zu validierende Zeile und Zelle ermitteln  
    Dim locCurrentRow As DataGridViewRow = dgvAdressen.Rows(e.RowIndex)  
    Dim locCurrentCell As DataGridViewCell = locCurrentRow.Cells(e.ColumnIndex)  
  
    'Zellen werden nur validiert, wenn sie bearbeitet wurden.  
    '(Und beim Validieren *werden* sie noch bearbeitet)  
    If locCurrentCell.IsInEditMode Then  
  
        'Postleitzahleninhalt überprüfen  
        If e.ColumnIndex = dgvAdressen.Columns("PLZDataGridViewTextBoxColumn").Index Then  
  
            'Postleitzahlen dürfen nicht mehr als 5-stellig sein,
```

```

'und nur aus Ziffern bestehen.
If (e.FormattedValue.ToString.Length < 6 And _
    e.FormattedValue.ToString.Length > 4 And _
    IsNumeric(e.FormattedValue.ToString)) Then
    locCurrentCell.ErrorText = Nothing
Else
    locCurrentCell.ErrorText = "Die Postleitzahl hat das falsche Format!"
End If

ElseIf e.ColumnIndex = dgvAdressen.Columns("GeburtsdatumDataGridViewTextBoxColumn").Index Then

    'Geburtsdatumsformat überprüfen
    Dim locDate As Date
    If Not Date.TryParse(e.FormattedValue.ToString, locDate) Then
        locCurrentCell.ErrorText = "Das eingegebene Datum hat das falsche Format!"
    Else
        'Zurücksetzen
        locCurrentCell.ErrorText = Nothing
    End If
Else
    'Alle anderen Felder nur auf nicht vorhandene Eingabe überprüfen
    If String.IsNullOrEmpty(e.FormattedValue.ToString) Then
        locCurrentCell.ErrorText = "Fehlende Eingabe!"
    Else
        locCurrentCell.ErrorText = Nothing
    End If
End If
End Sub

'Wird aufgerufen, wenn beim Wechseln der Zeile
'die Richtigkeit der Inhalte *aller* Zellen überprüft werden sollen.
Private Sub dgvAdressen_RowValidating(ByVal sender As Object, ByVal e As _
    System.Windows.Forms.DataGridViewCellCancelEventArgs) Handles dgvAdressen.RowValidating

    'Alle Zellen der betroffenen Zeile durchlaufen
    For Each locCell As DataGridViewCell In dgvAdressen.Rows(e.RowIndex).Cells

        'Schauen, ob noch irgendwo ein Fehlertext vermerkt ist
        If Not String.IsNullOrEmpty(locCell.ErrorText) Then
            'Falls ja, dann auch einen Fehler auf Zeilenbasis eintragen
            dgvAdressen.Rows(e.RowIndex).ErrorText = "Diese Zeile hat noch nicht korrigierte Fehler!"
            e.Cancel = True
            Return
        End If
    Next

    'Keine der Zellen wies einen Fehler auf --> Zeilenfehler zurücksetzen.
    dgvAdressen.Rows(e.RowIndex).ErrorText = Nothing
End Sub

```

## Ändern des Standardpositionierungsverhaltens des Zellencursors nach dem Editieren einer Zelle

Das DataGridView-Steuerelement hat ein etwas seltsames Verhalten, das bei Anwendern, die viele Eingaben mit seiner Hilfe vornehmen müssen, auf wenig Gegenliebe stößt. Nachdem Sie eine Zelle im Steuerelement geändert haben, springt der Cursor nicht in die rechts daneben stehende Zelle sondern springt eine Zelle nach unten.

Da es Anwender nun einmal gewohnt sind, eine Eingabe auch mit der Taste **Eingabe** und nicht mit der Cursor-nach-rechts-Taste zu beenden, macht sich hier schnell Unmut breit.

Dummerweise verfügt das DataGridView-Steuerelement zwar über die meisten Einstellmöglichkeiten aller Steuerelemente überhaupt, aber über keine, um dieses Verhalten direkt zu ändern.

Die Klassenvererbung ist die einzige Möglichkeit, diesem Problem zu entgehen. Das bedeutet: Sie vererben das DataGridView-Steuerelement, das im Prinzip ja nichts anderes ist als eine Klasse, in eine neue Klasse und kommen auf diese Weise an bestimmte Methoden und Eigenschaften heran, die Ihnen bei der direkten Verwendung eines DataGridView-Objektes nicht zugänglich sind. Und dabei handelt es sich um genau die Elemente, die als Protected so gekennzeichnet sind, dass auf sie nur von der Klasse selbst und von jeder geerbten Klasse darauf zugegriffen werden kann.

Wenn Sie ein Steuerelement in Visual Basic 2005 vererben, erstellen Sie automatisch ein neues Steuerelement. Damit hat dieser Abschnitt in diesem Kapitel eigentlich gar nichts verloren, denn erst ► Kapitel 30 beschäftigt sich mit diesem Thema detaillierter. Mir schien es aber sinnvoll, gerade dieses herbe Manko des DataGridView-Steuerelements im Kontext des Themas *DataGridView* zu erklären. Wenn Sie genauer interessiert, wie Sie eigene Steuerelemente mit Visual Basic 2005 für die Vereinfachung von Programmierungen Ihrer eigenen Anwendungen erstellen, lesen Sie eben Kapitel 30. Die Erklärung des Programms an dieser Stelle erfolgt deswegen auch nur in aller Kürze.

---

**BEGLEITDATEIEN:** Sie finden das folgende Beispielprojekt im Verzeichnis `.\VB 2005 - Entwicklerbuch\G - Smart-Client\Kap27\Adresse04`.

---

Zentrale Änderung zum vorherigen Beispielprojekt ist eine neue Klassendatei namens *DataGridViewEx.vb*, die die Erweiterung des DataGridView-Steuerelements in Form eines neuen Steuerelements namens *DataGridViewEx* implementiert.

Um dieses erweiterte Steuerelement mit der »richtig« funktionierenden **Eingabe**-Taste in Ihren zukünftigen Projekten zu verwenden, kopieren Sie diese Klassencode datei einfach in Ihr Projektverzeichnis, lassen anschließend mit Klick auf das entsprechende Symbol im Projektmappenexplorer alle Dateien anzeigen, wählen die nun sichtbar gewordene Klassendatei aus und rufen aus dem Kontextmenü des Projektmappenexplorers den Befehl *Zum Projekt hinzufügen* auf. Nach dem ersten Kompilieren des Projekts erscheint das neue Steuerelement automatisch in der Toolbox, und Sie können es wie eine normale DataGridView verwenden.

Möchten Sie, dass dieses Steuerelement anstelle bestehender DataGridView-Steuerelemente verwendet wird, dann schließen Sie zunächst ein möglicherweise im Designer geöffnetes Formular, das die »betroffene« DataGridView beinhaltet, und öffnen Sie den Designer-Code dieses Formulars. Den Designer-Code erreichen Sie, indem Sie im Projektmappen-Explorer mit dem entsprechenden Symbol *alle Dateien anzeigen* lassen und anschließend den Zweig vor dem Formular erweitern, das das DataGridView-Steuerelement erhält. Öffnen Sie nun den Code der unter der eigentlichen Formular-

klasse stehenden Codedatei mit der Endung *.designer.vb*. Suchen Sie hier alle System.Windows.Forms.DataGridView-Referenzen (per DataGridView dürften es nicht mehr als 2 sein) und ersetzen Sie sie durch DataGridViewEx. Kompilieren Sie das Projekt anschließend neu.

Den dokumentierten Code des erweiterten Steuerelements finden Sie im Folgenden:

```
Public Class DataGridViewEx
    Inherits DataGridView

    'Verarbeitet Tasten, z. B. TAB, ESC, die EINGABETASTE und Pfeiltasten,
    'die zum Steuern von Dialogfeldern verwendet werden.
    Protected Overrides Function ProcessDialogKey(ByVal keyData As Keys) As Boolean

        'Nur die Keycodes interessieren, Rest wird ausmaskiert.
        Dim key As Keys = (keyData And Keys.KeyCode)

        'Eingabe-Taste gedrückt?
        If key = Keys.Enter Then

            'Ja, dann in Cursor-rechts umleiten, wo
            'Eingabe-Taste nochmal gesondert behandelt wird.
            Return MyBase.ProcessRightKey(keyData)
        End If

        'Andere Keyes bleiben nicht betroffen
        Return MyBase.ProcessDialogKey(keyData)
    End Function

    'Die alte Cursor-Rechts-Funktion wird überschattet, damit die
    'Polymorphie dieser Funktion an dieser Stelle unterbrochen wird.
    Public Shadows Function ProcessRightKey(ByVal keyData As Keys) As Boolean

        'Nur die Keycodes interessieren, Rest wird ausmaskiert.
        Dim key As Keys = (keyData And Keys.KeyCode)

        'Wenn es sich um die umgeleitete Eingabe-Taste handelte...
        If key = Keys.Enter Then

            'Feststellen, ob sich der Cursor am Ende der letzten Spalte befand, ...
            If MyBase.CurrentCell.ColumnIndex = (MyBase.ColumnCount - 1) Then
                '...dann den Cursor nach vorn und eine Zeile nach unten verschieben.
                MyBase.CurrentCell = MyBase.Rows(MyBase.CurrentCell.RowIndex + 1).Cells(0)
                Return True
            End If
        End If
        'Bei allen anderen Position reicht es aus,
        'das Standardverhalten von Cursor-rechts statt Eingabe zu verwenden.
        Return MyBase.ProcessRightKey(keyData)
    End Function
```

```

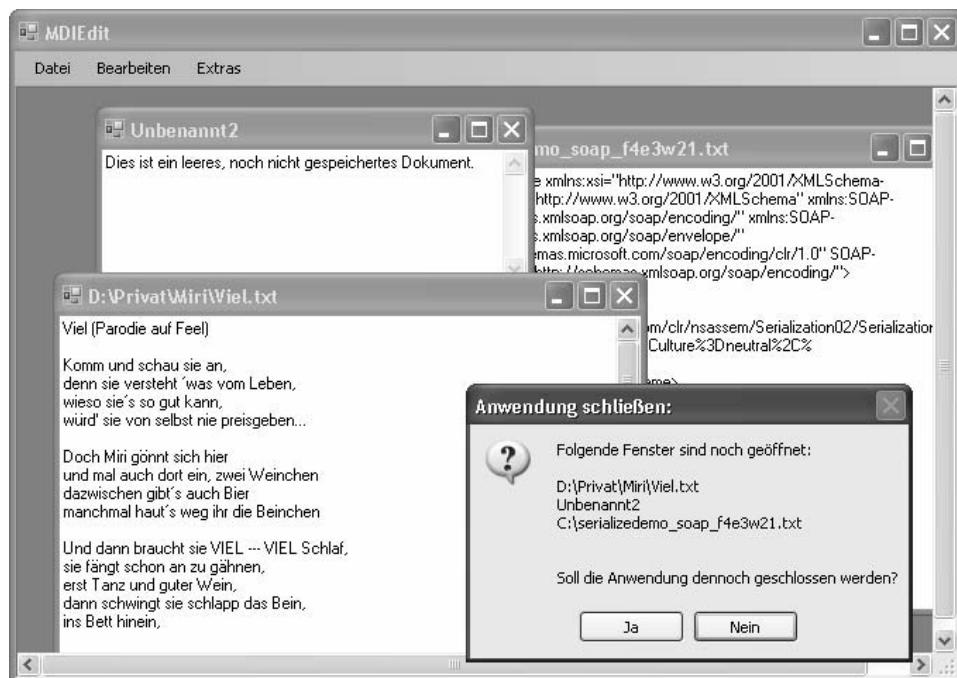
'Verarbeitet Tasten, die zum Navigieren in der DataGridView verwendet werden.
Protected Overrides Function ProcessDataGridViewKey(ByVal e As KeyEventArgs) As Boolean
    If e.KeyCode = Keys.Enter Then
        Return Me.ProcessRightKey(e.KeyData)
    End If
    Return MyBase.ProcessDataGridViewKey(e)
End Function
End Class

```

## Entwickeln von MDI-Anwendungen

MDI ist die Abkürzung für *Multi Document Interface*, und das bedeutet etwas freier übersetzt: Benutzerschnittstelle für mehrere Dokumente. Was dieser Ausdruck meint: MDI-Anwendungen verfügen über ein Hauptformular (oder, um meinen Fachlektor zufrieden zu stellen, über ein Hauptfenster), das die generische Grundfunktionalität der gesamten Anwendung anbietet. Es bindet aber gleichzeitig für jedes eigentliche Dokument, das Sie mit ihm bearbeiten, mehrere Unterfenster (die aus dem Englischen übernommene Bezeichnung müsste eigentlich Kindfenster – von Child Window – heißen, hat sich aber nicht durchgesetzt) ein, die jeweils das eigentliche Dokument darstellen.

Was dabei genau ein »Dokument« darstellt, bestimmt die Anwendung. Das können Bitmaps, Zeichnungen, Kalkulationstabellen oder auch Textdokumente sein, etwa wie in der folgenden Abbildung zu sehen:



**Abbildung 27.24:** Eine typische MDI-Anwendung – hier zum Bearbeiten von Textdokumenten

Das eigentliche Darstellen eines untergeordneten Fensters in einem übergeordneten (eigentlich: *Elternfenster*, da von *Parent Window* abgeleitet), ist noch das kleinere Problem bei der Entwicklung bzw. Konzeption von MDI-Anwendungen. Mit insgesamt zwei Eigenschaften der Formularklasse sorgen Sie nämlich dafür, ein Formular einerseits zum Container für untergeordnete Fenster zu machen (*FormInstanz.IsMdiContainer = True*), andererseits dafür, dass eine neue Instanz eines Formulars zum untergeordneten Formular eines MDI-Hauptformulars wird (*UntergeordneteForm.MdiParent = Elternform*). Ein Aufruf der *Show*-Methode des untergeordneten Formulars genügt anschließend, um das Formular als Kindfenster einer MDI-Anwendung darzustellen.

Größere Probleme bilden bei der Entwicklung von MDI-Anwendungen die folgenden Punkte:

- Welche Funktionen werden von der Hauptanwendung, welche von den untergeordneten Formularklassen idealerweise übernommen?
- Wie lassen sich die Funktionsaufrufmöglichkeiten der beiden Instanzen visuell vereinen?

Lassen Sie uns beim Beispiel aus Abbildung 27.24 bleiben, um diese Problemstellungen zu erläutern.

Beim Entwickeln einer MDI-Texteditor-Anwendung wird recht schnell klar, dass globale Funktionen wie das Öffnen einer Datei, das Zur-Verfügung-Stellen eines globalen Optionsdialogs oder das Beenden des Programms in der Klasse des Hauptformulars implementiert werden müssen. Alle weiteren Funktionen wie das Speichern einer Datei, das Drucken, Suchen und Ersetzen im Text müssen von der jeweiligen Dokumentenklasse bereitgestellt werden, denn diese zuletzt genannten Funktionen beziehen sich immer auf die verschiedenen Dokumentinstanzen.

---

**BEGLEITDATEIEN:** Sie finden das Beispielprojekt für diesen Abschnitt im Verzeichnis `.\VB 2005 - Entwicklerbuch\G - SmartClient\Kap27\MDIEdit`.

---

Um das Beispiel einfach zu halten, beschränken wir uns auf folgende Funktionalität.

Für das globale Hauptformular:

- Öffnen einer Datei
- Erstellen einer neuen Datei
- Beenden des Programms (und Schließen aller noch geöffneten Dokumente mit Sicherheitsabfrage und Abbruchmöglichkeit)
- Aufruf eines Options-Dialogs.

Für jede Dokumentenklasse:

- Speichern der Datei
- Aufruf einer Suchfunktion.

In dieser Beispielanwendung gibt es nur eine einzige Dokumentenklasse. Denkbar wären aber auch zwei Dokumentenklassen mit unterschiedlicher Funktionalität – eine beispielsweise für das Bearbeiten reiner Textdateien, eine andere für das Bearbeiten von Richt-Text-Dokumenten, also solchen, die mit Formatierungen ausgestattet werden können.

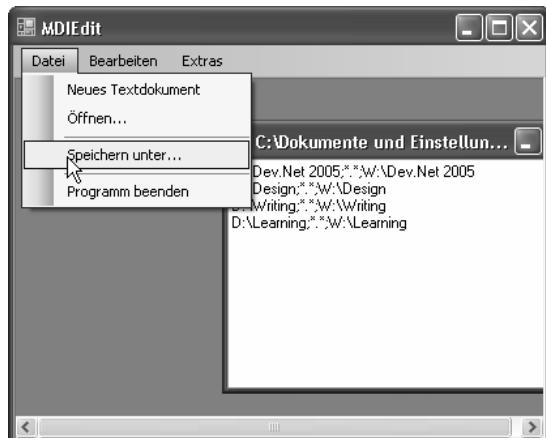
Und hier ergibt sich nun das nächste Problem, nämlich das der unterschiedlichen Benutzeroberfläche. Eine Dokumentenklasse, die Richt-Text-Dokumente bearbeitet, muss beispielsweise in der Hauptanwendung ein *Format*-Menü bereitstellen; eine einfache Textdokumentklasse benötigt dieses nicht.

Elegant wäre es, wenn jede Dokumentenklasse ihre eigene Benutzeroberfläche in Form von Menü und Toolbar mitbrächte, die dann mit der Hauptanwendung verschmelzen würde, sobald man eine ihrer Instanz aktivierte.

Das Ergebnis sähe dann so aus, wie in den folgenden beiden Abbildungen zu sehen:



**Abbildung 27.25:** Solange es keine geöffneten Dokumente gibt (und damit keine instanziierten Dokumentklassen), steht nur der Grundpool an Funktionen zur Verfügung, der über die Menüstruktur zu erreichen ist. Dieser wird ...



**Abbildung 27.26:** ... aber durch die jeweils aktive Dokumentklasseninstanz erweitert – die beiden Funktionssätze verschmelzen sozusagen.

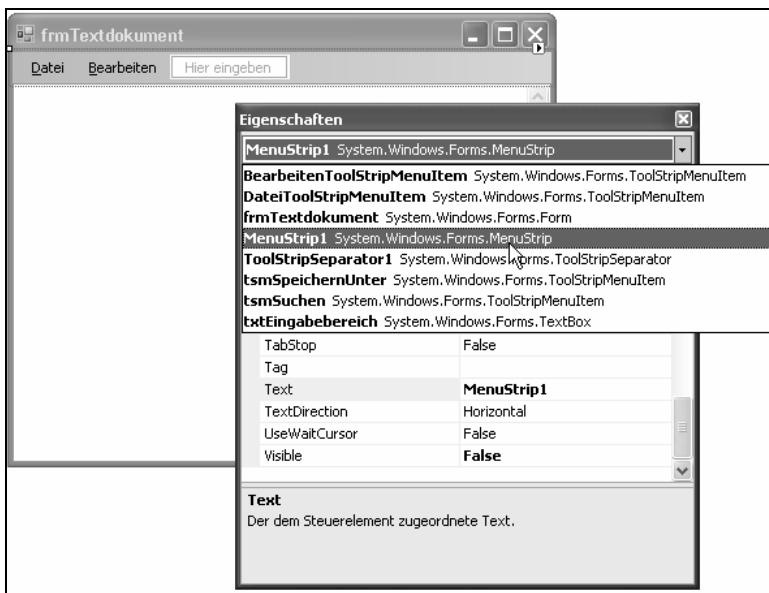
Um diese Technik zu erreichen, bieten sowohl das `MenuStrip`-Steuerelement (mit dem Sie eine Menüstruktur im Fenster darstellen lassen können) als auch das `ToolStrip`-Steuerelement (mit dem Sie Toolbars anzeigen lassen können) besondere Eigenschaften an: Menüs (und auch Toolbars) von untergeordneten Formularen können dynamisch und nur durch Aktivierung zur Laufzeit mit denen von übergeordneten Formularen zusammengeführt werden.

Wichtig zu wissen sind dabei die folgenden Punkte:

- Sie erstellen ein Menü oder eine Toolbar für das Hauptfenster und müssen nur darauf achten, dass die standardmäßig gesetzte `AllowMerge`-Eigenschaft des jeweiligen Steuerelements (`MenuStrip`, `ToolStrip`) auch wirklich gesetzt ist.

- Sie erstellen ein Menü oder eine Toolbar für jedes untergeordnete Fenster und »tun« zunächst so, als wäre das Menü oder die Toolbar ausschließlich für dieses Fenster gedacht.
- Für untergeordnete Fenster bestimmen Sie anschließend über die MergeAction und MergeIndex-Eigenschaften, auf welche Weise und an welcher Stelle im übergeordneten Fenster die Elemente vereinigt werden sollen. Bei Elementen mit Unterelementen ergibt sich dabei eine besonders knifelige Problematik, wenn Elemente untergeordneter Ebenen zusammengeführt werden sollen. In diesem Fall müssen Sie die MergeIndex-Eigenschaft der Elemente der übergeordneten Ebene auf die gleiche Indexnummer des entsprechenden Elements im übergeordneten Fenster setzen, und als MergeAction den Wert MergeOnly einstellen. Möchten Sie aus dem untergeordneten Formular bestimmte Elemente übernehmen, legen Sie für diese als MergeIndex die Positionen fest, die der Position im Hauptfenster entsprechen, und für MergeAction bestimmen Sie den Wert Insert.
- Wenn das untergeordnete Formular zur Laufzeit seine sämtlichen Elemente (Menüs, Toolbars) mit dem übergeordneten verschmelzen lassen soll, dann legen Sie schon zur Entwurfszeit die Visible-Eigenschaft des Steuerelements auf False fest.

**ACHTUNG:** Aber aufgepasst dabei: Wenn Sie die Visible-Eigenschaft eines ToolStrip- oder eines MenuStrip-Steuerelements zur Entwurfszeit auf False setzen, dann verschwindet es auch im Designer. Das ist ungewöhnlich, aber logisch: Denn Ihnen muss beim Gestalten des Formulars ja schließlich der Platz für das unsichtbare Steuerelement »gutgeschrieben« werden. Das Problem: Wie verändern Sie die Elemente eines unsichtbaren Steuerelements, denn Sie können es ja nicht mehr anklicken!? Abbildung 27.27 hat die Lösung: Sie können es für den Augenblick der Bearbeitung wieder sichtbar machen, indem Sie es mithilfe des Eigenschaftenfensters (obere Aufklappliste) selektieren. Sobald das Steuerelement anschließend seinen Fokus wieder verloren, weil Sie ein anderes angeklickt haben, ist es auch schon wieder verschwunden.



**Abbildung 27.27:** Ein ToolStrip- oder MenuStrip-Steuerelement, das Sie mit seiner Visible-Eigenschaft unsichtbar machen mussten, erreichen Sie zum Selektieren über das Eigenschaftenfenster

- Die Einstellungen von MergeIndex und MergeAction für das jeweilige Steuerelement im übergeordneten Formular bleiben völlig unberührt.

---

**HINWEIS:** Viele Entwickler sind der Meinung, dass korrelierende Indexnummern der MergeIndex-Eigenschaften der Steuerelemente im untergeordneten und übergeordneten Fenster die Korrelation bestimmen. Das ist falsch. Die MergeIndex-Nummer des Steuerelements des untergeordneten Formulars bezieht sich immer nur auf die Nummer schon *vorhandener* Elemente von Steuerelementen des übergeordneten Formulars, und nur auf *gleicher Ebene!*

---

Und das bedeutet: Wenn das Menü des untergeordneten Steuerelements die Einträge:

- *Datei*
- *Datei / Trennlinie*
- *Datei / Speichern*
- *Bearbeiten*
- *Bearbeiten / Suche*

aufweist, und das übergeordnete Formular bereits die Menüeinträge

- *Datei (0)*
- *Datei / Neues Dokument (0) (0)*
- *Datei / Dokument öffnen (0) (1)*
- *Datei / Trennlinie (0) (2)*
- *Datei / Programm beenden (0) (3)*
- *Extra (1)*
- *Extras / Optionen (1) (0)* enthält,

so ist die MergeAction-Eigenschaft für *Datei* des untergeordneten Steuerelements auf MergeOnly und die MergeIndex-Eigenschaft auf 0 zu setzen. Der erste Menüeintrag auf dieser Ebene (*Datei*) im untergeordneten Formular soll nämlich nur mit dem ersten Menüeintrag gleicher Ebene im übergeordneten Formular verschmolzen werden.

Für *Datei/Trennlinie* und *Datei/Speichern* des untergeordneten Formulars setzen Sie die MergeAction-Eigenschaft auf Insert, weil diese Menüpunkte im Menü des übergeordneten Formulars eingefügt werden sollen, und zwar hinter *Datei/Dokument öffnen* und vor *Datei/Trennlinie*. Da bei Insert immer *vor* einem Element eingefügt wird, und oben angegebene Nummerierung im übergeordneten Formular gilt, bestimmen Sie für *Datei/Trennlinie* den Wert 2 als MergeIndex für *Datei speichern*.

Für *Bearbeiten* bestimmen Sie aus gleichen Gründen 1 für MergeIndex und Insert für MergeAction, weil das komplette *Bearbeiten*-Menü dann vor dem *Extras*-Menü eingefügt wird.

Und damit ist die Einrichtung der Benutzeroberfläche auch schon vollbracht! Sie können sich nun um die programmtechnische Umsetzung der eigentlichen Funktionen kümmern. Die ist in unserem Beispiel vergleichsweise simpel, wie die beiden folgenden Listings des Hauptformulars und das des Formulars zeigen, das die Dokumentenklasse in der Beispielanwendung ausmacht.

## **Listing von frmMain.vb (übergeordnetes Hauptformular)**

```
Public Class frmMain

    'Der "Unbenannt"-Dateinamenzähler.
    Private myUnbekanntesDokumentZähler As Integer

    Protected Overrides Sub OnLoad(ByVal e As System.EventArgs)
        MyBase.OnLoad(e)

        'Den "Unbenannt"-Dateinamenzähler beim Laden des Formulars initialisieren.
        myUnbekanntesDokumentZähler = 1
    End Sub

    'Wird aufgerufen, wenn der Anwender aus dem Menü Datei den Menüpunkt Neues Dokument auswählt.
    Private Sub NeuesDokument_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles _
        tsmNeuesDokument.Click
        'Neue Instanz des MDI-Dokuments erstellen.
        Dim locFrmTextdokument As New frmTextdokument

        'Dieser Instanz mitteilen, dass es ein MDI-untergeordnetes Fenster werden soll.
        locFrmTextdokument.MdiParent = Me

        'Die Funktion zum Anzeigen des neuen untergeordneten Fensters aufrufen.
        locFrmTextdokument.NeuesTextdokument(myUnbekanntesDokumentZähler)

        'Den "Unbenannt"-Dateinamenzähler um eins erhöhen.
        myUnbekanntesDokumentZähler += 1
    End Sub

    'Wird aufgerufen, wenn der Anwender aus dem Menü Datei den Menüpunkt Öffnen auswählt.
    Private Sub tsmÖffnen_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles _
        tsmÖffnen.Click

        Dim dateiÖffnenDialog As New OpenFileDialog
        With dateiÖffnenDialog
            .CheckFileExists = True
            .CheckPathExists = True
            .DefaultExt = "*.txt"
            .Filter = "Textdateien" & " (" & "*.txt" & ")|" & "*.txt" & "|Alle Dateien (*.*)|*.*"

            'Dateinamen der zu öffnenden Textdatei ermitteln.
            Dim dialogErgebnis As DialogResult = .ShowDialog
            If dialogErgebnis = Windows.Forms.DialogResult.Cancel Then
                Exit Sub
            End If

            'Neue Instanz des MDI-Dokuments erstellen.
            Dim locFrmTextdokument As New frmTextdokument

            'Dieser Instanz mitteilen, dass es ein MDI-untergeordnetes Fenster werden soll.
            locFrmTextdokument.MdiParent = Me
        End With
    End Sub

```

```

'Die Funktion zum Laden des Textes und Anzeigen des untergeordneten Fensters aufrufen.
locFrmTextdokument.TextdokumentÖffnen(.FileName)
End With
End Sub

'Wird aufgerufen, wenn der Anwender aus dem Menü Extras den Menüpunkt Optionen auswählt.
Private Sub tsmOptionen_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles tsmOptionen.Click
    MessageBox.Show("Hier für die Optionen-Funktion implementiert werden")
End Sub

'Wird aufgerufen, wenn der Anwender aus dem Menü Datei den Menüpunkt Programm beenden auswählt.
Private Sub tsmProgrammBeenden_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles tsmProgrammBeenden.Click
    Me.Close()
End Sub

'Wird beim Schließen des Formulars aufgerufen
Protected Overrides Sub OnFormClosing(ByVal e As System.Windows.Forms.FormClosingEventArgs)
    MyBase.OnFormClosing(e)

    'Gibt es untergeordnete Fenster?
    If Me.MdiChildren IsNot Nothing AndAlso Me.MdiChildren.Length > 0 Then
        Dim locFenstertitel As String = ""

        'Die Fenstertitel der untergeordneten Fenster ermitteln
        For Each locForm As Form In Me.MdiChildren
            locFenstertitel &= locForm.Text & vbCrLf
        Next

        'Warnhinweis mit der Liste der noch offenen Fenster.
        Dim locDr As DialogResult = MessageBox.Show("Folgende Fenster sind noch geöffnet:" & _
            vbCrLf & vbCrLf & locFenstertitel & _
            vbCrLf & vbCrLf & _
            "Soll die Anwendung dennoch geschlossen werden?", _
            "Anwendung schließen:", MessageBoxButtons.YesNo, _
            MessageBoxIcon.Question, MessageBoxDefaultButton.Button2)
        If locDr = Windows.Forms.DialogResult.No Then
            'Der Anwender hat Nein gesagt!
            'Das Fenster bleibt offen.
            e.Cancel = True
        End If
    End If
End Sub
End Class

```

### Listung von frmTextdokument.vb (untergeordnete Dokumentenklasse)

```
Public Class frmTextdokument
    ''' <summary>
    ''' Zeigt dieses Formular mit einem leeren Textfeld an
    ''' </summary>
    ''' <remarks></remarks>
    Public Sub NeuesTextdokument(ByVal DokumentNr As Integer)
        txtEingabebereich.Text = Nothing

        'Dokumentnamen "Unbenannt_X" im Titel anzeigen.
        Me.Text = "Unbenannt" & DokumentNr

        'Formular nicht-modal anzeigen.
        Me.Show()
    End Sub

    ''' <summary>
    ''' Zeigt dieses Formular an, und lädt eine Textdatei in den Eingabebereich.
    ''' </summary>
    ''' <param name="Dateiname"></param>
    ''' <remarks></remarks>
    Public Sub TextdokumentÖffnen(ByVal Dateiname As String)

        'Textdatei in den Eingabebereich laden.
        txtEingabebereich.Text = My.Computer.FileSystem.ReadAllText(Dateiname)

        'Dateinamen als Dokumentnamen im Titel anzeigen.
        Me.Text = Dateiname

        'Formular nicht-modal anzeigen.
        Me.Show()
    End Sub

    'Wird aufgerufen, wenn der Anwender aus dem Menü Datei den Menüpunkt Speichern unter auswählt.
    Private Sub tsmSpeichernUnter_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
        Handles tsmSpeichernUnter.Click
        Dim dateiSpeichernDialog As New SaveFileDialog
        With dateiSpeichernDialog

            'Pfad muss vorhanden sein!
            .CheckPathExists = True

            'Warnen, wenn die Datei schon existiert!
            .OverwritePrompt = True

            'Dateinamenerweiterungen einrichten:
            .DefaultExt = "*.txt"
            .Filter = "Textdateien" & " (" & "*.txt" & ")|" & "*.txt" & "|Alle Dateien (*.*)|*.*"
            Dim dialogErgebnis As DialogResult = .ShowDialog
            If dialogErgebnis = Windows.Forms.DialogResult.Cancel Then
                Exit Sub
            End If
        End With
    End Sub
```

```

'Textdatei speichern
My.Computer.FileSystem.WriteAllText(.FileName, txtEingabebereich.Text, False)

'Fenstertitel ist Dateiname
Me.Text = .FileName
End With
End Sub

'Wird aufgerufen, wenn der Anwender im Menü Bearbeiten auf Suchen klickt.
Private Sub tsmSuchen_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
Handles tsmSuchen.Click
    MessageBox.Show("Hier würde die Suchenfunktion implementiert werden!")
End Sub
End Class

```

## Vererben von Formularen

Da Formulare nichts anderes sind als Klassen, können Sie sie natürlich auch vererben. Im Prinzip machen Sie das automatisch, sobald Sie eine neue Windows Forms-Anwendung anlegen: Das Formular, das Ihrer Anwendung automatisch hinzugefügt wird, erbt aus der Basisklasse *Form*.

Allerdings gibt es eine besondere Möglichkeit, Formulare zu vererben; in Visual Studio .NET spricht man dabei von der visuellen Vererbung von Formularen. Die IDE bezeichnet solche Formulare schlicht als *geerbte Formulare*.

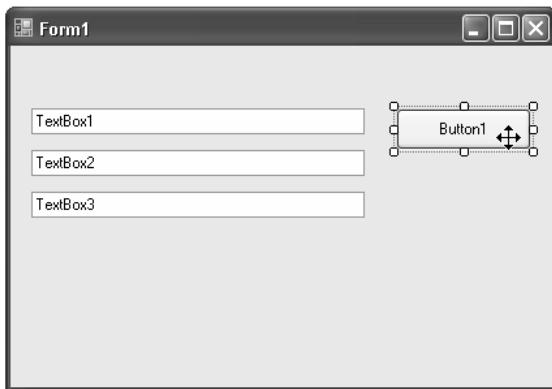
---

**BEGLEITDATEIEN:** Wenn Sie, anstatt die folgenden Schritte selbst durchzuführen, lieber auf das fertige Projekt zurückgreifen wollen: Sie finden Sie es unter *.VB 2005 - Entwicklerbuch\G - SmartClient\Kap27\FormVererbung*.

---

Die Vorgehensweise dazu ist denkbar einfach:

- Erstellen Sie ein neues *Windows Forms*-Projekt, beispielsweise unter dem Namen *FormVererbung*.
- Platzieren Sie ein paar *TextBox*-Steuerelemente auf dem Formular, etwa wie in Abbildung 27.28 zu sehen.



**Abbildung 27.28:** Dieses Formular dient als Basis für die visuelle Vererbung

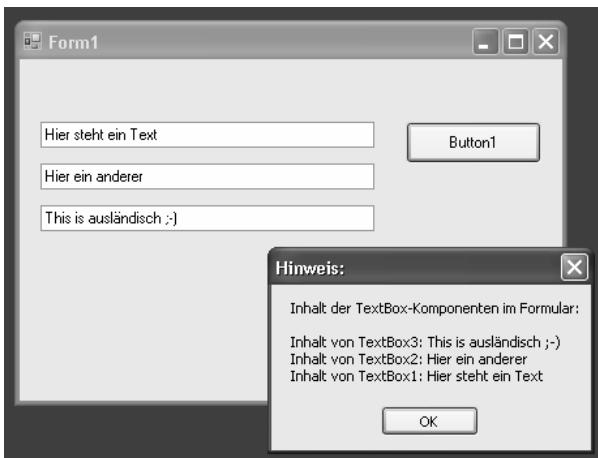
- Doppelklicken Sie auf die Schaltfläche *OK*, um den Codeeditor zu öffnen, und fügen Sie den folgenden Programmcode ein:

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

    Dim locString As String

    For Each locControl As Control In Me.Controls
        If TypeOf locControl Is TextBox Then
            locString += "Inhalt von " + locControl.Name + ": "
            locString += DirectCast(locControl, TextBox).Text + vbCrLf
        End If
    Next
    locString = "Inhalt der TextBox-Komponenten im Formular:" + _
        vbCrLf + vbCrLf + locString
    MessageBox.Show(locString, "Hinweis:")
End Sub
```

- Starten Sie das Programm anschließend, geben Sie ein paar Zeichen in das TextBox-Steuerelement ein, und beobachten Sie, welche Aktion ein Mausklick auf *OK* anschließend auslöst (siehe Abbildung 27.29).



**Abbildung 27.29:** Formulare kommen übrigens auch ohne Steuerelemente-Arrays aus. Das hat zwar nichts mit dem Thema dieses Abschnitts zu tun, ergibt sich aber durch das Demo für die visuelle Vererbung so ganz nebenbei.

## ControlCollection vs. Steuerelemente-Array aus VB6

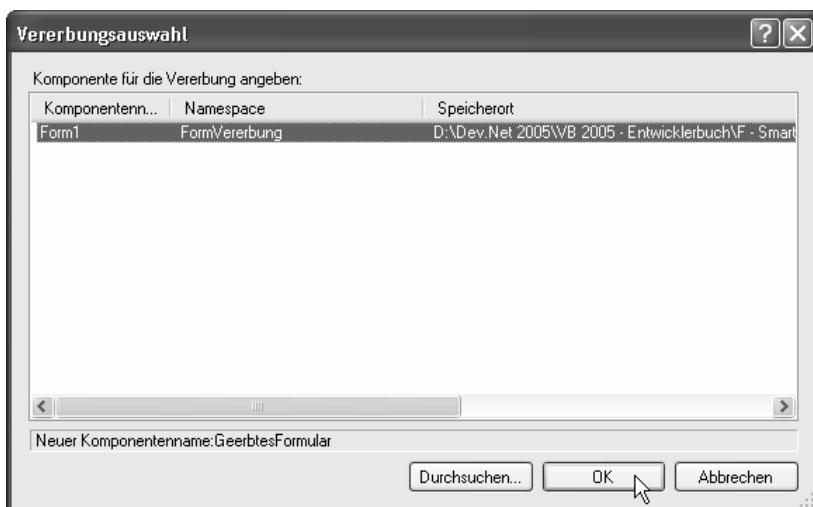
Auch wenn es nichts mit dem eigentlichen Thema dieses Abschnittes zu tun hat, so zeigt dieses Beispielprogramm dennoch auf einfache Weise, wie unnötig Steuerelemente-Arrays in .NET sind und wieso es sie aus diesem Grund auch nicht mehr gibt. Denn auch die *Controls*-Auflistung, die vordergründig dazu dient, das Formular mit Steuerelementen auszustatten, lässt sich wie jede andere Auflistung, die über einen Enumerator verfügt, mit *For/Each* durchlaufen. Anhand des Typen, den Sie wie im Beispiel mit *Type Of* ermitteln können, haben Sie die Möglichkeit, durch eine entsprechende Typkonvertierung auf die gewünschte Eigenschaft des Steuerelements zuzugreifen. ►

Kleiner Tipp am Rande: Falls Sie weitere, spezifische Informationen in einem Steuerelement abspeichern möchten, die nur der Identifizierung dienen, verwenden Sie die Tag<sup>6</sup>-Eigenschaft, die jedes auf Control basierende Steuerelement zur Verfügung stellt. Anders als in Visual Basic 6.0 können Sie in der Tag-Eigenschaft nicht nur Strings, sondern beliebige Objekte speichern.

Für die eigentliche visuelle Vererbung dieses Formulars beenden Sie bitte das Programm, indem Sie auf die Schließschaltfläche des Formulars klicken. Um ein geerbtes Formular dem Projekt hinzuzufügen, verfahren Sie folgendermaßen:

- Fahren Sie mit der Maus in den Projektmappen-Explorer, klicken Sie mit der rechten Maustaste über dem Projektnamen, und wählen Sie aus dem Kontextmenü, das sich jetzt öffnet, den Eintrag *Hinzufügen* und *geerbtes Formular hinzufügen*.
- Geben Sie im Dialog, der jetzt erscheint, *GeerbtesFormular* als neuen Formularnamen ein.
- Klicken Sie anschließend auf *Öffnen*.

Visual Studio zeigt Ihnen anschließend einen Dialog, wie Sie ihn auch in Abbildung 27.30 sehen können. In diesem Dialog werden alle Formulare der Projektmappe angezeigt, die sie visuell vererben können. Wählen Sie für dieses Beispiel den einzigen vorhandenen Eintrag aus, und klicken Sie abschließend auf OK.

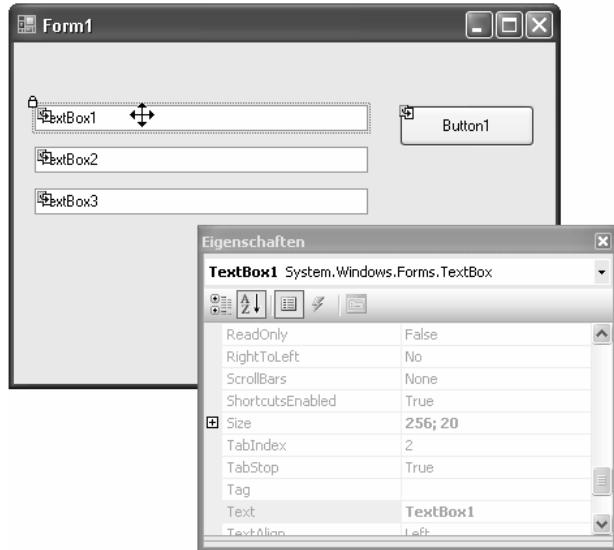


**Abbildung 27.30:** In dieser Liste, die alle Formulare der Projektmappe zeigt, wählen Sie das Formular, aus dem das neue Formular abgeleitet werden soll

Wenn Sie diesen Vorgang ausgeführt haben, gibt es anschließend einen zweiten Dialog in Ihrer Projektmappe mit dem Namen *GeerbtesFormular.vb*. Klicken Sie ihn doppelt an, um ihn darzustellen und sich davon zu überzeugen, dass er seine äußerliche Ähnlichkeit vom Vater-Dialog geerbt hat (siehe Abbildung 27.31).

<sup>6</sup> Engl. Tag (ausgesprochen: »Tähg«), auf deutsch etwa: *Markierung, Kennzeichnung*.

Wenn Sie Ihre ersten Experimente mit dem vererbten Steuerelement unternehmen, wird Ihnen eines gleich auffallen: Zwar sind alle Steuerelemente im Formular vorhanden, doch lassen sie sich nicht verändern. Weder können Sie ihre Eigenschaften editieren, noch lassen sie sich umpositionieren oder in ihrer Größe verändern (was letzten Endes auch nichts anderes ist als ein Verändern der Eigenschaften).

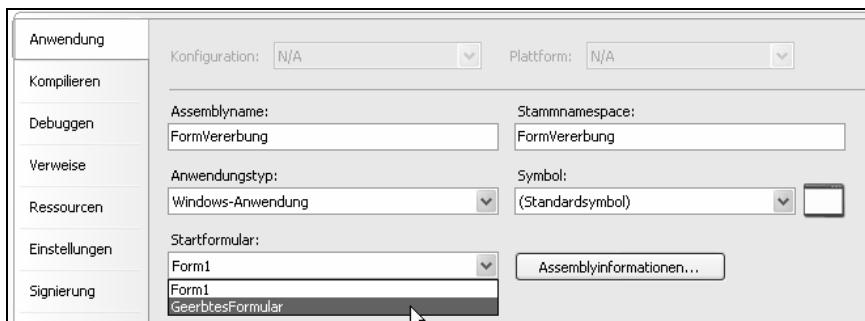


**Abbildung 27.31:** Die Steuerelemente des Basisdialogs sind zwar alle vorhanden, aber ihre Eigenschaften lassen sich in diesem Zustand nicht verändern, da sie standardmäßig als *Friend* deklariert, damit nicht überschreibbar und somit aus dem vererbten Formular heraus nicht manipulierbar sind

Fürs Erste behalten Sie diese Erkenntnis einfach nur im Hinterkopf – ich werde später darauf zurückkommen.

Lassen Sie uns vorerst herausfinden, wie wir aus dem Vererben des Formulars nun einen Nutzen ziehen können. Zu diesem Zweck fügen Sie unter den vorhandenen TextBox-Steuerelementen zwei weitere ein – denn das ist durchaus möglich!

Öffnen Sie anschließend die Projekteigenschaften (rechte Maustaste für das Kontextmenü im Projektmappen-Explorer über dem Projektnamen), und wählen Sie als Startobjekt das neue, vererbte Formular *GeerbtesFormular* aus. Starten Sie das Programm anschließend.



**Abbildung 27.32:** Legen Sie hier das geerbte Formular als neues Startformular fest

Geben Sie in alle Eingabefelder einen beliebigen Text ein, und beobachten Sie, was geschieht, wenn Sie auf die Schaltfläche klicken (siehe Abbildung 27.33).



**Abbildung 27.33:** Keine Zeile Code ist hinzugekommen, und dennoch funktioniert das Programm automatisch auch mit den neu hinzugefügten Komponenten im abgeleiteten Formular!

Überrascht? Sie haben nicht eine einzige Zeile am Code geändert und dennoch – dank Polymorphie – hat der ursprünglich in *Form1* implementierte Code auch in dieser Ableitung seine volle Gültigkeit.

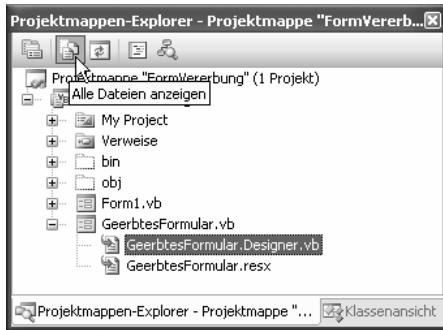
## Ein erster Blick auf den Designer-Code eines Formulars

Vielleicht sind Sie sogar ein weiteres Mal überrascht, wenn Sie sich jetzt, nachdem Sie das Programm beendet haben, auf die Suche nach dem Code in der abgeleiteten Klasse begeben. Wechseln Sie mit dem Projektmappen-Explorer in die Codeansicht von *GeerbtesFormular.vb*, und bestaunen Sie, was der Visual Studio-Designer aus dem Code gemacht hat:

```
Public Class GeerbtesFormular
```

```
End Class
```

Nichts, absolut gar nichts an zusätzlichem Code ist hier zu sehen. Im Übrigen nicht einmal der Hinweis auf eine Vererbung?!



**Abbildung 27.34:** Nur mit Klick auf dieses Symbol können Sie alle Dateien eines Projektes anzeigen lassen. Bei mehreren Projekten in einer Projektmappe müssen Sie diese Einstellung für jedes Projekt einer Projektmappe wiederholen.

Des Rätsels Lösung liegt darin, dass der Designer den Designercode in Visual Basic 2005 in einer besonderen Codedatei versteckt, die Sie normalerweise gar nicht zu Gesicht bekommen.

Um Sie dennoch zu sehen, klicken Sie im Projektmappenexplorer auf das Symbol *Alle Dateien anzeigen* (siehe Abbildung 27.34). Erst dann können Sie den Zweig vor jedem Formular aufklappen, und Sie werden feststellen, dass jedes in Visual Basic 2005 erstellte Formular eigentlich mindestens aus zwei Dateien besteht. Der Designer-Code, also der Code, der zum einen vom Designer erstellt wurde und der zum anderen dafür zuständig ist, das alle Steuerelemente des Formulars rechtzeitig instantiiert und auf dem Formular dargestellt werden, befindet sich in der Datei mit der Endung *.Designer.vb*.

```
<Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()> _
Partial Class GeerbtesFormular
    Inherits FormVererbung.Form1

    'Das Formular überschreibt den Löschvorgang, um die Komponentenliste zu bereinigen.
    <System.Diagnostics.DebuggerNonUserCode()> _
    Protected Overrides Sub Dispose(ByVal disposing As Boolean)
        If disposing AndAlso components IsNot Nothing Then
            components.Dispose()
        End If
        MyBase.Dispose(disposing)
    End Sub

    'Wird vom Windows Form-Designer benötigt.
    Private components As System.ComponentModel.IContainer

    'Hinweis: Die folgende Prozedur ist für den Windows Form-Designer erforderlich.
    'Das Bearbeiten ist mit dem Windows Form-Designer möglich.
    'Das Bearbeiten mit dem Codeeditor ist nicht möglich.
    <System.Diagnostics.DebuggerStepThrough()> _
    Private Sub InitializeComponent()
        Me.TextBox4 = New System.Windows.Forms.TextBox
        Me.TextBox5 = New System.Windows.Forms.TextBox
        Me.SuspendLayout()

        'TextBox4

        Me.TextBox4.Location = New System.Drawing.Point(16, 147)
        Me.TextBox4.Name = "TextBox4"
        Me.TextBox4.Size = New System.Drawing.Size(256, 20)
        Me.TextBox4.TabIndex = 5

        'TextBox5

        Me.TextBox5.Location = New System.Drawing.Point(16, 173)
        Me.TextBox5.Name = "TextBox5"
        Me.TextBox5.Size = New System.Drawing.Size(256, 20)
        Me.TextBox5.TabIndex = 6

        'GeerbtesFormular

        Me.ClientSize = New System.Drawing.Size(416, 262)
```

```

Me.Controls.Add(Me.TextBox5)
Me.Controls.Add(Me.TextBox4)
Me.Name = "GeerbtesFormular"
Me.Controls.SetChildIndex(Me.TextBox4, 0)
Me.Controls.SetChildIndex(Me.TextBox5, 0)
Me.ResumeLayout(False)
Me.PerformLayout()

End Sub
Friend WithEvents TextBox4 As System.Windows.Forms.TextBox
Friend WithEvents TextBox5 As System.Windows.Forms.TextBox

End Class

```

Mithilfe des Konzepts der partiellen Klassen (mehr zum Schlüsselwort `Partial` finden Sie in ► Kapitel 6) kann der Klassencode den manchmal störenden Designer Code in einer eigene Codedatei ablegen. Die eigentliche Vererbungsanweisung befindet sich, wie Sie sich anhand der ersten fett hervorgehobenen Zeilen dieses Listings selbst überzeugen können, ebenfalls in dieser Codedatei.

Ein Formular ist ja nichts weiter als eine ganz normale Klasse, und wenn Sie sie aus einer Basisklasse ableiten, erbt sie alle Eigenschaften dieser – und damit natürlich auch die Ereignisbehandlungsme-thode des `Click`-Ereignisses.

Das abgeleitete Formular verfügt über einen Unterschied zur Basisklasse, den allerdings nicht wir, sondern der Designer als Code in das Listing eingefügt hat:

Es sind dies, wie Sie hier sehen können, die zusätzlichen Definitionen der beiden `TextBox`-Steuerelemente. Was passiert nun genau, wenn Sie das Programm starten und im vererbten Formular auf die Schaltfläche klicken? Die nachstehende Abfolgenbeschreibung macht das deutlich:

- Wenn Sie das Programm starten, wird der Konstruktor der abgeleiteten Klasse aufgerufen. Dieser ruft zunächst den Konstruktor der Basisklasse auf, der seinerseits `InitializeComponent` der Basis-klasse aufruft. Damit sind die ursprünglichen drei `TextBox`-Steuerelemente im Formular vorhanden.
- Anschließend ruft der Konstruktor `InitializeComponent` seiner eigenen Klasse auf. Da es für jedes Formular nur eine `Controls`-Auflistung gibt (sie enthält die Steuerelemente eines Formulars und ist aus der Basisklasse entstanden), werden die beiden neuen `TextBox`-Steuerelemente zu den bereits vorhandenen in der Auflistung hinzugefügt.
- Klickt der Anwender zur Laufzeit auf die Schaltfläche, werden alle Texte aus den in der `Controls`-Auflistung vorhandenen `TextBox`-Steuerelementen ausgelesen – dazu gehören jetzt auch die beiden neuen `TextBox`-Steuerelemente.

Das Ergebnis: Die Texte aller fünf Steuerelemente werden im Nachrichtenfeld dargestellt – Poly-morphie ist eine feine Sache, finden Sie nicht?

## Modifizierer von Steuerelementen in geerbten Formularen

Wenn Sie sich das zuvor abgedruckte Listing des Designer Codes genau anschauen, dann werden Sie feststellen, dass die Variablen, die die TextBox-Steuerelemente instanzieren, mit dem Friend-Modifizierer gekennzeichnet sind. Nun eignet sich der Friend-Modifizierer genau wie Protected, Public oder Protected Friend gleichermaßen, einer geerbten Klasse in der gleichen Assembly (in der gleichen DLL oder der gleichen EXE-Datei) Zugriff auf derart deklarierte Member der Basisklasse zu verschaffen. Dennoch sieht es so aus, als wenn dem Designer der Zugriff verwehrt wäre – denn Sie können Steuerelemente, die als Friend deklariert sind, offensichtlich nicht in der abgeleiteten Klasse verändern. Das liegt daran, dass der Designer, der die Klasse zur Darstellung auf seine eigene Weise verwendet, sich in einer anderen Assembly befindet als das Formular, das Sie ableiten: Er instanziert es nicht nur, sondern leitet es auf seiner Basis neu ab – durch den Friend-Zugriffsmodifizierer kann er dann natürlich nicht mehr auf die Eigenschaften der Steuerelemente (die nunmehr versteckte Member der Ursprungsklasse sind) zugreifen. Anders ist das, wenn Sie die Modifiers-Eigenschaft eines Steuerelements in Protected, Protected Friend oder Public ändern. Jetzt kann der Designer auch auf die so definierten Steuerelemente des abgeleiteten Formulars zugreifen, und Sie können die Eigenschaften dieser Steuerelemente verändern und sie damit auch beispielsweise positionstechnisch verändern.

Probieren Sie es aus:

- Klicken Sie auf die Registerkarte *Form1.vb [Entwurf]*, um das Basisformular in der Entwurfsansicht anzeigen zu lassen.
- Klicken Sie die erste TextBox an, und ändern Sie im Eigenschaftenfenster ihre Modifiers-Eigenschaft auf Protected. Sobald Sie diese Änderung vorgenommen haben, zeigt Ihnen die Aufgabenliste einen Hinweis an, etwa wie in Abbildung 27.35 zu sehen.



**Abbildung 27.35:** Wenn Sie Änderungen am Basisformular vorgenommen haben, müssen Sie die Anwendung neu erstellen, damit sich die Änderungen auf die Ableitungen auswirken

- Erstellen Sie die Anwendung neu, indem Sie aus dem Menü *Erstellen* den Menüpunkt *Projektmappe neu erstellen* wählen.
- Lassen Sie anschließend den Designer zum vererbten Form *GeerbtesForm.vb [Entwurf]* anzeigen.

Sie sehen, dass Sie die erste TextBox jetzt nach Belieben verändern können. Sowohl die Position lässt sich beliebig anpassen als auch andere Eigenschaften der TextBox. Der Name des Steuerelements ist natürlich nach wie vor unveränderlich.

# 28 Im Motorraum von Formularen und Steuerelementen

---

- 
- 821 Über das Vererben von Form und die Geheimnisse des Designer-Codes
  - 827 Ereignisbehandlung von Formularen und Steuerelementen
  - 834 Wer oder was löst welche Formular- bzw. Steuerelementereignisse wann aus?
- 

Nachdem Sie Ihre ersten Anwendungen mit Visual Basic erstellt haben, gehen Ihnen bestimmte Prinzipien beim Verdrahten von Benutzeroberflächen mit den eigentlichen Funktionalitäten Ihrer Programme in Fleisch und Blut über. So wissen Sie nach einer Weile einfach, wie Sie dafür zu sorgen haben, dass beim Klicken einer Schaltfläche eine entsprechende Ereignisbehandlungsmethode ausgeführt werden soll.

Doch was passiert eigentlich genau, wenn der Benutzer ein Element bedient? In welcher Reihenfolge treten welche Ereignisse auf, wenn ein Formular oder ein Steuerelement dargestellt wird? Und wie gliedern sich .NET-Framework-Windows-Anwendungen in das »alte« Betriebssystem (also vor Windows Vista) eigentlich ein? Dieses Kapitel möchte ein wenig Licht ins Dunkel bringen und auf diese Fragen Antworten geben.

---

**BEGLEITDATEIEN:** Das Beispielprojekt für die folgenden Abschnitt finden Sie im Verzeichnis *|VB 2005 - Entwicklerbuch|G - SmartClient\Kap28\PictureViewer|*.

---

## Über das Vererben von Form und die Geheimnisse des Designer-Codes

Immer wenn Sie Ihrem Projekt ein neues Windows-Formular hinzufügen, legt Visual Basic nicht nur eine einfache Klassendatei an, die den Formularaufbau enthält. Die Wahrheit ist: Wenn Sie ein neues Formular anlegen, und es im Codeeditor öffnen, dann sehen Sie zunächst außer dem Klassenrumpf erst einmal gar nichts. Im Beispielprojekt gibt es das Formular LeeresFormular.vb, das, wenn Sie es im Codeeditor öffnen, den folgenden Code bereits hält:

```
Public Class LeeresFormular  
End Class
```

Öffnen Sie das Formular jedoch mit dem Designer, stellen Sie fest, dass es bereits eine ganze Reihe von Steuerelementen gibt. Nur: Wo ist denn nur die Information für diese Steuerelemente untergebracht?

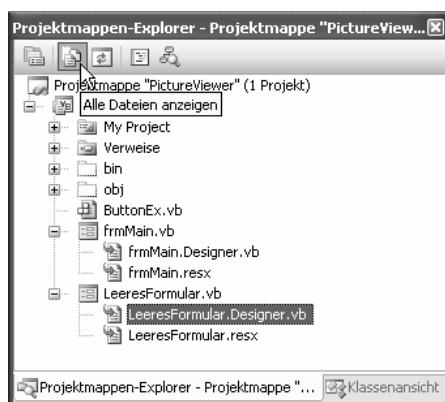
Gerade für Visual Basic .NET-Umsteiger, die ihre ersten Gehversuche mit Visual Basic 2005 und Windows Forms-Anwendungen machen, sieht diese Tatsache auf den ersten Blick geradezu erschreckend aus, weil die für .NET-Programmiersprachen postulierte »OOP-Konsequenz« gebrochen zu sein scheint.

Es gibt weder einen Hinweis darauf, dass die Grundfunktionalität des Formulars sich aus irgendeiner im .NET-Framework schon vorhandenen Klasse ableitet, noch, an welcher Stelle die Platzierung der Steuerelemente im Formular zur Laufzeit erfolgt.

Wenn Sie das vorherige Kapitel aufmerksam studiert haben, wissen Sie es bereits:

Des Rätsels Lösung liegt darin, dass der spezielle Code zur eigentlichen Darstellung der Steuerelemente eines jeden Formulars in Visual Basic 2005 in einer besonderen Codedatei versteckt ist, die Sie normalerweise gar nicht zu Gesicht bekommen.

Um Sie dennoch zu sehen, klicken Sie im Projektmappenexplorer auf das Symbol *Alle Dateien anzeigen* (siehe Abbildung 28.1).



**Abbildung 28.1:** Nur mit Klick auf dieses Symbol können Sie alle Dateien eines Projektes anzeigen lassen, und damit auch Zugriff auf den Designer-Code eines Formulars nehmen. Bei mehreren Projekten in einer Projektmappe müssen Sie diese Einstellung für *jedes* Projekt einer Projektmappe wiederholen.

Erst dann können Sie den Zweig vor jedem Formular aufklappen, und Sie werden feststellen, dass jedes in Visual Basic 2005 erstellte Formular eigentlich mindestens aus zwei Dateien besteht. Der Designer-Code, also der Code, der zum einen vom Designer erstellt wurde und der zum anderen dafür zuständig ist, dass alle Steuerelemente des Formulars rechtzeitig instanziert und auf dem Formular dargestellt werden, befindet sich in der Datei mit der Endung *.Designer.vb*.

Und ohne dass der Entwickler im Formular *LeeresFormular.vb* nur eine einzige Zeile selbst programmiert hat, beträgt der Umfang Ihrer Formularklasse zu diesem Zeitpunkt schon weit über 100 Zeilen, denn:

In der Datei *LeeresFormular.Designer.vb* befinden sich die Codezeilen, die der Windows-Designer (eigentlich: *die* Designer, denn jedes Steuerelement verfügt streng genommen über seinen eigenen – siehe dazu den entsprechenden grauen Kasten in ► Kapitel 3) beim Anlegen des Projektes und bei jedem Hinzufügen der Komponenten und dem dazugehörigen Einstellen der Eigenschaften produziert hat.

Auch die Angabe der Klassenableitung mit `Inherits` steht in diesem Designer-Teil des Formularcodes – und mit dieser Technik wird zweierlei erreicht:

- Das OOP-Konzept für Framework-Klassen ist auch in Visual Basic 2005 nicht verletzt.
- Dennoch ist die Codedatei, in der Sie Ihren eigenen programmtechnischen Teil zum Funktionieren des Formulars beitragen, sauber aufgeräumt und bleibt stets übersichtlich. Der Designer-Code steht buchstäblich völlig außen vor, und sie bekommen ihn nur zu Gesicht, wenn Sie es ausdrücklich wollen.

Es ist aber dennoch wichtig, dass Sie verstehen, was beim Anzeigen eines Formulars zur Laufzeit passiert, denn nur dann haben Sie die Möglichkeit, ins Geschehen einzugreifen, wenn mal etwas nicht so funktioniert, wie Sie es sich gedacht haben, oder wenn Sie Erweiterungen implementieren müssen, die über die Standardimplementierung hinausgehen.

Die folgenden Abschnitte beschreiben daher die Funktionsweise des Designer-Codes am Beispiel.

## Geburt und Tod eines Formulars – New und Dispose

Wenn Sie Ihrem Formular ein neues Formular hinzufügen, dann bekommt es automatisch vom Designer und einer `Dispose`-Methode verpasst.

Was in Visual Basic 2005 erstellten Formularen allerdings völlig fehlt, ist ein Konstruktor, und – nehmen wir die Tatsache vorweg, dass gerade der fehlende Konstruktor dafür zuständig ist, die Methode `InitializeComponent` aufzurufen, mit dem das eigentliche Einrichten und Platzieren der Steuerelemente des Formulars erfolgt – es stellt sich die Frage, wer oder was ruft `InitializeComponent` überhaupt auf?

Eine offensichtliche Antwort könnte lauten: Der Konstruktor der Basisklasse, denn der wird, wie Sie wissen, wenn Sie den Klassenteil dieses Buches aufmerksam studiert haben, in vererbten Klassen implizit aufgerufen. Doch das ist nicht der Fall.

Die Wahrheit ist: Der Visual Basic-Compiler kompiliert »virtuellen« Code beim Kompilieren in die Klasse hinein. Würden Sie das Kompilat des Beispielprogramms mit einem geeigneten Tool wieder zurück in seinen Visual Basic-Quelltext übersetzen, dann ergäbe sich nämlich Folgendes:

```
<DebuggerNonUserCode> _
Public Sub New()
    'Diese Zeile hilft beim schnellen Wiederauferstehen lassen des Formulars,
    'falls es doch nochmal benötigt werden sollte.
    frmMain.ENCList.Add(New WeakReference(Me))
    'Hier wird InitializeComponent aufgerufen
    Me.InitializeComponent
End Sub
```

Dieses Verhalten wird auch dadurch unterstrichen, dass etwas Außergewöhnliches passiert, wenn Sie in der eigentlichen Formularklasse einen parameterlosen Konstruktor einfügen. Der Editor zaubert nämlich aus der einfachen Eingabe von **Sub New** einen Coderumpf, wie Sie ihn auch in Abbildung 28.2 sehen können.

```

Public Class LeeresFormular
    Sub New()
        ' Dieser Aufruf ist für den Windows Form-Designer erforderlich.
        InitializeComponent()

        ' Fügen Sie Initialisierungen nach dem InitializeComponent()-Aufruf hinzu.
    End Sub
End Class

```

**Abbildung 28.2:** Wenn Sie einen Konstruktor durch die Eingabe von Sub New im Formularcode (nicht Designer-Code!) einfügen, ergänzt der Editor den Code um einen wesentlichen Aufruf

## Sub New des Formulars

Egal, wo also die Sub New des Formulars also letzten Endes herkommt – sie gestaltet sich grundsätzlich folgendermaßen:

```

Public Sub New()
    MyBase.New()

    ' Dieser Aufruf ist für den Windows Form-Designer erforderlich.
    InitializeComponent()

    ' Initialisierungen nach dem Aufruf InitializeComponent() hinzufügen.
End Sub

```

Der Konstruktor – repräsentiert durch Sub New – ist existenziell für das saubere Funktionieren des Formulars. Der Formularkonstruktor wird nämlich nicht nur durch den Designer aufgerufen, wenn Sie das Formular zur Entwurfszeit bearbeiten, er wird natürlich auch vom Anwendungsframework benötigt, das das Formular instanziert und darstellt, wenn das Formular das Startformular der Anwendung ist.

## InitializeComponent des Formulars

Der Konstruktor des Formulars ruft anschließend die Prozedur InitializeComponent auf. Und hier wird es interessant, denn InitializeComponent erledigt den Job, das Formular mit sinnvollen Inhalten zu füllen.

Um das zu tun, müssen die Komponenten und Steuerelemente, mit denen der Anwender später das Formular bedient, für den Gebrauch im Formular vorbereitet werden. Steuerelemente und Komponenten sind wie alle anderen Elemente in .NET Objekte. Und Objekte entstehen aus Klassen. Das heißt: Für jedes Steuerelement, das Sie zur Entwurfszeit auf das Formular gezogen haben, hat der Designer eine Objektvariable bereitgestellt. Die Deklaration dieser Objektvariablen finden Sie im Beispiel ganz am Ende des Designer-Klassencodes:

```

Friend WithEvents picViewArea As System.Windows.Forms.PictureBox
Friend WithEvents btnNextBitmap As PictureBox.ButtonEx
Friend WithEvents btnOpenBitmap As PictureBox.ButtonEx
Friend WithEvents btnQuitProgram As System.Windows.Forms.Button

```

```
Friend WithEvents btnSaveBitmap As System.Windows.Forms.Button
Friend WithEvents pnlPicture As System.Windows.Forms.Panel
```

Sie werden feststellen, dass die Namensgebung der Objektvariablen genau der entspricht, die Sie im Eigenschaftenfenster mit der Name-Eigenschaft vorgenommen haben. Sie stellen aber ebenfalls fest, dass die Komponentenvariablen mit dem Schlüsselwort `WithEvents` deklariert wurden. Diese Deklaration zeigt an, dass es über die jeweilige Objektvariable möglich ist, Ereignisse zu empfangen, die an eine zum Ereignis kompatible Prozedur delegiert werden kann. Mehr zu diesem Thema finden Sie übrigens in ► Kapitel 15.

Schauen wir uns als nächstes den ersten Teil von `InitializeComponent` genauer an:

```
'Hinweis: Die folgende Prozedur ist für den Windows Form-Designer erforderlich.
'Das Bearbeiten ist mit dem Windows Form-Designer möglich.
'Das Bearbeiten mit dem Codeeditor ist nicht möglich.
<System.Diagnostics.DebuggerStepThrough()> _
Private Sub InitializeComponent()
    Me.picViewArea = New System.Windows.Forms.PictureBox
    Me.btnExitBitmap = New PictureViewer.ButtonEx
    Me.btnOpenBitmap = New PictureViewer.ButtonEx
    Me.btnExitProgram = New System.Windows.Forms.Button
    Me.btnSaveBitmap = New System.Windows.Forms.Button
    Me.pnlPicture = New System.Windows.Forms.Panel
```

Alle innerhalb des Formulars verwendeten Komponenten werden in diesem Teil instanziert. Das Attribut `DebuggerStepThrough`<sup>1</sup> gibt dem Debugger an, dass er nicht in diesen Teil des Programms schrittweise eintreten darf (auch, wenn Sie das Programm im Einzelschrittmodus debuggen und er dies eigentlich sollte).

### Aussetzen der Layout-Logik – SuspendLayout und ResumeLayout

Um die nächsten beiden Codezeilen ranken sich in der Entwicklergemeinde mehr Mythen als Wahrheiten. Es geht um die Methode `SuspendLayout`, und Sie finden sie im besprochenen Beispielcode in den folgenden Codezeilen:

```
Me.pnlPicture.SuspendLayout()
Me.SuspendLayout()
```

Viele Entwickler glauben, dass sie mit `SuspendLayout` verhindern können, dass ein Steuerelement, das einem Container (das kann ein Formular oder ein Steuerelement sein, das andere Steuerelemente enthält, wie beispielsweise das `GroupBox`-Steuerelement) zugeordnet ist, gezeichnet wird, während es sich im `Suspend`-Zustand befindet. Das ist falsch. `SuspendLayout`, angewendet auf einen Container, der andere Steuerelemente enthält, sorgt lediglich dafür, dass das Layout-Ereignis<sup>2</sup> des Steuerelements nicht ausgelöst wird. Das Layout-Ereignis tritt dann ein, wenn einem Steuerelement, das als Container

---

<sup>1</sup> Sinngemäß übersetzt: »Debugger, überspring dies«.

<sup>2</sup> Mehr Informationen zu Ereignissen erfahren Sie zu einem späteren Zeitpunkt in diesem Kapitel. Fürs Erste reicht es zu wissen, dass Ereignisse dazu da sind, automatisch bestimmte Prozeduren auszuführen, wenn ein bestimmter Zustand eintritt. Die dem `Click`-Ereignis zugewiesene Prozedur beispielsweise wird also dann automatisch aufgerufen, wenn der Zustand »Anwender hat Schaltfläche angeklickt« eintritt.

fungiert, weitere Steuerelemente hinzugefügt werden, wenn Steuerelemente aus ihm entfernt werden oder wenn sich die Begrenzungen eines Steuerelements ändern.

Wieso ist es aber so wichtig, das Layout-Ereignis zu unterdrücken? Aus zweierlei Gründen. Zum einen ergibt es gerade beim Initialisieren eines Steuerelements aus Geschwindigkeitsgründen keinen Sinn, auf jede Eigenschaft zu reagieren, die das Layout verändert. Es reicht, wenn sich ein Steuerelement an das neue Layout anpasst, wenn alle seine Eigenschaften vollständig gesetzt sind. Zum anderen kann es gerade beim Initialisieren zum Dilemma kommen, wenn bestimmte Eigenschaften sich gegenseitig beeinflussen, und eine das Layout beeinflussende Eigenschaft, die weiter hinten im Programmcode ausgeführt wird, durch das *Layout*-Ereignis indirekt eine Eigenschaft verändert, die bereits gesetzt wurde und so Zirkelaufrufe entstehen könnten.

Die restlichen Zeilen im `InitializeComponent`-Code sind übrigens lediglich dafür verantwortlich, die geänderten Eigenschaften für die Steuerelemente so zu setzen, wie Sie sie im Eigenschaftenfenster zur Entwurfszeit definiert haben. Teilweise hilft der Code dabei ein wenig, wie beispielsweise beim Setzen der `Anchor`-Eigenschaft, dessen umständliche Formulierung

```
Me.pnlPicture.Anchor = CType(((System.Windows.Forms.AnchorStyles.Top Or _  
    System.Windows.Forms.AnchorStyles.Bottom) _  
    Or System.Windows.Forms.AnchorStyles.Left) _  
    Or System.Windows.Forms.AnchorStyles.Right),  
System.Windows.Forms.AnchorStyles)
```

auch einfach nur mit der Zeile

```
Me.pnlPicture.Anchor = AnchorStyles.Top Or AnchorStyles.Left Or AnchorStyles.Right Or AnchorStyles.Top
```

funktionieren würde; aber natürlich wurde diese Zeile nicht von Menschenhand, sondern durch eine Maschine erzeugt. Und vermutlich, um allgemein gültige Algorithmen bei der Codegenerierung einsetzen zu können, gibt es hier und da schon einmal Typ-Castings, wo keine nötig wären, aber offensichtlich schadet es auch nicht.

### **Steuerelemente auf dem Formular mit der ControlCollection sichtbar machen**

Ungleich interessanter sind die letzten Zeilen von `InitializeComponent`, die dafür sorgen, dass die zu dieser Zeit bereits instanzierten Steuerelemente auch auf dem Formular sichtbar werden. Jedes von der `Control`-Klasse abgeleitete Objekt – und dazu gehört auch `Form` – verfügt über eine so genannte *Controls-Auflistung*, die die Steuerelemente enthält, die dieses als Container fungierende Steuerelement enthält. Ein instanziertes Formular ist nichts weiter als ein erweitertes `Control`-Objekt, also gilt das für ein Formular gleichermaßen. In dem Moment, in dem ein Objekt mit einer instanzierten `Control`-Klasse (oder einer von `Control` abgeleiteten) der *Controls-Auflistung* einem `Control`-Objekt mit `Add` hinzugefügt wurde, wird das Steuerelement auch im Container (im Beispiel also dem Formular) sichtbar. Voraussetzung dafür ist natürlich, dass es sich um eine sichtbare Komponente handelt und dessen `Visible`-Eigenschaft auf `True` gesetzt wurde. Mal abgesehen von den nicht in diesen Zusammenhang passenden Zeilen

```
Me.AutoScaleDimensions = New System.Drawing.SizeF(6.0!, 13.0!)  
Me.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font  
Me.ClientSize = New System.Drawing.Size(582, 431)
```

die kurz vorher noch die Größeneinstellungen des Formulars vornehmen, dienen die folgenden Codezeilen schließlich dazu, den Komponenten die Möglichkeit zu geben, im Formular sichtbar zu werden.

```

Me.Controls.Add(Me.btnExit)
Me.Controls.Add(Me.btnOpen)
Me.Controls.Add(Me.btnExit)
Me.Controls.Add(Me.btnSave)
Me.Controls.Add(Me.pnlPicture)
Me.Name = "LeeresFormular"
Me.Text = "LeeresFormular"
Me.pnlPicture.ResumeLayout(False)
Me.ResumeLayout(False)

```

Die beiden letzten Zeilen schließlich sorgen dafür, dass die Layout-Ereignisse für Panel und das gesamte Formular wieder in der gewohnten Weise stattfinden können.

Um das Verhalten zu verdeutlichen: Wenn Sie der Controls-Auflistung eines Formulars eine Instanz einer TextBox-Komponente mit Add hinzufügen, ist die TextBox (entsprechend eingestellte Eigenschaften vorausgesetzt) direkt nach dem Ausführen der Add-Methode im Formular sichtbar und einsatzbereit.

## Ereignisbehandlung von Formularen und Steuerelementen

Ereignisse haben gerade bei der Programmierung von Windows-Anwendungen eine ganz zentrale Bedeutung. Wann immer der Anwender Ihres Programms im weitesten Sinne „Irgendetwas“ macht, löst er damit ein Ereignis aus, auf das Sie reagieren können.



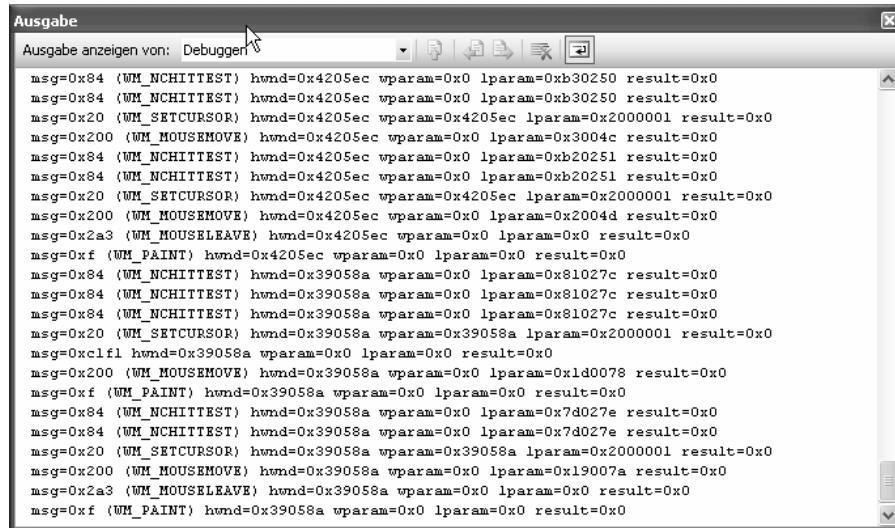
**Abbildung 28.3:** Mit dem PictureViewer können Sie beliebige Grafiken eines Verzeichnisses betrachten

Das wohl einfachste Beispiel ist der Klick auf eine Schaltfläche, und es mag Ihnen als erfahrenem Entwickler ein wenig überflüssig vorkommen, etwas über die Funktionsweise des *Click*-Ereignisses zu erfahren, doch wissen Sie genau, was unter der Haube passiert?

Dazu werfen wir an dieser Stelle einen ersten Blick auf das Beispielprogramm selbst, indem wir es starten. Nach dem Start des Programms sehen Sie einen Dialog auf dem Bildschirm, wie er auch in Abbildung 28.3 zu erkennen ist.

Auf den ersten Blick scheint es überflüssig zu sein, die Funktionen dieses Beispiels zu erklären, schließlich wirkt jede Funktion so offensichtlich!

Doch eine Besonderheit hat das Programm, von der Sie Notiz nehmen können, wenn Sie mit dem Mauszeiger über eine der Schaltflächen *Grafik öffnen* oder *Nächste Grafik* fahren. Achten Sie dazu auf das, was im Ausgabefenster von Visual Studio angezeigt wird.



The screenshot shows the 'Ausgabe' (Output) window in Visual Studio. The title bar says 'Ausgabe' and 'Ausgabe anzeigen von: Debugger'. The window contains a list of native Windows messages (msgs) with their parameters and results. The messages include:

```
msg=0x84 (WM_NCHITTEST) hwnd=0x4205ec wparam=0x0 lparam=0xb30250 result=0x0
msg=0x84 (WM_NCHITTEST) hwnd=0x4205ec wparam=0x0 lparam=0xb30250 result=0x0
msg=0x20 (WM_SETCURSOR) hwnd=0x4205ec wparam=0x4205ec lparam=0x2000001 result=0x0
msg=0x200 (WM_MOUSEMOVE) hwnd=0x4205ec wparam=0x0 lparam=0x3004c result=0x0
msg=0x84 (WM_NCHITTEST) hwnd=0x4205ec wparam=0x0 lparam=0xb20251 result=0x0
msg=0x84 (WM_NCHITTEST) hwnd=0x4205ec wparam=0x0 lparam=0xb20251 result=0x0
msg=0x20 (WM_SETCURSOR) hwnd=0x4205ec wparam=0x4205ec lparam=0x2000001 result=0x0
msg=0x200 (WM_MOUSEMOVE) hwnd=0x4205ec wparam=0x0 lparam=0x2004d result=0x0
msg=0x2a3 (WM_MOUSELEAVE) hwnd=0x4205ec wparam=0x0 lparam=0x0 result=0x0
msg=0xf (WM_PAINT) hwnd=0x4205ec wparam=0x0 lparam=0x0 result=0x0
msg=0x84 (WM_NCHITTEST) hwnd=0x39058a wparam=0x0 lparam=0x81027c result=0x0
msg=0x84 (WM_NCHITTEST) hwnd=0x39058a wparam=0x0 lparam=0x81027c result=0x0
msg=0x84 (WM_NCHITTEST) hwnd=0x39058a wparam=0x0 lparam=0x81027c result=0x0
msg=0x20 (WM_SETCURSOR) hwnd=0x39058a wparam=0x39058a lparam=0x2000001 result=0x0
msg=0xc1f1 hwnd=0x39058a wparam=0x0 lparam=0x0 result=0x0
msg=0x200 (WM_MOUSEMOVE) hwnd=0x39058a wparam=0x0 lparam=0x1d0078 result=0x0
msg=0xf (WM_PAINT) hwnd=0x39058a wparam=0x0 lparam=0x0 result=0x0
msg=0x84 (WM_NCHITTEST) hwnd=0x39058a wparam=0x0 lparam=0x7d027e result=0x0
msg=0x84 (WM_NCHITTEST) hwnd=0x39058a wparam=0x0 lparam=0x7d027e result=0x0
msg=0x20 (WM_SETCURSOR) hwnd=0x39058a wparam=0x39058a lparam=0x2000001 result=0x0
msg=0x200 (WM_MOUSEMOVE) hwnd=0x39058a wparam=0x0 lparam=0x19007a result=0x0
msg=0x2a3 (WM_MOUSELEAVE) hwnd=0x39058a wparam=0x0 lparam=0x0 result=0x0
msg=0xf (WM_PAINT) hwnd=0x39058a wparam=0x0 lparam=0x0 result=0x0
```

**Abbildung 28.4:** Wenn Sie eine der beiden Schaltflächen *Grafik öffnen* oder *Nächste Grafik* mit der Maus berühren, über sie hinwegfahren und diese betätigen, zeigt das Ausgabefenster die nativen Nachrichten der Windows-Nachrichtenschlange an

Um zu verstehen, was genau passiert, und dieses Verständnis für die funktionelle Erweiterung von Steuerelementen zu nutzen, ist allerdings ein detailliertes Verständnis für die internen Abläufe bei auf Windows-Nachrichten basierenden Ereignissen von Steuerelementen erforderlich. Der folgende Abschnitt liefert diese Grundlagen, doch nehmen Sie sich für seine Lektüre ein wenig Zeit: Er hat es nämlich »in sich«!

## Vom Programmstart mithilfe des Anwendungsframeworks über eine Benutzeraktion zur Ereignisauslösung

Solange Sie nichts anderes sagen, verwenden Windows Forms-Anwendungen das Anwendungsframework, um ein Windows-Programm ans Laufen zu bekommen. Ganz vereinfacht ausgedrückt startet dabei zunächst eigentlich nicht ihr eigenes Programm, sondern ein völlig anderer Prozess, der mit

Ihrem Programm noch nichts zu tun hat. Das ist notwendig, damit eine automatisierte Aktualisierung einer SmartClient-Anwendung (über das Thema ClickOnce-Deployment, die hier den Rahmen weit sprengen würde, hält die Online-Hilfe Genaueres bereit) überhaupt funktionieren kann – denn Programmdateien können nicht ausgetauscht werden, wenn das Programm, dessen Dateien ausgetauscht werden müssen, bereits läuft.

Dieser Prozess ist übrigens auch dafür zuständig, einen eventuell darzustellenden Begrüßungsdialog anzuzeigen (den Sie ganz einfach in den Projekteinstellungen definieren können) und schließlich auf den eigentlichen Prozess Ihrer Anwendung zu initiieren und eine so genannte Windows-Nachrichtenwarteschlange für das Startformular Ihrer Anwendung einzurichten.

Diese Warteschlange ist, stark vereinfacht ausgedrückt, eine Endlosschleife, die zunächst nichts weiter macht, als zu überprüfen, ob es irgendwelche neuen Meldungen vom Windows-System gibt, die an den Thread gerichtet sind, in dem sie ausgeführt werden.

Das Hauptformular spielt in der Warteschlange dabei eigentlich gar keine besondere Rolle – jedenfalls was die Verarbeitung der Warteschlange an sich anbelangt. Es wird mit seiner Dispose-Methode nur an den so genannten *Application Context* (etwa: Anwendungskontext) »gebunden«, und das bewirkt, dass die Warteschlangenroutine des laufenden Prozesses (und damit letzten Endes auch die Anwendung) beendet wird, wenn das Formular geschlossen wird.

Der Schlüssel der ganzen Ereignissesteuerung für Windows-Nachrichten liegt eigentlich in dem Zusammenspiel zwischen der Control-Klasse und einer intern verwendeten Klasse, die sich NativeWindow nennt – und damit sind wir mitten im Thema, nämlich was passiert, wenn der Anwender beispielsweise eine Schaltfläche (die ja eine wenn auch abgeleitete Control-Klasse darstellt) betätigt.

Wichtig ist es zunächst einmal zu wissen, dass jedes noch so kleine Element im Windows-Betriebssystem (Schaltfläche, PictureBox – sogar ein angezeigter Tooltip) ein Window darstellt, genauso wie ein Fenster, dass Sie selbst erst als Window bezeichneten würden. Der Ereignisverlauf von einem richtigen »Windows« ist also prinzipiell kein anderer als der einer Schaltfläche, die ja schließlich auch ein Window im Windows-Betriebssystemsinnern darstellt.

Schon vor .NET gab es OOP und wenn Sie sich einmal die MFC (Microsoft Foundation Class) unter C++ anschauen (Windows als Betriebssystem wurde fast vollständig in C und C++ geschrieben), sehen Sie, dass ein Button wirklich ein Fenster IST – es ist nämlich eine abgeleitete Klasse, die sie auch wieder zu einem Window downcasten können.

Sobald die Visible-Eigenschaft eines Control-Objekts oder eines von Control abgeleiteten Objekts auf True gesetzt wird und damit das erste Mal die Notwendigkeit besteht, ein Window im Sinne vom Windows-Betriebssystem ins Leben zu rufen, legt die Control-Klasse eine Member-Variablen auf Basis der Klasse NativeWindow (etwa: »grundlegendes Windows«) an. NativeWindow wird dabei die eigene Instanz von Control übergeben – es gibt damit einen Zirkelverweis zwischen der Control-Instanz (unserer Schaltfläche Grafik öffnen, um beim Beispiel zu bleiben) und ihrem NativeWindow-Member. Zirkelverweis in diesem Zusammenhang bedeutet: Das Control-Objekt kann nicht nur auf die NativeWindow-Instanz zugreifen, sondern die NativeWindow-Instanz weiß auch, zu welchem Control-Objekt sie gehört.

NativeWindow ist deswegen so wichtig für Control, weil es die Schnittstelle zur Warteschlange bildet und zwar auf eine ganz raffinierte Art und Weise: NativeWindow selbst erstellt bei seiner Instanzierung ein Objekt der so genannten WindowClass-Klasse. Diese Klasse können Sie selbst nicht verwenden, sie steht nur dem Framework zur Verfügung und bildet eine Art Verwalter zwischen dem Windows-Subsystem und dem darüber liegenden .NET-Framework. Wenn eine WindowClass-Klasse instanziert

wird, dann zu dem Zweck, ein Window im Windows-Betriebssystem-Sinne zu erstellen. Dazu legt sie zunächst durch spezielle Betriebssystemaufrufe ein so genanntes *Window Handle*<sup>3</sup> an (jede Schaltfläche, jedes Fenster, jede Liste – ja sogar jeder Auswahlbereich *unterhalb* einer aufklappbaren Liste unter Windows ist, wie schon gesagt, ein Window im Betriebssystem-Sinne). Anschließend trägt sie sowohl dieses Window Handle als auch das NativeWindow-Objekt (das sie kennt, da es ihr als Konstruktorparameter übergeben wurde) in eine Tabelle ein. Auf diese Tabelle kann die Warteschlange Zugriff nehmen, die vom Anwendungsframework (oder, für Puristen, mit der Methode `Application.Run`) beim Start der Applikation eingerichtet worden ist.

Irgendwann bekommt die Warteschlange eine für sie bestimmte Nachricht, beispielsweise wenn der Anwender unseres Beispielprogramms auf die Schaltfläche *Grafik öffnen* klickt. Sie prüft nun, welches spezielle Window Handle in der Nachricht als Kennung gespeichert ist und durchforstet die Tabelle mit der Zuordnung Window Handle/NativeWindow-Instanz nach diesem. Auf diese Weise findet sie das entsprechende NativeWindow-Objekt, ruft dessen `Callback`<sup>4</sup>-Funktion auf und übergibt der Funktion dabei die Nachricht. Callback macht seinerseits wiederum ein so genanntes Message-Objekt daraus, das für die Nachrichtenverarbeitung in .NET verwendet wird, und ruft die `WndProc`-Routine seiner Instanz auf.

Nun wird es Zeit für eine weitere Information, die ich Ihnen bisher verschwiegen habe, um die bisherigen Zusammenhänge nicht zu undurchschaubar werden zu lassen. Control arbeitet nämlich eigentlich gar nicht mit einer Instanz von NativeWindow, sondern mit einer davon abgeleiteten Klasse namens `ControlNativeWindow`. `ControlNativeWindow` überschreibt die `WndProc`-Routine seiner Basisklasse `NativeWindow` und leitet damit den Aufruf zur `OnMessage`-Funktion um (das Programm befindet sich zu diesem Zeitpunkt immer noch in der `ControlNativeWindow`-Instanz). `OnMessage` greift anschließend auf die zuvor gespeicherte Instanz des Controls zurück und ruft schließlich die `WndProc`-Routine von Control auf – die Nachricht ist jetzt bei der richtigen Komponente angekommen.

Die Aufgabe von Control ist jetzt nur noch, die Nachricht zu filtern, und das daraus resultierende Ereignis auszulösen. Das bedeutet aber auch, dass `WndProc` die unterste Basis jeder Control-Instanz (und damit auch jedes Formulars) darstellt, sich in die Ereigniskette hineinzuhängen.

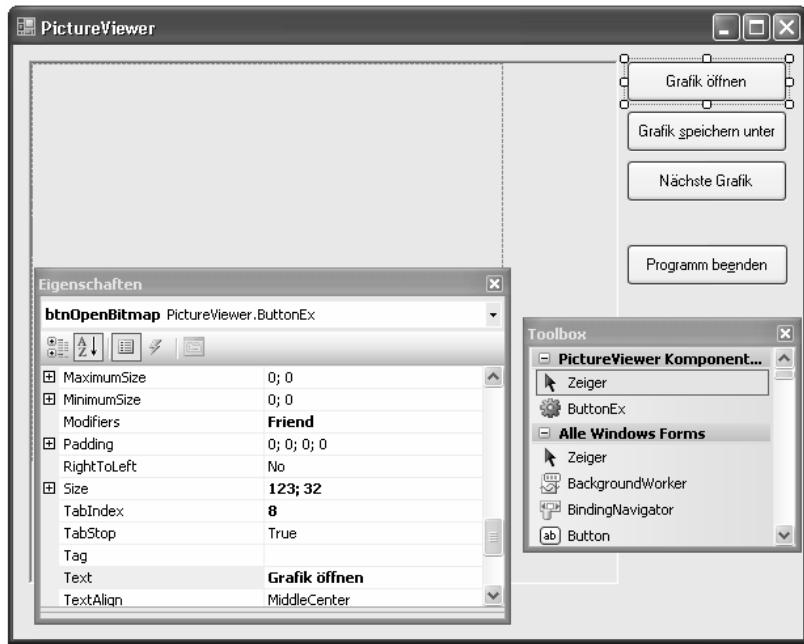
In unserem Beispiel war das Control die Schaltfläche *Grafik öffnen*. Dessen `Click`-Ereignis haben wir in unserem Programm eingebunden, mit einer entsprechenden Auswertungslogik versehen, und dass das Ergebnis tadellos funktioniert, können Sie sehen, wenn Sie das Programm starten und auf die Schaltfläche klicken.

Doch lassen Sie uns an dieser Stelle herausfinden, was es mit der Anzeige der Nachrichten im Ausgabefenster auf sich hat. Dazu werfen Sie einen Blick auf das Hauptformular des Beispielprojekts, und betrachten die *Grafik öffnen*-Schaltfläche ein wenig genauer im Designer (siehe Abbildung 28.5).

---

<sup>3</sup> Eine eindeutige ID, die jedes Windows-Element unter Windows zugewiesen bekommt, wenn es ins Leben gerufen wird.

<sup>4</sup> Es ist nicht nur eine Callback-Funktion im klassischen C-Sinne, die Funktion heißt tatsächlich so. Kleine Randnotiz: Wenn sich das .NET-Programm im Debug-Modus befindet, ruft sie eine andere Funktion namens `DebuggableCallback` auf.



**Abbildung 28.5:** Bei genauer Betrachtung wird deutlich, dass es sich bei der *Grafik öffnen*-Schaltfläche nicht um ein *Button*-, sondern um ein *ButtonEx*-Steuerelement handelt

Sie sehen, dass es sich bei der Schaltfläche im Formular gar nicht um eine »herkömmliche« Schaltfläche, sondern um eine Erweiterung handelt. Eine zusätzliche Codedatei namens *ButtonEx.vb* klärt, wieso sowohl Toolbox als auch Formular über eine eigentlich unbekannte Schaltfläche verfügen können:

```

Public Class ButtonEx
    Inherits Button

    Private Const WM_RBUTTONDOWN As Integer = &H204
    Private Const WM_RBUTTONUP As Integer = &H205
    Private myDownFlag As Boolean

    Public Event RightClick(ByVal Sender As Object, ByVal e As EventArgs)

    Protected Overrides Sub WndProc(ByRef m As System.Windows.Forms.Message)
        Debug.Print(m.ToString)
        If m.Msg = WM_RBUTTONDOWN Then
            myDownFlag = True
        End If
        If m.Msg = WM_RBUTTONUP And myDownFlag Then
            myDownFlag = False
            OnRightClick(Me, EventArgs.Empty)
        End If
        MyBase.WndProc(m)
    End Sub

```

```

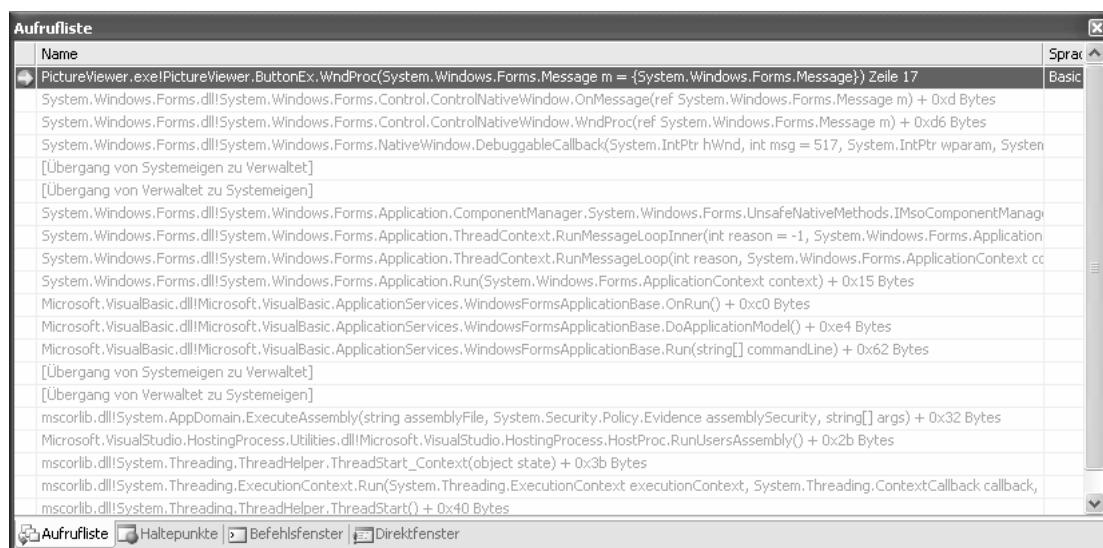
Protected Overridable Sub OnRightClick(ByVal Sender As Object, ByVal e As EventArgs)
    RaiseEvent RightClick(Sender, e)
End Sub
End Class

```

Das Erstellen einer Klassendatei mit diesem Namen und das Einfügen dieses Codes haben ausgereicht, um nach dem ersten Kompilieren das »neue« Steuerelement in der Toolbox anzeigen zu lassen als auch es sofort anschließend im Formular verwenden zu können.

Für ein weiteres Experiment setzen Sie nun in der im oben stehenden Klassencode in der fett ausgezeichneten Zeile mit der Funktionstaste **F9** einen Haltepunkt. Starten Sie das Programm anschließend und klicken Sie mit der *rechten* Maustaste auf die Schaltfläche *Grafik öffnen*.

Sie können im Ausgabefenster nun in aller Ruhe die Nachrichtenhistorie der Schaltfläche betrachten (siehe Abbildung 28.5). *hwnd* im Fenster bezeichnet übrigens das Window Handle, von dem in den vorherigen Erklärungen die Rede war und das die Warteschlangenroutine verwendet hat, um die .NET-Schaltfläche wieder zu finden. Der Debugger bietet Ihnen die Möglichkeit, die Aufruf-Hierarchie der aktuellen Funktion aufzulisten, in der das Programm gerade »steckt«. Um die Aufrufliste darzustellen, drücken Sie einfach die Tastenkombination **Strg+Alt+C** (alternativ wählen Sie aus dem Menü *Debuggen*, den Menüpunkt *Fenster* und weiter den Untermenüpunkt *Aufrufliste*).



**Abbildung 28.6:** Die Aufrufliste zeigt die Quelle des Funktionsaufrufs

Die Aufrufliste verifiziert die vorhin geschilderte Erklärung. Fast lückenlos sind die verschiedenen Funktionsaufrufe in der Kette zu verfolgen, die letzten Endes zum Ausführen der Routine *WndProc* des jeweiligen Steuerelements oder Formulars führen.

---

**HINWEIS:** In der Standardeinstellung von Visual Studio werden Aufrufe, die nicht von Ihrer Anwendung stammen, nicht im Ausgabefenster angezeigt. In diesem Fall öffnen Sie das Kontextmenü der Aufrufliste mit der rechten Maustaste und wählen den Eintrag *Nicht-benutzerseitigen Code anzeigen* aus.

---

# Implementieren neuer Steuerelementereignisse auf Basis von Warteschlangenauswertungen

Nun wissen Sie, wie Nachrichten des Windows-Betriebssystems in .NET-WinForms-Anwendungen verarbeitet werden. Dieses Wissen können Sie sich zunutze machen, um vorhandene Steuerelemente um eigene Ereignisse zu erweitern.

So könnte es für den Entwickler beispielsweise von Vorteil sein, ein RightClick-Ereignis zu erhalten, wenn der Anwender über der Schaltfläche die rechte Maustaste drückt.

Wenn ein Mausklickereignis ein bestimmtes Objekt erreichen soll, dann reicht es nicht aus, die Nachrichtenwarteschlange von Windows daraufhin abzufragen. Eine Klick-Nachricht gibt es nämlich in dieser Form überhaupt nicht. Ein Klick besteht aus einer direkten Folge aus Nachrichten vom Nachrichtentyp *WM\_LMOUSEDOWN*<sup>5</sup> und *WM\_LMOUSEUP* – jedenfalls soweit das die linke Maustaste betrifft.

In der Routine *WndProc*, die die Windows-Nachrichten schon gefiltert für die eigene Instanz der Klasse (*TestButton* im Beispiel) bekommt, sollte es deswegen eine Member-Variable geben, die sich merkt, ob die rechte Maustaste bereits gedrückt wurde. Dann kann der Auswertungsabschnitt für die Nachricht des Loslassens der rechten Maustaste beide Aktionen als »Mausklick rechts« interpretieren und das *RightClick*-Ereignis auslösen.

---

**HINWEIS:** Mir ist klar, dass es mehrere Möglichkeiten gibt, ein solches Ereignis zur Verfügung zu stellen. Alternativ zum Abfangen der Windows-Nachrichten in *WndProc* ließe sich auch *PreProcessMessage* überschreiben – eine Funktion, die den Vorteil hat, bereits fix und fertig aufbereitete Windows-Nachrichten im .NET-üblichen *Message*-Format zu empfangen. Auch das Überschreiben von *OnMouseDown* und *OnMouseUp* wäre denkbar – aber: Es geht an dieser Stelle mehr um die Demonstration der grundlegenden Verfahren als um die beste Form der Implementierung. Die schnellste ist die *WndProc*-Methode allemal, denn alle anderen Funktionen werden mehr oder weniger direkt aus *WndProc* heraus aufgerufen.

---

Der Beispielcode auf Seite 831 zeigt die komplette Implementierung, die letzten Endes zum Ausführen des Ereignisses führt. Mehr über die Grundlagen von Ereignissen und zum Auslösen von Ereignissen erfahren Sie übrigens in ► Kapitel 15.

Entfernen Sie nun den Haltepunkt, den Sie im vorherigen Abschnitt gesetzt haben, und starten Sie die Beispielanwendung erneut. Und nun beobachten Sie, was passiert, wenn Sie die Schaltflächen *Grafik öffnen* und *Nächste Grafik* mit der rechten Maustaste bedienen.

- Wenn Sie die Schaltfläche *Grafik öffnen* normal betätigen, wird standardmäßig jeweils das Verzeichnis in der Dateiauswahl angezeigt, das sie als Letztes verwendet haben. Klicken Sie auf diese Schaltfläche jedoch mit der *rechten* Maustaste, sehen Sie grundsätzlich die Verzeichnisauswahl Ihres *Eigene-Bilder*-Verzeichnisses.

---

<sup>5</sup> Die Bezeichnung von Ereignissen in Form von Konstanten hat eine lange Geschichte und reicht zurück bis zur Programmierung von Windows 2.11 unter C. WM\_ steht dabei natürlich für Windows Message. In .NET sind diese Konstanten leider nicht vordefiniert, aber spätere Beispiele zeigen, wie Sie an die entsprechenden Definitionen gelangen.

- Wenn Sie die Schaltfläche *Nächste Grafik* normal betätigen, wird automatisch das nächste Bild im Verzeichnis dargestellt. Betätigen Sie jedoch die *rechte* Maustaste, dann wird die erste Bilddatei des Verzeichnisses dargestellt.

## Wer oder was löst welche Formular- bzw. Steuerelementereignisse wann aus?

Es gibt eine Vielzahl von Ereignissen, die durch Benutzeraktionen bei Formularen (und den Schaltflächen, die sie beherbergen) ausgelöst werden können. Auch, wenn Sie sich schon eine Weile mit diesem Thema beschäftigt haben, bleibt es immer noch aufwändig herauszufinden, welche Aktion des Benutzers welches Ereignis wann auslöst.

Ich habe mir lange Gedanken darüber gemacht, was Ihnen beim Finden des richtigen Ereignisses und beim Verstehen der richtigen Zusammenhänge bei Ereignissen am besten helfen kann. Das Ergebnis ist das folgende Programm, das die Geheimnisse um Ereignisse sowohl für Formulare als auch für Komponenten auf seine Weise löst.

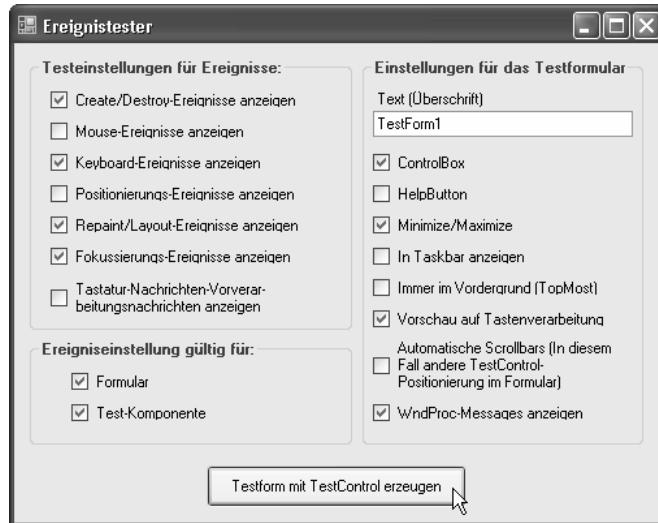
---

**BEGLEITDATEIEN:** Sie finden das Programm zu diesem Beispiel im Verzeichnis *\VB 2005 - Entwicklerbuch\G - Smart-Client\Kap28\Ereignistester*.

---

Dieses Programm soll gleich zweierlei Zwecke erfüllen. Auf der einen Seite soll es Ihnen als eine Art Nachschlagewerk dienen. Sämtliche Methoden, die es überschreibt, sind dokumentiert, und sie beschreiben sozusagen direkt vor Ort, welche Überschreibung welchen Zweck erfüllt. Wenn Sie dennoch nicht sicher sind, in welchem Zusammenhang die verschiedenen Ereignisse stehen, können Sie es auf der anderen Seite auch als Testprogramm verwenden, um mit den Ereignissen zu experimentieren. Die wichtigsten Ereignisse sind nämlich überschrieben, und wenn Sie das Programm starten, generiert es ein Testformular, das auch eine Testkomponente enthält. Durch Verschieben, Vergrößern, Verkleinern, Darüberfahren mit der Maus, Daraufklicken und das Ausführen anderer Aktionen sehen Sie im Ausgabefenster, welches Ereignis zu welcher Zeit aufgerufen wird.

Wenn Sie dieses Programm starten, sehen Sie zunächst einen Dialog, etwa wie in Abbildung 28.7 auf dem Bildschirm:



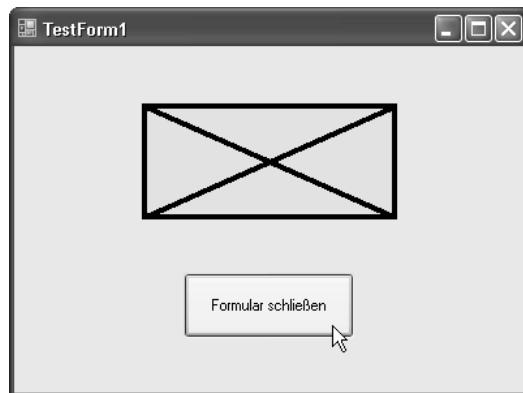
**Abbildung 28.7:** Im Hauptdialog der Anwendung nehmen Sie die Einstellungen für die Elemente und Ereignisse vor, die Sie nachverfolgen bzw. testen möchten

Dieser Dialog erlaubt Ihnen zu bestimmen, welche Ereigniskategorie im Ausgabefenster protokolliert werden soll. Die verschiedenen Ereignisse sind dabei in Gruppen eingeteilt. Durch die Kontrollkästchen im Bereich *Testeinstellungen für Ereignisse* wählen Sie die zu protokollierenden aus. Im darunter liegenden Bereich bestimmen Sie, ob die Ereignisse nur für das Formular, nur für die Testkomponente oder für beide ausgegeben werden sollen.

In der Rubrik *Einstellungen für das Testformular* bestimmen Sie, mit welchen Attributen Sie das Testformular ausstatten möchten. Diese Optionen repräsentieren die wichtigsten Eigenschaften, die Sie auch zur Entwurfszeit für ein Formular einstellen können.

Wählen Sie nun bitte die Einstellungen so aus, wie Sie sie auch in Abbildung 28.7 sehen können. Klicken Sie anschließend auf die Schaltfläche *Testform mit TestControl erzeugen*.

Sie sehen anschließend ein Formular mit einer wunderschönen, selbst gestrickten Komponente, wie Sie auch in Abbildung 28.8 zu sehen ist.



**Abbildung 28.8:** Mit diesem Testformular und seiner benutzerdefinierten Komponente können Sie Ihre Experimente durchführen

Und jetzt toben Sie sich aus! Bewegen Sie das Formular über den Bildschirm. Fahren Sie mit dem Mauszeiger über Formular und Komponenten. Vergrößern und verkleinern Sie das Formular. Wechseln Sie mit der Tabulator-Taste den Fokus der beiden Komponenten. Klicken Sie in das Formular. Halten Sie den Mausknopf über dem Formular gedrückt, und bewegen Sie dabei die Maus.

Wann immer Sie Aktionen durchführen, sehen Sie im Ausgabefenster eine entsprechende Kommentierung dazu, wie etwa in folgendem Protokollauszug zu sehen:

```
'FormEventChain.exe': 'c:\windows\assembly\gac\system.xml\1.0.5000.0_b77a5c561934e089\system.xml.dll'  
geladen, keine Symbole geladen.  
FormEventChain.frmTest, Text: frmTest: OnLayout: AffectedControl=FormEventChain.frmTest, Text: frmTest;  
AffectedProperty=Bounds  
FormEventChain.frmTest, Text: frmTest: OnLayout: AffectedControl=FormEventChain.frmTest, Text: frmTest;  
AffectedProperty=Bounds  
FormEventChain.frmTest, Text: frmTest: OnLayout: AffectedControl=; AffectedProperty=  
FormEventChain.TestControl: OnLayout: AffectedControl=FormEventChain.TestControl; AffectedProperty=Bounds  
FormEventChain.TestControl: OnLayout: AffectedControl=FormEventChain.TestControl; AffectedProperty=Bounds  
FormEventChain.frmTest, Text: TestForm1: OnLayout: AffectedControl=FormEventChain.TestControl;  
AffectedProperty=Parent  
FormEventChain.frmTest, Text: TestForm1: OnLayout: AffectedControl=System.Windows.Forms.Button, Text:  
Formular &schließen; AffectedProperty=Parent  
FormEventChain.frmTest, Text: TestForm1: OnHandleCreated  
FormEventChain.frmTest, Text: TestForm1: OnActivated  
FormEventChain.frmTest, Text: TestForm1: OnInvalidated: InvalidRect={X=0,Y=0,Width=390,Height=236}  
FormEventChain.TestControl: OnHandleCreated  
FormEventChain.TestControl: OnCreateControl  
FormEventChain.frmTest, Text: TestForm1: OnLoad  
FormEventChain.frmTest, Text: TestForm1: OnCreateControl  
FormEventChain.frmTest, Text: TestForm1: OnLayout: AffectedControl=; AffectedProperty=  
FormEventChain.frmTest, Text: TestForm1: OnPaintBackground:{X=0,Y=0,Width=390,Height=236}  
FormEventChain.TestControl: OnInvalidated: InvalidRect={X=0,Y=0,Width=195,Height=79}  
FormEventChain.frmTest, Text: TestForm1: SetVisibleCore: value=True  
FormEventChain.frmTest, Text: TestForm1: OnPaint:{X=0,Y=0,Width=390,Height=236}  
FormEventChain.TestControl: OnPaintBackground:{X=0,Y=0,Width=195,Height=79}  
FormEventChain.TestControl: OnPaint:{X=0,Y=0,Width=195,Height=79}  
FormEventChain.TestControl: OnInvalidated: InvalidRect={X=0,Y=0,Width=195,Height=79}  
FormEventChain.TestControl: OnPaintBackground:{X=0,Y=0,Width=195,Height=79}  
FormEventChain.TestControl: OnPaint:{X=0,Y=0,Width=195,Height=79}  
'FormEventChain.exe':  
'c:\windows\assembly\gac\microsoft.visualbasic\7.0.5000.0_b03f5f7f11d50a3a\microsoft.visualbasic.dll'  
geladen, keine Symbole geladen.  
FormEventChain.frmTest, Text: TestForm1: OnDeactivate  
FormEventChain.frmTest, Text: TestForm1: OnHandleDestroyed  
FormEventChain.TestControl: OnHandleDestroyed  
FormEventChain.TestControl: Dispose  
FormEventChain.frmTest, Text: : Dispose
```

Es ist interessant zu sehen, wie die einzelnen Ereignisse sich gegenseitig bedingen, finden Sie nicht?

Noch interessanter ist, welche Ereignisse mit welchen Prozeduren abgefangen werden können. Prinzipiell spielt es natürlich keine Rolle, ob Sie ein Ereignis als Event im Sinne eines .NET-Framework-Events behandeln oder, wenn es im Kontext möglich ist, eine entsprechende Prozedur durch Überschreiben verwenden. Zu diesem Zweck sollen Ihnen die folgenden Seiten dienen, die das kommentierte Listing enthalten. Es ist gemäß den Kategorien der Ereignisse in verschiedene Sektionen

unterteilt, die auch mit entsprechenden Überschriften versehen sind. Dadurch können Sie dieses Listing auch als Nachschlagewerk verwenden, wenn Sie später, beim Entwickeln Ihrer eigenen Komponenten und Anwendungen, schnell eine geeignete Ereignisroutine finden müssen.

---

**HINWEIS:** Die Definitionszeilen der einzelnen Prozeduren sind fett formatiert, damit Sie sie leichter im Listing erkennen können. Darüber hinaus sind die Prozeduren – soweit möglich – nach der Reihenfolge ihres Auftretens innerhalb der einzelnen Kategorien sortiert.

---

Auch wenn das Abdrucken beider Teile – der des Formulars und der der Control-Klasse – auf den ersten Blick doppelt und damit überflüssig erscheint: Formulare und Steuerelemente lösen in gleichen Situationen oft verschiedene Ereignisse aus – das ist der Grund.

Die Kommentare sind zu Gunsten der leichteren Lesbarkeit im folgenden Listung in normalen Fließtext umgewandelt worden. Im Code selbst finden Sie die Kommentare in gleichem Wortlaut als Kommentarzeilen.

## Kategorie Erstellen und Zerstören des Formulars

```
'*****  
''Erstellen, Aktivieren, Deaktivieren und Zerstören  
'*****
```

**OnHandleCreated:** Wird aufgerufen, nachdem das *Window-Handle* für die Formular-Instanz erstellt wurde. Ab diesem Zeitpunkt ist das Formular von der Nachrichtenwarteschlange erkennbar.

```
Protected Overrides Sub OnHandleCreated(ByVal e As System.EventArgs)  
    MyBase.OnHandleCreated(e)  
    If myShowCreationDestroy Then  
        Debug.WriteLine(Me.ToString + ": OnHandleCreated")  
    End If  
End Sub
```

**OnLoad:** Tritt ein, kurz bevor die Formular-Instanz das erste Mal sichtbar wird. Sie haben hier die Möglichkeit, Initialisierungen für das Formular vorzunehmen. Beachten Sie, dass das Fokussieren von Steuerelementen zu dieser Zeit noch nicht funktioniert und eine Ausnahme auslösen würde.

```
Protected Overrides Sub OnLoad(ByVal e As System.EventArgs)  
    MyBase.OnLoad(e)  
    If myShowCreationDestroy Then  
        Debug.WriteLine(Me.ToString + ": OnLoad")  
    End If  
End Sub
```

**OnCreateControl:** Tritt ein, nachdem die Framework-seitigen Ressourcen für das Formular erstellt wurden. Die Basisfunktion muss in den Framework-Versionen 1.0 und 1.1 nicht notwendigerweise aufgerufen werden; aus Aufwärtskompatibilitätsgründen sollte es aber dennoch passieren.

```
Protected Overrides Sub OnCreateControl()  
    MyBase.OnCreateControl()  
    If myShowCreationDestroy Then  
        Debug.WriteLine(Me.ToString + ": OnCreateControl")  
    End If  
End Sub
```

**OnActivated:** Wird aufgerufen, wenn das Formular aktiviert wurde. Bei einem Formular wird diese Funktion aufgerufen, wenn es zum zuoberst liegenden wird – entweder durch Benutzerklick auf das Formular, oder, da es das Hauptfenster der Anwendung ist, dadurch, dass die Anwendung gestartet oder aktiviert wurde.

```
Protected Overrides Sub OnActivated(ByVal e As System.EventArgs)
    MyBase.OnActivated(e)
    If myShowCreationDestroy Then
        Debug.WriteLine(Me.ToString + ": OnActivated")
    End If
End Sub
```

**SetVisibleCore:** Wird aufgerufen, um einer ableitenden Klasse beim Initialisierungsvorgang die Möglichkeit zu geben, die Sichtbarkeit (durch die Visible-Eigenschaft gesteuert) zu ändern. Eigentlich ist diese Routine kein richtiges Ereignis, sondern nur die ausführende Unterfunktion einer Eigenschaft.

```
Protected Overrides Sub SetVisibleCore(ByVal value As Boolean)
    MyBase.SetVisibleCore(value)
    If myShowCreationDestroy Then
        Debug.WriteLine(String.Format(Me.ToString + ": SetVisibleCore: value={0}", value))
    End If
End Sub
```

**OnClosing:** Wird aufgerufen, wenn der Schließen-Vorgang des Formulars beginnt. Sie können das Schließen verhindern, indem Sie die Cancel-Eigenschaft von e auf True setzen.

```
Protected Overrides Sub OnClosing(ByVal e As System.ComponentModel.CancelEventArgs)
    MyBase.OnClosing(e)
    If myShowCreationDestroy Then
        Debug.WriteLine(Me.ToString + ": OnClosing")
    End If
End Sub
```

**OnClosed:** Wird aufgerufen, wenn das Formular geschlossen wurde.

```
Protected Overrides Sub OnClosed(ByVal e As System.EventArgs)
    MyBase.OnClosed(e)
    If myShowCreationDestroy Then
        Debug.WriteLine(Me.ToString + ": OnClosed")
    End If
End Sub
```

**OnDeactivate:** Wird aufgerufen, wenn das Formular deaktiviert wurde.

```
Protected Overrides Sub OnDeactivate(ByVal e As System.EventArgs)
    MyBase.OnDeactivate(e)
    If myShowCreationDestroy Then
        Debug.WriteLine(Me.ToString + ": OnDeactivate")
    End If
End Sub
```

**OnHandleDestroyed:** Wird aufgerufen, wenn das *Window-Handle* des Formulars zerstört wurde.

```
Protected Overrides Sub OnHandleDestroyed(ByVal e As System.EventArgs)
    MyBase.OnHandleDestroyed(e)
    If myShowCreationDestroy Then
        Debug.WriteLine(Me.ToString + ": OnHandleDestroyed")
    End If
End Sub
```

**Dispose:** Das Formular überschreibt den Löschgong der Basisklasse, um Komponenten zu bereinigen. Diese Routine wird in der Regel durch den Formular-Designer implementiert.

```
Protected Overrides Sub Dispose(ByVal disposing As Boolean)
    If disposing Then
        If Not (components Is Nothing) Then
            components.Dispose()
        End If
    End If
    MyBase.Dispose(disposing)
    If myShowCreationDestroy Then
        Debug.WriteLine(Me.ToString + ": Dispose")
    End If
End Sub
```

## Kategorie Mausereignisse des Formulars

```
'*****
'Mausereignisse
'*****
```

**OnMouseDown:** Wird aufgerufen, wenn ein Mausbutton gedrückt wird und sich die Maus über einem Bereich des Formulars, aber nicht über einem *ChildWindow*-Bereich (andere Komponente) befindet.

```
Protected Overrides Sub OnMouseDown(ByVal e As System.Windows.Forms.MouseEventArgs)
    MyBase.OnMouseDown(e)
    If myShowMouse Then
        Debug.WriteLine(String.Format(
            Me.ToString + ": OnMouseDown: x={0}; y={1}; delta={2}; button={3}; clicks={4}" _
            , e.X, e.Y, e.Delta, e.Button, e.Clicks))
    End If
End Sub
```

**OnClick:** Wird aufgerufen, wenn ein Mausklick mit der linken Maustaste über einem Bereich des Formulars, aber nicht über einem *ChildWindow*-Bereich (andere Komponente) durchgeführt wird.

```
Protected Overrides Sub OnClick(ByVal e As System.EventArgs)
    MyBase.OnClick(e)
    If myShowMouse Then
        Debug.WriteLine(Me.ToString + ": OnClick")
    End If
End Sub
```

**OnDoubleClick:** Wird aufgerufen, wenn ein Doppelklick mit der linken Maustaste über einem Bereich des Formulars, aber nicht über einem *ChildWindow*-Bereich (andere Komponente) durchgeführt wird.

```
Protected Overrides Sub OnDoubleClick(ByVal e As System.EventArgs)
    MyBase.OnDoubleClick(e)
    If myShowMouse Then
        Debug.WriteLine(Me.ToString + ": OnDoubleClick")
    End If
End Sub
```

**OnMouseUp:** Wird aufgerufen, wenn ein Mausbutton losgelassen wird und sich die Maus über einem Bereich des Formulars, aber nicht über einem *ChildWindow*-Bereich (andere Komponente) befindet.

```
Protected Overrides Sub OnMouseUp(ByVal e As System.Windows.Forms.MouseEventArgs)
    MyBase.OnMouseUp(e)
    If myShowMouse Then
        Debug.WriteLine(String.Format(
            Me.ToString + ": OnMouseUp: x={0}; y={1}; delta={2}; button={3}; clicks={4}" +
            , e.X, e.Y, e.Delta, e.Button, e.Clicks))
    End If
End Sub
```

**OnMouseEnter:** Wird aufgerufen, wenn der Mauszeiger den Bereich des Formulars, aber nicht einen *ChildWindow*-Bereich (andere Komponente) betritt.

```
Protected Overrides Sub OnMouseEnter(ByVal e As System.EventArgs)
    MyBase.OnMouseEnter(e)
    If myShowMouse Then
        Debug.WriteLine(Me.ToString + ": OnMouseEnter:" + e.ToString)
    End If
End Sub
```

**OnMouseHover:** Wird aufgerufen, wenn der Mauszeiger das erste Mal nach dem Betreten des Formularbereichs zur Ruhe gekommen ist.

```
Protected Overrides Sub OnMouseHover(ByVal e As System.EventArgs)
    MyBase.OnMouseHover(e)
    If myShowMouse Then
        Debug.WriteLine(Me.ToString + ": OnMouseHover:" + e.ToString)
    End If
End Sub
```

**OnMouseMove:** Wird aufgerufen, wenn der Mauszeiger über dem Bereich des Formulars, aber nicht über einem *ChildWindow*-Bereich (andere Komponente) bewegt wird.

```
Protected Overrides Sub OnMouseMove(ByVal e As System.Windows.Forms.MouseEventArgs)
    MyBase.OnMouseMove(e)
    If myShowMouse Then
        Debug.WriteLine(String.Format(
            Me.ToString + ": OnMouseMove: x={0}; y={1}; delta={2}; button={3}; clicks={4}" -
            , e.X, e.Y, e.Delta, e.Button, e.Clicks))
    End If
End Sub
```

**OnMouseLeave:** Wird aufgerufen, wenn der Mauszeiger den Bereich des Formulars verlässt.

```
Protected Overrides Sub OnMouseLeave(ByVal e As System.EventArgs)
    MyBase.OnMouseLeave(e)
    If myShowMouse Then
        Debug.WriteLine(Me.ToString + ": OnMouseLeave:" + e.ToString)
    End If
End Sub
```

**OnMouseWheel:** Wird aufgerufen, wenn das Mausrad über dem Bereich des Formulars bewegt wird. Dieses Ereignis wird für alle untergeordneten Komponenten (*ChildWindows*) ebenfalls ausgelöst!

```
Protected Overrides Sub OnMouseWheel(ByVal e As System.Windows.Forms.MouseEventArgs)
    MyBase.OnMouseWheel(e)
    If myShowMouse Then
        Debug.WriteLine(String.Format(
            Me.ToString + ": OnMouseWheel: x={0}; y={1}; delta={2}; button={3}; clicks={4}" -
            , e.X, e.Y, e.Delta, e.Button, e.Clicks))
    End If
End Sub
```

## Kategorie Tastaturereignisse des Formulars

```
'*****
'Tastatur
'*****
```

**OnKeyDown:** Wird aufgerufen, wenn eine Taste gedrückt wird. Wird allerdings nicht aufgerufen, wenn es eine weitere, fokussierte Komponente im Formular gibt und die KeyPreview-Eigenschaft auf False gesetzt wurde bzw. die überschriebene ProcessKeyPreview-Methode (s.u.) das Ereignis schon verarbeitet hat.

```
Protected Overrides Sub OnKeyDown(ByVal e As System.Windows.Forms.KeyEventArgs)
    MyBase.OnKeyDown(e)
    If myShowKeyboard Then
        Debug.WriteLine(String.Format(
            Me.ToString + ": OnKeyDown: KeyCode={0}; KeyData={1}; KeyValue={2}; Modifiers={3}" -
            , e.KeyCode, e.KeyData, e.KeyValue, e.Modifiers))
    End If
End Sub
```

**OnKeyPress:** Wird aufgerufen, wenn eine Taste gedrückt wird; wird nicht aufgerufen, wenn eine Steuerungstaste (wie **Strg** oder **Shift**) alleine oder in Kombination mit einer anderen gedrückt wird. Diese Prozedur wird auch dann nicht aufgerufen, wenn es eine weitere, fokussierte Komponente im Formular gibt und die KeyPreview-Eigenschaft auf **False** gesetzt wurde bzw. die überschriebene ProcessKeyPreview-Methode (s.u.) das Ereignis schon verarbeitet hat.

```
Protected Overrides Sub OnKeyPress(ByVal e As System.Windows.Forms.KeyPressEventArgs)
    MyBase.OnKeyPress(e)
    If myShowKeyboard Then
        Debug.WriteLine(String.Format(
            Me.ToString + ": OnKeyPress: KeyChar={0}, _ 
            e.KeyChar))
    End If
End Sub
```

**OnKeyUp:** Wird aufgerufen, wenn eine Taste wieder losgelassen wird. Diese Prozedur wird nicht aufgerufen, wenn es eine weitere, fokussierte Komponente im Formular gibt und die KeyPreview-Eigenschaft auf **False** gesetzt wurde bzw. die überschriebene ProcessKeyPreview-Methode (s.u.) das Ereignis schon verarbeitet hat.

```
Protected Overrides Sub OnKeyUp(ByVal e As System.Windows.Forms.KeyEventArgs)
    MyBase.OnKeyUp(e)
    If myShowKeyboard Then
        Debug.WriteLine(String.Format(
            Me.ToString + ": OnKeyUp: KeyCode={0}; KeyData={1}; KeyValue={2}; Modifiers={3}, _ 
            e.KeyCode, e.KeyData, e.KeyValue, e.Modifiers))
    End If
End Sub
```

## Kategorie Position und Größe des Formulars

```
'*****
'* Größe und Position
'*****
```

**OnMove:** Wird aufgerufen, wenn die Formularposition verändert wird. Diese Methode wird kontinuierlich aufgerufen, während der Anwender das Formular verschiebt und die Anzeigeneinstellung so vorgenommen wurde, dass der Fensterinhalt beim Ziehen mit verschoben wird.

```
Protected Overrides Sub OnMove(ByVal e As System.EventArgs)
    MyBase.OnMove(e)
    If myShowPositioning Then
        Debug.WriteLine(Me.ToString + ": OnMove:" + e.ToString)
    End If
End Sub
```

**OnLocationChanged:** Wird aufgerufen, wenn sich die Position des Formulars verändert hat. Diese Methode wird leider ebenfalls kontinuierlich aufgerufen, wenn die Anzeigeneinstellung so vorgenommen wurde, dass der Fensterinhalt beim Ziehen mit verschoben wird, sodass ein Ende der Verschiebeaktion hiermit nicht festgestellt werden kann. Um das zu erreichen, müssten Sie **WndProc** überschreiben und die empfangene Nachricht dort auf **WM\_EXITSIZEMOVE** überprüfen. Ein Beispiel dazu finden Sie in ► Kapitel 29.

```

Protected Overrides Sub OnLocationChanged(ByVal e As System.EventArgs)
    MyBase.OnLocationChanged(e)
    If myShowPositioning Then
        Debug.WriteLine(Me.ToString + ": OnLocationChanged:" + e.ToString)
    End If
End Sub

```

**OnResize:** Wird aufgerufen, wenn die Formulargröße verändert wird. Diese Methode wird kontinuierlich aufgerufen, während der Anwender das Formular vergrößert oder verkleinert und die Anzeigeneinstellung so vorgenommen wurde, dass der Fensterinhalt beim Ziehen mit verschoben wird.

```

Protected Overrides Sub OnResize(ByVal e As System.EventArgs)
    MyBase.OnResize(e)
    If myShowPositioning Then
        Debug.WriteLine(Me.ToString + ": OnResize:" + e.ToString)
    End If
End Sub

```

**OnSizeChanged:** Wird aufgerufen, wenn sich die Größe des Formulars verändert hat. Diese Methode wird leider ebenfalls kontinuierlich aufgerufen, wenn die Anzeigeneinstellung so vorgenommen wurde, dass der Fensterinhalt beim Ziehen mit verschoben wird, sodass ein Abschluss der Größenänderung hiermit nicht festgestellt werden kann. Um das zu erreichen, müssten Sie WndProc überschreiben und die empfangene Nachricht dort auf den Wert WM\_EXITSIZEMOVE überprüfen. Ein Beispiel dazu finden Sie in ► Kapitel 29.

```

Protected Overrides Sub OnSizeChanged(ByVal e As System.EventArgs)
    MyBase.OnSizeChanged(e)
    If myShowPositioning Then
        Debug.WriteLine(Me.ToString + ": OnSizeChanged:" + e.ToString)
    End If
End Sub

```

## Kategorie Anordnen der Komponenten und Neuzeichnen des Formulars

```

' ****
' Anordnen und Neuzeichnen
' ****

```

**OnInvalidated:** Wird aufgerufen, wenn eine Entität das Neuzeichnen des Formularinhalts mit Invalidate anfordert. Invalidate sollte in der Regel von OnResize aufgerufen werden, wenn der Inhalt des Fensters in Abhängigkeit von der Fenstergröße komplett neu gezeichnet werden muss. Ausgenommen sind Änderungen am Verhalten durch SetStyle (► Kapitel 29).

```

Protected Overrides Sub OnInvalidated(ByVal e As System.Windows.Forms.InvalidateEventArgs)
    MyBase.OnInvalidated(e)
    If myShowRepaintAndLayout Then
        Debug.WriteLine(String.Format(
            Me.ToString + ": OnInvalidated: InvalidRect={0}",
            e.InvalidRect))
    End If
End Sub

```

**OnLayout:** Wird aufgerufen, wenn das Formular anzeigt, dass seine beinhaltenden Steuerelemente aus irgendwelchen Gründen neu angeordnet werden müssen.

**HINWEIS:** Für Änderungen an einem Steuerelement, z. B. Größenänderungen, Ein- oder Ausblenden sowie Hinzufügen oder Entfernen untergeordneter Steuerelemente ist es notwendig, dass das Layout der untergeordneten Steuerelemente vom Steuerelement festgelegt wird. Der diesem Ereignis mitgegebene Parameter LayoutEventArgs gibt das geänderte untergeordnete Steuerelement und die davon betroffene Eigenschaft an. Wenn z. B. ein Steuerelement seit dem letzten Layoutvorgang sichtbar gemacht wurde, ist davon die Visible-Eigenschaft betroffen. Die AffectedControl- und AffectedProperty-Eigenschaften werden auf Nothing festgelegt, wenn beim Aufruf der PerformLayout-Methode keine Werte bereitgestellt wurden. Dieses Ereignis erfolgt nicht, wenn das Formular das Layout-Ereignis mit SuspendLayout außer Kraft gesetzt hat.

```
Protected Overrides Sub OnLayout(ByVal levent As System.Windows.Forms.LayoutEventArgs)
    MyBase.OnLayout(levent)
    If myShowRepaintAndLayout Then
        Debug.WriteLine(String.Format(
            Me.ToString() + ": OnLayout: AffectedControl={0}; AffectedProperty={1}",
            levent.AffectedControl, levent.AffectedProperty))
    End If
End Sub
```

**OnPaintBackground:** Wird aufgerufen, wenn der Hintergrund des Formulars neu gezeichnet werden muss. Das Graphics-Objekt, das mit dem Parameter vom Typ PaintEventArgs dem Ereignis übergeben wird, ist ausschließlich auf den Bereich geclipped, der neu gezeichnet werden muss. Wenn durch die Vergrößerung des Fensters der Fensterinhalt komplett neu gezeichnet werden muss, dann sollte OnResize bzw. das Resize-Ereignis die Methode Invalidate aufrufen, damit den Paint-Ereignissen ein ungeclippter Bereich für das Neuzeichnen des kompletten Inhalts übergeben wird. Beispiele dafür gibt es auch in ► Kapitel 29.

```
Protected Overrides Sub OnPaintBackground(ByVal pevent As System.Windows.Forms.PaintEventArgs)
    MyBase.OnPaintBackground(pevent)
    If myShowRepaintAndLayout Then
        Debug.WriteLine(Me.ToString() + ": OnPaintBackground:" + pevent.ClipRectangle.ToString())
    End If
End Sub
```

**OnPaint:** Wird aufgerufen, wenn der Fensterinhalt neu gezeichnet werden muss. Das Graphics-Objekt, das mit dem Parameter vom Typ PaintEventArgs dem Ereignis übergeben wird, ist ausschließlich auf den Bereich geclipped, der neu gezeichnet werden muss. Wenn durch die Vergrößerung des Fensters der Fensterinhalt komplett neu gezeichnet werden muss, dann sollte OnResize bzw. das Resize-Ereignis die Methode Invalidate aufrufen, damit den Paint-Ereignissen ein ungeclippter Bereich für das Neuzeichnen des kompletten Inhalts übergeben wird. Beispiele dafür gibt es auch in ► Kapitel 29.

```
Protected Overrides Sub OnPaint(ByVal e As System.Windows.Forms.PaintEventArgs)
    MyBase.OnPaint(e)
    If myShowRepaintAndLayout Then
        Debug.WriteLine(Me.ToString() + ": OnPaint:" + e.ClipRectangle.ToString())
    End If
End Sub
```

## Kategorie Fokussierung des Formulars

```
'*****
'Fokussierung
*****
```

**OnEnter:** Wird aufgerufen, wenn das Formular aktiviert wird, aber nur, wenn es mindestens eine weitere Komponente beinhaltet, die den Fokus beim Aktivieren bekommen kann.

```
Protected Overrides Sub OnEnter(ByVal e As System.EventArgs)
    MyBase.OnEnter(e)
    If myShowFocussing Then
        Debug.WriteLine(Me.ToString + ": OnEnter:" + e.ToString)
    End If
End Sub
```

**OnGotFocus:** Wird aufgerufen, wenn das Formular aktiviert wird, aber nur, wenn es kein weiteres Steuerelement beinhaltet, das den Fokus beim Aktivieren bekommen könnte.

```
Protected Overrides Sub OnGotFocus(ByVal e As System.EventArgs)
    MyBase.OnLostFocus(e)
    If myShowFocussing Then
        Debug.WriteLine(Me.ToString + ": OnGotFocus:" + e.ToString)
    End If
End Sub
```

**OnLostFocus:** Wird aufgerufen, wenn das Formular deaktiviert wird (zum Beispiel, weil ein anderes Fenster in den Vordergrund geklickt wurde), aber nur, wenn es keine weitere Komponente beinhaltet, die den Fokus beim Deaktivieren verlieren könnte.

```
Protected Overrides Sub OnLostFocus(ByVal e As System.EventArgs)
    MyBase.OnLostFocus(e)
    If myShowFocussing Then
        Debug.WriteLine(Me.ToString + ": OnLostFocus:" + e.ToString)
    End If
End Sub
```

**OnLeave:** Wird aufgerufen, wenn das Formular deaktiviert wird, aber nur, wenn es mindestens eine weitere Komponente beinhaltet, die den Fokus beim Deaktivieren verlieren kann.

```
Protected Overrides Sub OnLeave(ByVal e As System.EventArgs)
    MyBase.OnLeave(e)
    If myShowFocussing Then
        Debug.WriteLine(Me.ToString + ": OnLeave:" + e.ToString)
    End If
End Sub
```

## Kategorie Tastaturvorverarbeitungsnachrichten des Formulars

```
*****  
'Nachrichtenverarbeitung  
*****
```

**ProcessCmdKey:** Wird ausgelöst, wenn eine Befehlstaste (z.B. **ALT+Anfangsbuchstabe**) gedrückt wurde.

```
Protected Overrides Function ProcessCmdKey(ByRef msg As System.Windows.Forms.Message, _  
                                         ByVal keyData As System.Windows.Forms.Keys) As Boolean  
    If myShowPreProcessing Then  
        Debug.WriteLine(Me.ToString + ": ProcessCmdKey:" +  
                        msg.ToString + ": KeyData: " + keyData.ToString)  
    End If  
    'Wenn Ihre Instanz eine Nachricht verarbeitet hat, dann geben Sie  
    'True als Funktionsergebnis zurück, sonst False. Die Basis rufen Sie nur  
    'auf (dann aber auf jeden Fall!), wenn die Nachricht NICHT verarbeitet wurde.  
    Return MyBase.ProcessCmdKey(msg, keyData)  
End Function
```

**ProcessDialogChar:** Wird ausgelöst, wenn eine Dialogtaste (auch Steuerungstaste) gedrückt wurde.

```
Protected Overrides Function ProcessDialogChar(ByVal charCode As Char) As Boolean  
    If myShowPreProcessing Then  
        Debug.WriteLine(Me.ToString + ": ProcessDialogChar: " + charCode)  
    End If  
    'Wenn Ihre Instanz eine Nachricht verarbeitet hat, dann geben Sie  
    'True als Funktionsergebnis zurück, sonst False. Die Basis rufen Sie nur  
    'auf (dann aber auf jeden Fall!), wenn die Nachricht NICHT verarbeitet wurde.  
    Return MyBase.ProcessDialogChar(charCode)  
End Function
```

**ProcessDialogKey:** Wird ausgelöst, wenn eine Dialog-Taste (aber keine Steuerungstaste) gedrückt wurde.

```
Protected Overrides Function ProcessDialogKey(ByVal keyData As System.Windows.Forms.Keys) As Boolean  
    If myShowPreProcessing Then  
        Debug.WriteLine(Me.ToString + ": ProcessDialogKey:" +  
                        ": KeyData: " + keyData.ToString)  
    End If  
    'Wenn Ihre Instanz eine Nachricht verarbeitet hat, dann geben Sie  
    'True als Funktionsergebnis zurück, sonst False. Die Basis rufen Sie nur  
    'auf (dann aber auf jeden Fall!), wenn die Nachricht NICHT verarbeitet wurde.  
    Return MyBase.ProcessDialogKey(keyData)  
End Function
```

**ProcessKeyPreview:** Wird bei jedem Tastenereignis des Formulars ausgelöst und regelt bei Formularen, ob in Abhängigkeit der KeyPreview-Eigenschaft Tastatur-Ereignis-Prozeduren aufgerufen werden.

```
Protected Overrides Function ProcessKeyPreview(ByRef m As System.Windows.Forms.Message) As Boolean
    If myShowPreProcessing Then
        Debug.WriteLine(Me.ToString + ": ProcessKeyPreview:" + m.ToString)
    End If
    'Wenn Ihre Instanz eine Nachricht verarbeitet hat, dann geben Sie
    'True als Funktionsergebnis zurück, sonst False. Die Basis rufen Sie nur
    'auf (dann aber auf jeden Fall!), wenn die Nachricht NICHT verarbeitet wurde
    Return False
End Function
```

**WndProc:** Wird bei jeder Nachricht aufgerufen, die das Fenster in irgendeiner Form betrifft. Um erweiterte Ereignisse selbst auszulösen, überschreiben Sie diese Prozedur. Rufen Sie die Basisfunktion im Anschluss nur dann auf, wenn Sie möchten, dass die Nachrichten, die Sie bereits verarbeitet haben, von der Basisklasse auch verarbeitet werden sollen.

```
Protected Overrides Sub WndProc(ByRef m As System.Windows.Forms.Message)
    If myShowWndProcMessages Then
        Console.WriteLine(m)
    End If
    MyBase.WndProc(m)
End Sub
```

## Kategorie Erstellen/Zerstören des Controls (des Steuerelements)

**OnHandleCreated:** Wird aufgerufen, nachdem das *Window-Handle* für die Steuerelement-Instanz erstellt wurde.

```
Protected Overrides Sub OnHandleCreated(ByVal e As System.EventArgs)
    MyBase.OnHandleCreated(e)
    If myShowCreationDestroy Then
        Debug.WriteLine(Me.ToString + ": OnHandleCreated")
    End If
End Sub
```

**OnCreateControl:** Tritt ein, nachdem die Framework-seitigen Ressourcen für das Steuerelement erstellt wurden. Die Basisfunktion muss in den Framework-Versionen 1.0 und 1.1 nicht notwendigerweise aufgerufen werden; aus Aufwärtskompatibilitätsgründen sollte das aber dennoch passieren.

```
Protected Overrides Sub OnCreateControl()
    MyBase.OnCreateControl()
    If myShowCreationDestroy Then
        Debug.WriteLine(Me.ToString + ": OnCreateControl")
    End If
End Sub
```

**OnHandleDestroyed:** Wird aufgerufen, wenn das *Window-Handle* zerstört wurde.

```
Protected Overrides Sub OnHandleDestroyed(ByVal e As System.EventArgs)
    MyBase.OnHandleDestroyed(e)
    If myShowCreationDestroy Then
        Debug.WriteLine(Me.ToString + ": OnHandleDestroyed")
    End If
End Sub
```

**Dispose:** Wird aufgerufen, wenn das Steuerelement entweder durch den Garbage Collector oder durch Dispose des einbindenden *Controls/Formulars* entsorgt wird.

```
Protected Overrides Sub Dispose(ByVal disposing As Boolean)
    MyBase.Dispose(disposing)
    If myShowCreationDestroy Then
        Debug.WriteLine(Me.ToString + ": Dispose")
    End If
End Sub
```

## Kategorie Mausereignisse des Controls

```
*****
'Mausereignisse
*****
```

**OnMouseDown:** Wird aufgerufen, wenn ein Mausbutton gedrückt wird und sich die Maus über einem Bereich des *Controls*, aber nicht über einem *ChildWindow-Bereich* (andere Komponente) befindet.

```
Protected Overrides Sub OnMouseDown(ByVal e As System.Windows.Forms.MouseEventArgs)
    MyBase.OnMouseDown(e)
    If myShowMouse Then
        Debug.WriteLine(String.Format(
            Me.ToString + ": OnMouseDown: x={0}; y={1}; delta={2}; button={3}; clicks={4}" _
            , e.X, e.Y, e.Delta, e.Button, e.Clicks))
    End If
End Sub
```

**OnClick:** Wird aufgerufen, wenn ein Mausklick mit der linken Maustaste über einem Bereich des *Controls*, aber nicht über einem *ChildWindow-Bereich* (andere Komponente) durchgeführt wird.

```
Protected Overrides Sub OnClick(ByVal e As System.EventArgs)
    MyBase.OnClick(e)
    If myShowMouse Then
        Debug.WriteLine(Me.ToString + ": OnClick")
    End If
End Sub
```

**OnDoubleClick:** Wird aufgerufen, wenn ein Doppelklick mit der linken Maustaste über einem Bereich des *Controls*, aber nicht über einem *ChildWindow*-Bereich (andere Komponente) durchgeführt wird.

```
Protected Overrides Sub OnDoubleClick(ByVal e As System.EventArgs)
    MyBase.OnDoubleClick(e)
    If myShowMouse Then
        Debug.WriteLine(Me.ToString + ": OnDoubleClick")
    End If
End Sub
```

**OnMouseUp:** Wird aufgerufen, wenn ein Mausbutton losgelassen wird und sich die Maus über einem Bereich des *Controls*, aber nicht über einem *ChildWindow*-Bereich (andere Komponente) befindet.

```
Protected Overrides Sub OnMouseUp(ByVal e As System.Windows.Forms.MouseEventArgs)
    MyBase.OnMouseUp(e)
    If myShowMouse Then
        Debug.WriteLine(String.Format(
            Me.ToString + ": OnMouseUp: x={0}; y={1}; delta={2}; button={3}; clicks={4}" +
            , e.X, e.Y, e.Delta, e.Button, e.Clicks))
    End If
End Sub
```

**OnMouseEnter:** Wird aufgerufen, wenn der Mauszeiger den Bereich des *Controls*, aber nicht einen *ChildWindow*-Bereich (andere Komponente) betritt.

```
Protected Overrides Sub OnMouseEnter(ByVal e As System.EventArgs)
    MyBase.OnMouseEnter(e)
    If myShowMouse Then
        Debug.WriteLine(Me.ToString + ": OnMouseEnter:" + e.ToString)
    End If
End Sub
```

**OnMouseHover:** Wird aufgerufen, wenn der Mauszeiger das erste Mal nach dem Betreten des Steuerelement-Bereichs zur Ruhe gekommen ist.

```
Protected Overrides Sub OnMouseHover(ByVal e As System.EventArgs)
    MyBase.OnMouseHover(e)
    If myShowMouse Then
        Debug.WriteLine(Me.ToString + ": OnMouseHover:" + e.ToString)
    End If
End Sub
```

**OnMouseMove:** Wird aufgerufen, wenn der Mauszeiger über dem Bereich des *Controls*, aber nicht über einem *ChildWindow*-Bereich (andere Komponente) bewegt wird.

```
Protected Overrides Sub OnMouseMove(ByVal e As System.Windows.Forms.MouseEventArgs)
    MyBase.OnMouseMove(e)
    If myShowMouse Then
        Debug.WriteLine(String.Format(
            Me.ToString + ": OnMouseMove: x={0}; y={1}; delta={2}; button={3}; clicks={4}" -
            , e.X, e.Y, e.Delta, e.Button, e.Clicks))
    End If
End Sub
```

**OnMouseLeave:** Wird aufgerufen, wenn der Mauszeiger den Bereich des *Controls* verlässt.

```
Protected Overrides Sub OnMouseLeave(ByVal e As System.EventArgs)
    MyBase.OnMouseLeave(e)
    If myShowMouse Then
        Debug.WriteLine(Me.ToString + ": OnMouseLeave:" + e.ToString)
    End If
End Sub
```

**OnMouseWheel:** Wird aufgerufen, wenn das Mausrad bewegt wird. Wichtig: Alle Komponenten des Formulars empfangen dieses Ereignis!

```
Protected Overrides Sub OnMouseWheel(ByVal e As System.Windows.Forms.MouseEventArgs)
    MyBase.OnMouseWheel(e)
    If myShowMouse Then
        Debug.WriteLine(String.Format(
            Me.ToString + ": OnMouseWheel: x={0}; y={1}; delta={2}; button={3}; clicks={4}" -
            , e.X, e.Y, e.Delta, e.Button, e.Clicks))
    End If
End Sub
```

## Kategorie Tastaturereignisse des Controls

```
*****
'Tastatur
*****
```

**OnKeyDown:** Wird aufgerufen, wenn eine Taste gedrückt wird und das Steuerelement den Fokus hat. Fungiert das Steuerelement als Container (von Scrollable- oder ContainerControl abgeleitet), verwenden Sie die Tastaturvorverarbeitungsnachrichten-Ereignisse, um die Tastaturereignisse auszuwerten, da sie selbst in diesem Fall nicht ausgelöst werden.

```
Protected Overrides Sub OnKeyDown(ByVal e As System.Windows.Forms.KeyEventArgs)
    MyBase.OnKeyDown(e)
    If myShowKeyboard Then
        Debug.WriteLine(String.Format(
            Me.ToString + ": OnkeyDown: KeyCode={0}; KeyData={1}; KeyValue={2}; Modifiers={3}" ,
            e.KeyCode, e.KeyData, e.KeyValue, e.Modifiers))
    End If
End Sub
```

**OnKeyPress:** Wird aufgerufen, wenn eine Taste gedrückt wird und das Steuerelement fokussiert ist; wird nicht aufgerufen, wenn eine Steuerungstaste (wie **Strg** oder **Shift**) alleine oder in Kombination mit einer anderen gedrückt wird. Fungiert das Steuerelement als Container (von Scrollable- oder ContainerControl abgeleitet), verwenden Sie die Tastaturvorverarbeitungsnachrichten-Ereignisse, um die Tastatureignisse auszuwerten, da sie selbst in diesem Fall nicht ausgelöst werden.

```
Protected Overrides Sub OnKeyPress(ByVal e As System.Windows.Forms.KeyPressEventArgs)
    MyBase.OnKeyPress(e)
    If myShowKeyboard Then
        Debug.WriteLine(String.Format(
            Me.ToString + ": OnKeyPress: KeyChar={0}", _
            e.KeyChar))
    End If
End Sub
```

**OnKeyUp:** Wird ausgerufen, wenn eine Taste wieder losgelassen wird und das Steuerelement den Fokus hat. Fungiert das Steuerelement als Container (von Scrollable- oder ContainerControl abgeleitet), verwenden Sie die Tastaturvorverarbeitungsnachrichten-Ereignisse, um die Tastatureignisse auszuwerten, da sie selbst in diesem Fall nicht ausgelöst werden.

```
Protected Overrides Sub OnKeyUp(ByVal e As System.Windows.Forms.KeyEventArgs)
    MyBase.OnKeyUp(e)
    If myShowKeyboard Then
        Debug.WriteLine(String.Format(
            Me.ToString + ": OnKeyUp: KeyCode={0}; KeyData={1}; KeyValue={2}; Modifiers={3}", _
            e.KeyCode, e.KeyData, e.KeyValue, e.Modifiers))
    End If
End Sub
```

## Kategorie Größe und Position des Controls

```
*****  
'Größe und Position  
*****
```

**OnMove:** Wird aufgerufen, wenn die Steuerelement-Position verändert wird.

```
Protected Overrides Sub OnMove(ByVal e As System.EventArgs)
    MyBase.OnMove(e)
    If myShowPositioning Then
        Debug.WriteLine(Me.ToString + ": OnMove:" + e.ToString)
    End If
End Sub
```

**OnLocationChanged:** Wird aufgerufen, wenn sich die Position des *Controls* verändert hat.

```
Protected Overrides Sub OnLocationChanged(ByVal e As System.EventArgs)
    MyBase.OnLocationChanged(e)
    If myShowPositioning Then
        Debug.WriteLine(Me.ToString + ": OnLocationChanged:" + e.ToString)
    End If
End Sub
```

**OnResize:** Wird aufgerufen, wenn sich die Ausmaße des *Controls* ändern. Wenn das Steuerelement seinen Inhalt in Abhängigkeit seiner Größe verändert, sollte an dieser Stelle ein Aufruf an Invalidate erfolgen, damit das parallel automatisch ausgelöste Paint-Ereignis (nur beim Vergrößern) verhindert wird und stattdessen ein neues Paint-Ereignis ausgelöst wird, das dann aber in der Lage ist, den Neuaufbau des *gesamten* Client-Bereichs durchzuführen. Hintergrund: Das Standard-Paint-Ereignis kann nur die neu zu zeichnenden Bereiche verarbeiten und wird gar nicht ausgelöst, wenn das Steuerelement nur verkleinert wird (siehe auch ► Kapitel 29 und Kapitel 30).

```
Protected Overrides Sub OnResize(ByVal e As System.EventArgs)
    MyBase.OnResize(e)
    If myShowPositioning Then
        Debug.WriteLine(Me.ToString() + ": OnResize:" + e.ToString())
    End If
    Invalidate()
End Sub
```

**OnSizeChanged:** Wird aufgerufen, wenn sich die Ausmaße eines *Controls* geändert haben (siehe auch ► Kapitel 29 und Kapitel 30).

```
Protected Overrides Sub OnSizeChanged(ByVal e As System.EventArgs)
    MyBase.OnSizeChanged(e)
    If myShowPositioning Then
        Debug.WriteLine(Me.ToString() + ": OnSizeChanged:" + e.ToString())
    End If
End Sub
```

**SetBoundsCore:** Diese Prozedur dient zweierlei Dingen: Zum einen wird sie von der Basisklasse bei jedem Ereignis aufgerufen, das durch das Ändern der Größe oder der Position des *Controls* aufgerufen wird. Klassen, die diese Routine überschreiben, können die Position und Ausmaße des *Controls* durch das Verändern der Parameter auf der anderen Seite reglementieren. Wenn Sie also beispielsweise nicht wollen, dass die Ausmaße des *Controls* eine bestimmte Größe überschreiten, definieren Sie in dieser Funktion für den entsprechenden Parameter einen neuen Wert, bevor Sie die Basisfunktion mit den geänderten Werten aufrufen. Der Parameter BoundsSpecified informiert Sie darüber, welcher Parameter durch ein Ereignis geändert wurde. Ein richtiges Beispiel dafür finden Sie in ► Kapitel 30.

```
Protected Overrides Sub SetBoundsCore(ByVal x As Integer, ByVal y As Integer, ByVal width As Integer, _
    ByVal height As Integer, ByVal specified As System.Windows.Forms.BoundsSpecified)
    MyBase.SetBoundsCore(x, y, width, height, specified)
    If myShowPositioning Then
        Debug.WriteLine(String.Format(
            Me.ToString() + ": SetBoundsCore: X={0}; y={1}; width={2}; height={3}; specified={4}" _
            , x, y, width, height, specified))
    End If
End Sub
```

**SetClientSize:** Wird aufgerufen, wenn sich die Größe des *Controls* durch das Setzen der ClientSize-Eigenschaft ändern soll.

```
Protected Overrides Sub SetClientSizeCore(ByVal x As Integer, ByVal y As Integer)
    MyBase.SetClientSizeCore(x, y)
    If myShowPositioning Then
        Debug.WriteLine(String.Format(Me.ToString + ": SetClientSizeCore: X={0}; y={1}", x, y))
    End If
End Sub
```

## Kategorie Neuzeichnen des Controls und Anordnen untergeordneter Komponenten

```
' ****
' Anordnen und Neuzeichnen
' ****
```

**OnInvalidated:** Wird aufgerufen, wenn eine Entität das Neuzeichnen des Steuerelement-Inhalts durch Invalidate anfordert. Hier im Beispielprogramm geschieht das durch Resize, GotFocus und LostFocus.

```
Protected Overrides Sub OnInvalidated(ByVal e As System.Windows.Forms.InvalidateEventArgs)
    MyBase.OnInvalidated(e)
    If myShowRepaintAndLayout Then
        Debug.WriteLine(String.Format(
            Me.ToString + ": OnInvalidated: InvalidRect={0}", _
            e.InvalidRect))
    End If
End Sub
```

**OnLayout:** Wird aufgerufen, wenn das Steuerelement anzeigt, dass seine beinhaltenden Steuerelemente aus irgendwelchen Gründen neu angeordnet werden müssen.

```
Protected Overrides Sub OnLayout(ByVal levent As System.Windows.Forms.LayoutEventArgs)
    MyBase.OnLayout(levent)
    If myShowRepaintAndLayout Then
        Debug.WriteLine(String.Format(
            Me.ToString + ": OnLayout: AffectedControl={0}; AffectedProperty={1}", _
            levent.AffectedControl, levent.AffectedProperty))
    End If
End Sub
```

**OnPaintBackground:** Wird aufgerufen, wenn der Hintergrund des *Controls* neu gezeichnet werden muss.

```
Protected Overrides Sub OnPaintBackground(ByVal pevent As System.Windows.Forms.PaintEventArgs)
    MyBase.OnPaintBackground(pevent)
    If myShowRepaintAndLayout Then
        Debug.WriteLine(Me.ToString + ": OnPaintBackground:" + pevent.ClipRectangle.ToString)
    End If
    DrawControlBackground(pevent.Graphics)
End Sub
```

**OnPaint:** Wird aufgerufen, wenn der Fensterinhalt neu gezeichnet werden muss.

```
Protected Overrides Sub OnPaint(ByVal e As System.Windows.Forms.PaintEventArgs)
    MyBase.OnPaint(e)
    If myShowRepaintAndLayout Then
        Debug.WriteLine(Me.ToString + ": OnPaint:" + e.ClipRectangle.ToString)
    End If
    DrawControl(e.Graphics)
End Sub
```

## Kategorie Fokussierung des Controls

```
*****  
'Fokussierung  
*****
```

**OnEnter:** Wird aufgerufen, wenn das Steuerelement dabei ist, den Fokus zu erhalten.

```
Protected Overrides Sub OnEnter(ByVal e As System.EventArgs)
    MyBase.OnEnter(e)
    If myShowFocussing Then
        Debug.WriteLine(Me.ToString + ": OnEnter:" + e.ToString)
    End If
End Sub
```

**OnGotFocus:** Wird aufgerufen, wenn das Steuerelement fokussiert wird, aber nicht, wenn es ein ContainerControl ist, das weitere Komponenten enthält! (In diesem Fall verwenden Sie OnEnter, um das Ereignis zu empfangen.)

```
Protected Overrides Sub OnGotFocus(ByVal e As System.EventArgs)
    MyBase.OnLostFocus(e)
    If myShowFocussing Then
        Debug.WriteLine(Me.ToString + ": OnGotFocus:" + e.ToString)
    End If
    'Das fokussierte Control sieht anders aus als das nicht-fokussierte;
    'deswegen: alles NeuZeichnen
    Invalidate()
End Sub
```

**OnLeave:** Wird aufgerufen, wenn das Steuerelement dabei ist, den Fokus zu verlieren.

```
Protected Overrides Sub OnLeave(ByVal e As System.EventArgs)
    MyBase.OnLeave(e)
    If myShowFocussing Then
        Debug.WriteLine(Me.ToString + ": OnLeave:" + e.ToString)
    End If
End Sub
```

**OnLostFocus:** Wird aufgerufen, wenn das Steuerelement den Fokus verloren hat, aber nicht, wenn es ein ContainerControl ist, das weitere Komponenten enthält! (In diesem Fall verwenden Sie OnLeave, um das Ereignis zu empfangen.)

```
Protected Overrides Sub OnLostFocus(ByVal e As System.EventArgs)
    MyBase.OnLostFocus(e)
    If myShowFocussing Then
        Debug.WriteLine(Me.ToString + ": OnLostFocus:" + e.ToString)
    End If
    'Das fokussierte Control sieht anders aus als das nicht fokussierte;
    'deswegen: alles neu zeichnen.
    Invalidate()
End Sub
```

## Kategorie Tastaturnachrichtenvorverarbeitung des Controls

```
*****
'Tastatur-Vorverarbeitung
*****
```

**ProcessCmdKey:** Wird ausgelöst, wenn eine Befehlstaste (z.B. **ALT+Anfangsbuchstabe**) gedrückt wurde.

```
Protected Overrides Function ProcessCmdKey(ByRef msg As System.Windows.Forms.Message, _
                                         ByVal keyData As System.Windows.Forms.Keys) As Boolean
    If myShowPreProcessing Then
        Debug.WriteLine(Me.ToString + ": ProcessCmdKey:" +
                      msg.ToString + ": KeyData: " + keyData.ToString)
    End If
    'Wenn Ihre Instanz eine Nachricht verarbeitet hat, dann geben Sie
    'True als Funktionsergebnis zurück, sonst False. Die Basis rufen Sie nur
    'auf (dann aber auf jeden Fall!), wenn die Nachricht NICHT verarbeitet wurde.
    Return MyBase.ProcessCmdKey(msg, keyData)
End Function
```

**ProcessDialogChar:** Wird ausgelöst, wenn eine Dialogtaste (auch Steuerungstaste) gedrückt wurde.

```
Protected Overrides Function ProcessDialogChar(ByVal charCode As Char) As Boolean
    If myShowPreProcessing Then
        Debug.WriteLine(Me.ToString + ": ProcessDialogChar: " + charCode)
    End If
    'Wenn Ihre Instanz eine Nachricht verarbeitet hat, dann geben Sie
    'True als Funktionsergebnis zurück, sonst False. Die Basis rufen Sie nur
    'auf (dann aber auf jeden Fall!), wenn die Nachricht NICHT verarbeitet wurde.
    Return MyBase.ProcessDialogChar(charCode)
End Function
```

**ProcessDialogKey:** Wird ausgelöst, wenn eine Dialogtaste (aber keine Steuerungstaste) gedrückt wurde.

```
Protected Overrides Function ProcessDialogKey(ByVal keyData As System.Windows.Forms.Keys) As Boolean
    If myShowPreProcessing Then
        Debug.WriteLine(Me.ToString + ": ProcessDialogKey:" + _
                      ": KeyData: " + keyData.ToString)
    End If
    'Wenn Ihre Instanz eine Nachricht verarbeitet hat, dann geben Sie
    'True als Funktionsergebnis zurück, sonst False. Die Basis rufen Sie nur
    'auf (dann aber auf jeden Fall!), wenn die Nachricht NICHT verarbeitet wurde.
    Return MyBase.ProcessDialogKey(keyData)
End Function
```

## Die Steuerungsroutinen des Beispielprogramms

Der Vollständigkeit halber finden Sie an dieser Stelle auch die Listings der Programmabschnitte, die das Testformular ins Leben rufen, den Code des Testformulars selbst und auch den Code der TestControl-Klasse, deren Instanz die Anwendung zur Laufzeit erstellt.

Soviel an Information vorweg: Sowohl TestControl als auch das Testformular haben zwei zusätzliche Konstruktoren, denen eine Sammlung mit Flags übergeben wird. Diese Flags steuern, welche der Kategorien im Ausgabefenster von Visual Studio protokolliert werden.

### Programmcode von frmTest:

```
Public Class frmTest
    Inherits System.Windows.Forms.Form

    Private myShowCreationDestroy As Boolean
    Private myShowMouse As Boolean
    Private myShowKeyboard As Boolean
    Private myShowPositioning As Boolean
    Private myShowRepaintAndLayout As Boolean
    Private myShowFocussing As Boolean
    Private myShowPreProcessing As Boolean
    Private myShowWndProcMessages As Boolean

    '<Vom Windows Form Designer generierter Code (ausgeblendet)>

    Public Sub New(ByVal ShowCreationDestroy As Boolean, _
```

```

        ByVal ShowMouse As Boolean, _
        ByVal ShowKeyboard As Boolean, _
        ByVal ShowPositioning As Boolean, _
        ByVal ShowRepaintAndLayout As Boolean, _
        ByVal ShowFocussing As Boolean, _
        ByVal ShowPreProcessing As Boolean, _
        ByVal ShowWndProcMessages As Boolean)
 MyBase.New()
myShowCreationDestroy = ShowCreationDestroy
myShowMouse = ShowMouse
myShowKeyboard = ShowKeyboard
myShowPositioning = ShowPositioning
myShowRepaintAndLayout = ShowRepaintAndLayout
myShowFocussing = ShowFocussing
myShowPreProcessing = ShowPreProcessing
myShowWndProcMessages = ShowWndProcMessages
InitializeComponent()
End Sub

```

### **Steuerungscode von TestControl:**

```

Public Class TestControl
    Inherits Control

    Private myShowCreationDestroy As Boolean
    Private myShowMouse As Boolean
    Private myShowKeyboard As Boolean
    Private myShowPositioning As Boolean
    Private myShowRepaintAndLayout As Boolean
    Private myShowFocussing As Boolean
    Private myShowPreProcessing As Boolean

    'Standardkonstruktor: Basiskonstruktor aufrufen
    Public Sub New()
        MyBase.new()
    End Sub

    'Erweiterter Konstruktor: Parameter für die Debug-Ausgaben setzen.
    Public Sub New(ByVal ShowCreationDestroy As Boolean, _
                  ByVal ShowMouse As Boolean, _
                  ByVal ShowKeyboard As Boolean, _
                  ByVal ShowPositioning As Boolean, _
                  ByVal ShowRepaintAndLayout As Boolean, _
                  ByVal ShowFocussing As Boolean, _
                  ByVal ShowPreProcessing As Boolean)
        MyBase.New()
        myShowCreationDestroy = ShowCreationDestroy
        myShowMouse = ShowMouse
        myShowKeyboard = ShowKeyboard
        myShowPositioning = ShowPositioning
        myShowRepaintAndLayout = ShowRepaintAndLayout
        myShowFocussing = ShowFocussing
        myShowPreProcessing = ShowPreProcessing
    End Sub

```

```

'Wird von OnBackgroundPaint aufgerufen, damit der Hintergrund
'des Controls gelöscht wird. Zeichnet hier im Beispiel
'einen gelben Hintergrund.
Protected Overrides Sub DrawControlBackground(ByVal g As Graphics)
    Dim locBrush As New SolidBrush(Color.Yellow)
    g.SetClip(Me.ClientRectangle, Drawing2D.CombineMode.Replace)
    g.FillRectangle(locBrush, Me.ClientRectangle)
End Sub

'Wird von OnPaint aufgerufen, damit das TestControl einen sichtbaren Inhalt hat.
'Zeichnet hier im Beispiel ein umrandetes Kreuz mit einer bestimmten Stiftdicke,
'die von der Fokussierung der Komponente abhängig ist.
Protected Overrides Sub DrawControl(ByVal g As Graphics)
    Dim locPenWidth As Integer
    Dim locClientRecPenWidthIncluded As Rectangle

    'Wenn das Control fokussiert ist,
    If Me.Focused Then
        locPenWidth = 4
    Else
        locPenWidth = 2
    End If

    'Die Dicke des Pens bei den Koordinaten berücksichtigen!
    locClientRecPenWidthIncluded = New Rectangle(
        Me.ClientRectangle.X + locPenWidth \ 2, -
        Me.ClientRectangle.Y + locPenWidth \ 2, -
        Me.ClientRectangle.Width - locPenWidth, -
        Me.ClientRectangle.Height - locPenWidth)

    'Pen zum Malen.
    Dim locPen As New Pen(Color.Black, locPenWidth)

    'Rahmen zeichnen.
    g.DrawRectangle(locPen, locClientRecPenWidthIncluded)

    'Kreuz malen.
    g.DrawLine(locPen, locClientRecPenWidthIncluded.X, locClientRecPenWidthIncluded.Y, -
        locClientRecPenWidthIncluded.Right, locClientRecPenWidthIncluded.Bottom)

    g.DrawLine(locPen, locClientRecPenWidthIncluded.Right, locClientRecPenWidthIncluded.Y, -
        locClientRecPenWidthIncluded.X, locClientRecPenWidthIncluded.Bottom)

End Sub

```

### **Programmcode von frmMain:**

```
Public Class frmMain
    Inherits System.Windows.Forms.Form

    '<Vom Windows Form Designer generierter Code (ausgeblendet)>

    Private Sub btnCreateWithTestControl_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
        Handles btnCreateWithTestControl.Click
        Dim locfrmTest As frmTest
        Dim locTestControl As TestControl
        Dim locButton As Button

        'Einstellungen gelten für das Formular...
        If chkFormular.Checked Then
            locfrmTest = New frmTest(
                chkCreateDestroy.Checked, chkMouse.Checked, chkKeyboard.Checked,
                chkPositioning.Checked, chkRepaintLayout.Checked, chkFocussing.Checked,
                chkPreProcessing.Checked, chkWndProcMessages.Checked)
        Else
            locfrmTest = New frmTest
        End If

        '...und/oder für die TestButton-Komponente.
        If chkSchaltfläche.Checked Then
            locTestControl = New TestControl(
                chkCreateDestroy.Checked, chkMouse.Checked, chkKeyboard.Checked,
                chkPositioning.Checked, chkRepaintLayout.Checked, chkFocussing.Checked,
                chkPreProcessing.Checked)
        Else
            locTestControl = New TestControl
        End If

        'Einstellungen für das Formular durchführen.

        'Das Formular hat ein Viertel der Bildschirmgröße
        'und soll in der Bildschirmmitte des primären Bildschirms erscheinen.
        Dim locBounds As Rectangle = Screen.PrimaryScreen.Bounds
        locfrmTest.Width = locBounds.Width \ 4
        locfrmTest.Height = locBounds.Height \ 4
        locfrmTest.StartPosition = FormStartPosition.CenterScreen
        locfrmTest.Text = txtFormText.Text

        'Einstellungen entsprechend der CheckBox-Controls im Formular
        locfrmTest.ControlBox = chkControlBox.Checked
        locfrmTest.MinimizeBox = chkMinimizeBox.Checked
        locfrmTest.MaximizeBox = chkMaximizeBox.Checked
        locfrmTest.HelpButton = chkHelpButton.Checked
        locfrmTest.ShowInTaskbar = chkShowInTaskbar.Checked
        locfrmTest.TopMost = chkTopMost.Checked
        locfrmTest.KeyPreview = chkKeyPreview.Checked
        locfrmTest.AutoScroll = chkScrollBars.Checked
```

```

'TestControl mittig und im oberen Drittel Formular platzieren
'- nicht zu klein oder zu groß.
locTestControl.Width = CInt(locfrmTest.ClientSize.Width / 2)
locTestControl.Height = CInt(locfrmTest.ClientSize.Height / 3)
locTestControl.Location =
    New Point(CInt(locfrmTest.ClientSize.Width / 2 - locTestControl.Width / 2),
              CInt(locfrmTest.ClientSize.Height / 3 - locTestControl.Height / 2))

'TestControl verankern, wenn die AutoScroll-Funktion des Formulars nicht gewünscht wird
If Not chkScrollBars.Checked Then
    locTestControl.Anchor = AnchorStyles.Bottom Or AnchorStyles.Top Or _
                           AnchorStyles.Left Or AnchorStyles.Right
End If

'Schließschaltfläche im unteren Drittel positionieren.
locButton = New Button
locButton.Width = CInt(locfrmTest.ClientSize.Width / 3)
locButton.Height = CInt(locfrmTest.Height / 6)
locButton.Location =
    New Point(CInt(locfrmTest.ClientSize.Width / 2 - locButton.Width / 2),
              CInt(locfrmTest.ClientSize.Height / 4 * 3 - locButton.Height / 2))

locButton.Text = "Formular &schließen"
'Schließschaltfläche verankern, wenn die AutoScroll-Funktion des Formulars nicht gewünscht wird.
If Not chkScrollBars.Checked Then
    locButton.Anchor = AnchorStyles.Bottom Or _
                      AnchorStyles.Left Or AnchorStyles.Right
End If

'Return und Escape lösen Click-Ereignis des Buttons aus.
Me.AcceptButton = locButton
Me.CancelButton = locButton

'Zur Laufzeit einstellen, dass das Click-Ereignis der Schließschaltfläche
'veon TestButton-Click behandelt wird.
AddHandler locButton.Click, AddressOf TestButton_Click

'Beide Controls der Formular-ControlCollection hinzufügen;
'damit werden die beiden Komponenten windowstechnisch angelegt und dargestellt.
locfrmTest.Controls.Add(locTestControl)
locfrmTest.Controls.Add(locButton)

locfrmTest.Show()
End Sub

'Ereignis-Routine des Buttons; wird zur Laufzeit eingebunden (s.o.).
Sub TestButton_Click(ByVal Sender As Object, ByVal e As EventArgs)

    Dim locButton As Button

    'Könnte schief gehen, wenn Sender nicht die Schaltfläche ist,
    'deswegen sichergehen durch Try/Catch.
    Try

```

```

'Das sendende Objekt herausfinden
locButton = DirectCast(Sender, Button)
Catch ex As Exception
    Return
End Try
'Sendendes Objekt war der Button selbst, dann dessen Parent (das Formular) entsorgen.
'Damit wird das Formular, das den Button enthält, geschlossen.
locButton.Parent.Dispose()
End Sub
End Class

```

## Anmerkungen zum Beispielprogramm

Dem einen oder anderen unter Ihnen könnte es sich nicht auf den ersten Blick erschließen, wie der Inhalt von TestControl letzten Endes auf den Bildschirm gelangt. Die Zeichenroutinen sind ja offensichtlich die, die sich in OnPaint befinden. Das würde ja bedeuten, dass das Steuerelement sich jedes Mal neu zeichnen muss, wenn es beispielsweise durch ein anderes Fenster zuvor verdeckt wurde – und man könnte meinen, dass das sehr lange dauerte. Die Frage, die sich aus diesem Grund vielleicht stellt: Gibt es keinen Weg, den Inhalt von TestControl nur einmal zeichnen zu müssen, und der bleibt dann bestehen?

Genau das ist aber die Vorgehensweise, wenn Sie benutzerdefinierte Inhalte in Formularen oder sichtbaren Komponenten (Steuerelementen) anzeigen lassen wollen. Wenn Sie – wie in einem vorherigen Beispiel zu sehen war – beispielsweise die Image-Eigenschaft des PictureBox-Steuerelements bestimmen, brauchen Sie sich um das ständige Neuzeichnen nicht selbst zu kümmern. Dafür muss das PictureBox in der OnPaint-Prozedur selbst erledigen. Es gibt unter Windows keine festen Inhalte in *der Form*. Wenn ein Fenster von einem anderen überdeckt wird und dann wieder sichtbar wird, muss es seinen Inhalt neu zeichnen – diese Vorgehensweise muss jedes Windows-Programm anwenden, egal in welcher Sprache es entwickelt wurde. Selbst wenn es so aussieht, als sei der Inhalt »fest«, gibt es dennoch irgendwo eine Routine, die dafür sorgt, dass er nur »fest wirkt«. Das nächste Kapitel und ► Kapitel 30 verraten Ihnen weitere Geheimnisse zu diesem Thema.



# 29 GDI+ zum Zeichnen von Formular- und Steuerelementinhalten verwenden

---

864 Einführung in GDI+

878 Flimmerfreie, fehlerfreie und schnelle Darstellungen von GDI+-Zeichnungen

884 Was Sie beim Zeichnen von breiten Linienzügen beachten sollten

---

GDI+ ist eine Klassenbibliothek, die das Zeichnen verschiedener Elemente in Windows (Windows im Sinne vom Windows-Betriebssystem) erlaubt. GDI steht als Abkürzung von *Graphic Device Interface* – etwa *grafische Geräteschnittstelle* – und das Pluszeichen hinter dem Akronym lässt schon vermuten, dass es sich um etwas Weiterentwickeltes handeln muss.

Allerdings ist das im Grunde genommen nur halb richtig – jedenfalls in der gegenwärtigen Version des GDI+. GDI+ ist der von Microsoft erklärte Nachfolger des GDI, der, wenn auch in ständig weiterentwickelter Form, schon zu 16-Bit-Windows-Zeiten sozusagen den ausführenden Produzenten für alles darstellte, was in irgendeiner Form auf dem Bildschirm erscheinen sollte. Aus diesem Grund erfuhren viele Grundfunktionen des GDI eine umfangreiche Unterstützung durch die Treiber von Grafikkarten der verschiedensten Hersteller. Das heißt im Klartext: Wenn Sie eine Linie von einem zum anderen Punkt auf dem Bildschirm mit GDI zeichnen, dann ist es nicht eine bestimmte Prozedur im GDI, die die eigentlichen Punkte setzt, sondern die Grafikkarte selbst, die diese Aufgabe übernimmt. Das GDI teilt dem Treiber lediglich mit, dass es eine Linie gezeichnet haben möchte. Und genau das ist zurzeit noch der Unterschied zum GDI+. Es bietet viel umfassendere und vor allen Dingen auch einfacher zu handhabende Zeichenfunktionen, doch viele davon sind momentan noch nicht hardwareunterstützt. Und das ist der Grund, weswegen Microsoft mit GDI+ grundsätzlich auf dem richtigen Weg ist, es sich für einige wenige Anwendungen unter Windows aus Geschwindigkeitsgründen aber einfach noch nicht hundertprozentig zur Verwendung eignet.

Nun ist das Framework für die Zukunft konzipiert, und in zukünftigen .NET- und Windows-Betriebssystemversionen werden sich definitiv leistungsfähigere Werkzeuge für die Darstellung von Grafik befinden – die *Windows Presentation Foundation* (auch bekannt unter seinem Codenamen »Avalon«) wird es nach Fertigstellung dieses Buches sowohl noch für Windows XP als natürlich auch für den Windows XP-Nachfolger Windows Vista geben. Aus diesem Grund hat es wohl für die Entwickler des Frameworks bislang keinen Sinn ergeben, die ältere Version des GDI (ohne Plus) als Klassenlibrary in .NET zu implementieren.

Außer Acht gelassen, was die wirklichen Gründe für die Entscheidung zu diesem Schritt waren, gilt: Es ist derzeit jedenfalls nicht vorhanden, und es wird aller Wahrscheinlichkeit nach auch niemals implementiert werden. Für den Moment müssen wir für den Inhalt der Dokumente unserer .NET-Programme mit der aktuellen Version des GDI+ vorlieb nehmen.<sup>1</sup>

---

**HINWEIS:** GDI+ liefert genug Stoff, um ein eigenes Buch damit zu füllen. Charles Petzold hat das mit seiner Core-Reference,<sup>2</sup> die ein wenig GDI+-lastig geworden ist, eindrucksvoll unter Beweis gestellt. Aus diesem Grund möchte ich mich an dieser Stelle nur auf grundlegendste Erklärungen zum GDI+ beschränken – gerade auf so viel, dass es für andere Projekte ausreicht – um beispielsweise Benutzersteuerelemente mit sinnvollen Inhalten zu füllen. In den nächsten Abschnitten finden Sie deswegen zunächst die wichtigsten Informationen, die Sie als Grundlage benötigen, um die ersten Gehversuche mit dem GDI+ bewältigen zu können, in sehr kompakter Form. Aus Platzgründen möchte ich mich nicht in endlos langen Beschreibungen von einzelnen Funktionen des GDI+ verlieren. Vielmehr möchte ich mich auf die *Anwendung* des GDI+ zur Lösung bestimmter Probleme beim Entwickeln von Komponenten und Anwendungen konzentrieren. Ersatzweise verwenden Sie für Fragen, die die korrekte Anwendung einer Methode oder eines Konstruktors betreffen, die Online-Hilfe von Visual Studio. Sie leistet gerade beim GDI+ als Referenz ausgezeichnete Arbeit. Durch die IntelliSense-Funktion des Editors werden selbst viele Blicke in die Online-Hilfe überflüssig.

---

## Einführung in GDI+

Wie schon an mehreren Stellen in diesem Buch beschrieben, ist alles, was Sie in ein Windows zeichnen, hoch volatil. Es hält genau so lange, wie sie etwas anderes über das Fenster schieben. Dann ist sein Inhalt verschwunden. Das gilt sogar für das Fenster selbst, das den Inhalt darstellt: Wenn Sie ein Fenster in einer bestimmten Größe auf den Bildschirm gebracht haben, seinen Inhalt zeichnen, es anschließend verkleinern und wieder auf seine alte Größe bringen, ist der Ursprungsinhalt ebenfalls verloren.

Das liegt am Prinzip, wie Windows-Inhalte gemanagt werden: Jedes Fenster ist für das Zeichnen seines Inhaltes selbst verantwortlich. Alle auf Control basierenden Komponenten (dazu gehören auch Formulare) müssen deswegen ihr Paint-Ereignis behandeln oder noch besser, da schneller, die OnPaint-Methode ihrer Basisklasse überschreiben. In beiden Fällen wird den Prozeduren das so genannte Graphics-Objekt mit dem PaintEventArgs-Objekt übergeben, das den Dreh- und Angelpunkt für alle Grafikoperationen bildet.

---

<sup>1</sup> Natürlich gibt es auch die Möglichkeit, direkte Betriebssystemaufrufe an das herkömmliche GDI aus .NET heraus durchzuführen – doch das sind dann grundsätzlich so genannte *Unsafe Calls* (unsichere Aufrufe). Unsicher deswegen, weil das Programm das behütete, sichere .NET-Zuhause verlässt. Und solche Schritte sind nur in einer voll vertrauenswürdigen Umgebung erlaubt. Das heißt, dass alle Programme, die unsichere Aufrufe durchführen oder sonst irgendeinen unsicheren Code ausführen, mit den Standardsicherheitseinstellungen nur auf dem Computer selbst ausgeführt werden können. Starten Sie ein solches Programm beispielsweise von einer Netzwerk-Ressource, löst es beim Erreichen des unsicheren Codes eine Sicherheitsausnahme aus.

<sup>2</sup> Ein sehr empfehlenswertes Buch, gerade wenn es um GDI+ und native Textausgabe/-formatierung geht. Englische Ausgabe (.NET 1.0/1.1), ISBN: 0-7356-1799-6. Deutsche Ausgabe: Windows-Programmierung mit Visual Basic .NET, ISBN: 3-86063-691-X.

Die Zeichenroutinen, die dieses Graphics-Objekt nun zur Verfügung stellt, beziehen sich auf den so genannten *Client*-Bereich des Fensters. Das ist das Innere des Fensters, also der Bereich, ohne Rahmen, Titel oder Rollbalken.

Sie können ein Graphics-Objekt übrigens nicht selbst direkt durch seinen Konstruktor erstellen; sie können es nur durch bestimmte Funktionen ermitteln – der Graphics-Parameter, der Ihnen durch OnPaint mit PaintEventArgs geliefert wird, ist nur ein (wenn auch das am häufigsten auftretende) Beispiel dafür.

Sie haben zwar auch die Möglichkeit, das für eine Control-Ableitung gültige Graphics-Objekt auch ohne die OnPaint-Ereignisparameter zu bekommen, doch ist das Anwenden dieser Vorgehensweise eher selten der Fall. Außerdem führt es oft zu einer falschen Vorgehensweise, wie das folgende Beispiel zeigt:

---

**BEGLEITDATEIEN:** Sie finden dieses Projekt im Verzeichnis .\VB 2005 - Entwicklerbuch\G - SmartClient\Kap29\Simple-Gdi01\.

---

```
Public Class frmMain
```

```
    Private Sub btnLinieZeichnen_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
        Handles btnLinieZeichnen.Click
```

```
        Dim g As Graphics
```

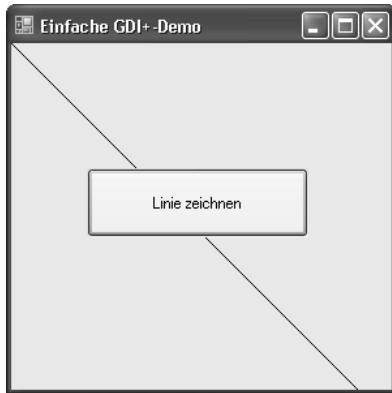
```
'Ermittelt das Graphics-Objekt, das zu einem bestimmten
'Window gehört, dessen Handle (ID) zur Identifizierung dient.
g = Graphics.FromHwnd(Me.Handle)
```

```
'Schwarze, ein Pixel dünne Linie zeichnen von (0,0) zu (500,500).
'Koordinaten werden standardmäßig in Pixel angegeben;
'(0,0) liegt in der linken, oberen Ecke des Client-Bereichs.
g.DrawLine(New Pen(Color.Black), 0, 0, 500, 500)
```

```
    End Sub
```

```
End Class
```

Wenn Sie dieses Programm starten, werden Sie nach dem Anklicken der einzigen Schaltfläche zwei Dinge feststellen: a) Das ermittelte Graphics-Objekt bezieht sich tatsächlich nur auf das Fenster, für das es ermittelt wurde. Denn obwohl der Linienpfad die Schaltfläche kreuzt, zeichnet der Befehl die Linie im Bereich der Schaltfläche nicht (siehe Abbildung 29.1). Und b) Wenn Sie irgendetwas über das Formular bewegen (der Windows-Taschenrechner eignet sich für solche Experimente am besten), ist die Linie verschwunden.



**Abbildung 29.1:** Zeichenoperationen mit einem Graphics-Objekt beziehen sich nur auf das Fenster, für das das Graphics-Objekt ermittelt wurde

Wenn Sie wollen, dass die Linie auch noch vorhanden ist, *nachdem* sich ein anderes Objekt über dem Formular befunden hat, müssen Sie ein anderes Verfahren verwenden, das grob skizziert folgendermaßen funktioniert:

- Alle Zeichenroutinen finden in der Paint-Ereignisbehandlungsroutine des Formulars statt. Dazu können Sie – wie schon gesagt – entweder das Ereignis im Formular einbinden oder, der zu empfehlende Weg, die Basisprozedur OnPaint überschreiben, die das Zeichnen der Linie vornimmt.
- Damit die Linie nur dann gezeichnet wird, wenn die Schaltfläche vom Anwender angeklickt wurde, muss es eine Art Informationsspeicher geben (ein Flag, das von überall im Formular zugänglich ist – also einen Klassen-Member), der OnPaint darüber informiert, ob die Linie im Falle eines neu zeichnen Müssens überhaupt gemalt werden soll.
- Damit die Linie auch dann gezeichnet wird, wenn der Anwender die Schaltfläche angeklickt hat, muss der Code zur Behandlung des Klickereignisses für die Schaltfläche nicht nur das Flag setzen, sondern auch den kompletten Client-Bereich des Formulars für ungültig erklären. Geschieht das mit einer bestimmten Methode namens Invalidate, wird automatisch das Paint-Ereignis ausgelöst, damit OnPaint aufgerufen und die Linie gezeichnet werden.

---

**BEGLEITDATEIEN:** Sie finden das Projekt, das diese Modifizierungen enthält, im Verzeichnis *.|VB 2005 - Entwicklerbuch|G - SmartClient\Kap29\SimpleGdi02.*

---

Der dazugehörige Formularcode sieht folgendermaßen aus:

```
Public Class frmMain
    Inherits System.Windows.Forms.Form
    Private myDrawLineFlag As Boolean

    'Klassenweiter Member, der von Click und OnPaint aus zugänglich ist.
    Private myDrawLineFlag As Boolean

    Private Sub btnLinieZeichnen_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles _
        btnLinieZeichnen.Click
```

```

'Ab sofort darf gezeichnet werden!
myDrawLineFlag = True
'Client-Bereich für ungültig erklären --> OnPaint wird ausgelöst,
'und damit, da myDrawLineFlag jetzt true ist, die Linie gezeichnet.
Me.Invalidate()

End Sub

Protected Overrides Sub OnPaint(ByVal e As System.Windows.Forms.PaintEventArgs)

If myDrawLineFlag Then
    Dim g As Graphics = e.Graphics

    'Ermittelt das Graphics-Objekt, das zu einem bestimmten
    'Window gehört, dessen Handle (ID) zur Identifizierung dient.
    g = Graphics.FromHwnd(Me.Handle)

    'Schwarze, ein Pixel dünne Linie zeichnen von (0,0) zu (500,500).
    'Koordinaten werden standardmäßig in Pixel angegeben;
    '(0,0) liegt in der linken, oberen Ecke des Client-Bereichs.
    g.DrawLine(New Pen(Color.Black), 0, 0, 500, 500)
End If
End Sub
End Class

```

Wenn Sie diese Version des Programms starten, funktioniert es auf den ersten Blick wie die erste Version des Programms, mit einem entscheidenden Unterschied: Das Formular zeichnet sich in den entscheidenden Schritten neu, und das ist schließlich auch genau das, was es muss.

Sie sollten dieses einfache Prinzip in Ihren eigenen Anwendungen berücksichtigen, wenn Sie selbst gezeichnete Inhalte in Formularen darstellen müssen.

## **Linien, Flächen, Pens und Brushes**

Im Beispiel des letzten Abschnittes haben Sie bereits teilweise sehen können, dass für alle Zeichenoperationen gilt: Sie benötigen entweder ein Pen-Objekt (einen Stift) oder ein Brush-Objekt (einen Pinsel), um Linien oder Flächen zu zeichnen.

Für Liniendefinitionen benötigen Sie ein Pen-Objekt. Ein Pen bestimmt, wie ein Linienzug gezeichnet werden soll – oder genauer: in welcher Farbe oder welcher Stärke das geschehen soll. Wie die Anwendung des Pen-Objektes generell funktioniert, konnten Sie bereits im vergangenen Beispiel sehen. Der Pen-Konstruktor verfügt über mehrere Überladungen. Neben Farbe und Linienstärke haben Sie die Möglichkeit, ein Pen-Objekt auch aus einem Brush-Objekt zu erstellen und so dickere Linien oder Umrisse bestimmter geometrischer Figuren mit einer spezifischen Füllung zu versehen.

Brush-Objekte in GDI+ sind vielseitig. Sie dienen der Bestimmung der Eigenschaften von Flächenfüllungen. Das Brush-Objekt selbst ist, anders als ein Pen, eine abstrakte Basisklasse, es kann also nicht direkt instanziert und verwendet werden. Stattdessen gibt es in GDI+ fünf verschiedene Brush-Ableitungen, die unterschiedliche Aufgaben beim Erstellen von Füllfarben bzw. -mustern für Flächen erfüllen:

- **SolidBrush:** Definiert einen simplen, einfarbigen Pinsel. Flächen, die Sie mit einem SolidBrush füllen, werden mit der durch den Brush bestimmten Farbe ausgefüllt.
- **HatchBrush:** Definiert einen rechteckigen Pinsel mit einer Schraffurart; Vorder- und Hintergrundfarbe der Schraffur sind dabei getrennt einstellbar.
- **TextureBrush:** Definiert einen Pinsel aus einer Bitmap. In welcher Form die Bitmap als Zeichenvorlage für eine Füllung verwendet wird, stellen Sie über den Konstruktor ein. Interessantes hierzu finden Sie in der VS.NET-Onlinehilfe unter dem ImageAttributes-Objekt und der WrapMode-Enum.
- **LinearGradientBrush:** Definiert ein Brush-Objekt mit einem linearen Farbverlauf.
- **PathGradientBrush:** Definiert ein Brush-Objekt mit einem Farbverlauf, der durch einen beliebigen Kurvenverlauf (GraphicsPath) bestimmt wird. Ein solcher Pinsel entsteht entweder aus einem GraphicsPath-Objekt oder einem Points- bzw. PointF-Array, das die Eckpunkte des Kurvenverlaufs bestimmt.

Grundsätzlich gilt: Wenn Sie eine Linie, einen Linienzug ein Polygonumriss oder einen Kreisumriss zeichnen möchten, verwenden Sie eine der DrawXXX-Methoden des Graphics-Objektes. Sie benötigen zum Zeichnen in diesen Fall ein Pen-Objekt, das die Eigenschaften des verwendeten Stiftes bestimmt.

Möchten Sie eine Fläche füllen, verwenden Sie eine der FillXXX-Methoden des Graphics-Objektes. Sie benötigen zum Zeichnen dann eines der von Brush abgeleiteten Objekte, das die Eigenschaften der Füllung bestimmt.

## Angabe von Koordinaten

Die meisten Zeichenfunktionen, die Ihnen das Graphics-Objekt zur Verfügung stellt, arbeiten mit der Angabe von Koordinaten, die Ihnen entweder als Single- oder als Integer-Werte übergeben werden. Der Koordinatenursprung liegt in der linken, oberen Ecke des Bildschirms. Neben der Bestimmung von Koordinaten durch einzelne Single- oder Integer-Werte hält das Framework einige Strukturen bereit, um Sie bei der Angabe von Positionen, Umrissen oder Größen zu unterstützen. Die wichtigsten sind:

- **Point:** Die Point-Struktur dient zur Angabe einer Koordinate. Sie übergeben ihr im Konstruktor den X- und Y-Wert als Integerwert.
- **PointF:** Die PointF-Struktur dient zur Angabe einer Koordinate mit Fließkommawerten. Sie übergeben ihr im Konstruktor den X- und Y-Wert jeweils als Wert vom Typ Single.
- **Size:** Die Size-Struktur dient zur Angabe des Ausmaßes eines rechteckigen Objektes. Sie übergeben ihr im Konstruktor die Breite und Höhe als Integer-Wert.
- **SizeF:** Die SizeF-Struktur dient zur Angabe des Ausmaßes eines rechteckigen Objektes mit Fließkommawerten. Sie übergeben ihr im Konstruktor die Breite und Höhe jeweils als Wert vom Typ Single.
- **Rectangle:** Diese Struktur dient zur Angabe von Position und Ausmaßen eines Rechtecks. Ein Rectangle definiert sich durch einen Startpunkt, der als Koordinate angegeben wird, sowie durch die Höhe und die Breite.
- **RectangleF:** Es gilt das zu Rectangle Gesagte, nur dass die Werte als Fließkommawert vom Typ Single angegeben werden.

## **Wieso Integer- und Fließkommaangaben für Positionen und Ausmaße?**

Das Graphics-Objekt ist nicht auf Pixel als feste Maßeinheit für die Positions- bzw. Größenangaben festgelegt. Vielmehr erlauben die `PageUnit`-Eigenschaft, die `PageSize`-Eigenschaft sowie verschiedene Transformationsmethoden eine individuelle Skalierung des Koordinatensystems. Integerwerte sind dann natürlich nicht mehr ausreichend, wenn Sie beispielsweise *Inches* (englische Bezeichnung für die Maßeinheit Zoll, entspricht 2,54 cm) als Maßeinheit bestimmt haben. Auf dem Bildschirm liegen zwischen zwei Pixel, die einen Inch auseinander liegen, natürlich viele weitere Pixel. Die dazwischen liegenden Punkte ließen sich nicht erreichen, könnte man keine gebrochenen Werte für Koordinaten angeben.

Beide Verfahren haben Vor- und Nachteile: Wenn Sie sich für Pixel als Maßeinheit entscheiden (das ist die Standardeinstellung, die durch `PageUnit` bestimmt wird), können Sie Integer-Werte verwenden; interne Skalierungsumrechnungen entfallen dabei, allerdings können Sie damit nur bestimmte Anwendungen realisieren, wie beispielsweise die Begrenzungen eigener Steuerelemente zeichnen, deren Ausmaße ohnehin in Pixel angegeben werden.

Entscheiden Sie sich für eine andere Maßeinheit, wie beispielsweise Millimeter oder Inch, müssen Sie Fließkommawerte verwenden, um alle Pixel im Koordinatensystem ansteuern zu können. Intern finden natürlich wieder Umrechnungen auf die eigentlichen, physischen Pixelkoordinaten statt, und das kostet einen wenig Rechenzeit. Allerdings ist die Anwendung von Maßeinheiten, die Sie aus der »echten« Welt kennen, für die Umsetzung vieler Anwendungen leichter und flexibler.

## **Wie viel Platz habe ich zum Zeichnen?**

Um die zur Verfügung stehenden Ausmaße eines Formulars (oder einer von `Control` abgeleiteten Klasse) zu ermitteln, gibt es zwei Möglichkeiten.

- Wenn Ihnen das Graphics-Objekt bekannt ist, verwenden Sie die Nur-Lesen-Eigenschaft `VisibleClipbounds` des Grafikobjektes. Der Vorteil: Wenn Sie mit der `PageUnit`-Eigenschaft eine andere Maßeinheit für das Koordinatensystem eingestellt haben, liefert `VisibleClipbounds` die Ausmaße in den Einheiten zurück, die durch `PageUnit` eingestellt sind.
- Mit `ClientArea` des Formulars oder der verwendeten `Control`-Klasse (oder deren Ableitung) ermitteln Sie den sichtbaren Zeichenbereich in Pixel. Sie benötigen dazu kein Graphics-Objekt.

## **Das gute, alte Testbild und GDI+ im Einsatz sehen!**

Nach so viel grauer Theorie sind Sie sicherlich gespannt, GDI+-Funktionen in der Praxis zu sehen. Seit der Einführung von Kabel- und Satellitenfernsehen Mitte der 80er Jahre können wir uns über eine Unterversorgung mit farbigen Bildern nicht mehr beklagen. Allerdings: Ein bestimmtes Programm ist fast gänzlich von unseren Mattscheiben verschwunden – gemeint ist das gute, alte Testbild. Die Älteren unter Ihnen werden sich sicherlich noch erinnern können, wie es uns erging, wenn wir morgens nicht zur Schule wollten, weil dort entweder eine nicht zu packende Klassenarbeit oder der Kerl aus der 4. Klasse auf uns wartete. Diesem hatten wir tags zuvor sein Pausenbrot »frisiert«, gleichzeitig aber nicht bedacht, dass wir zwar der Held des Tages aber auch ein potentieller Kandidat für die Notaufnahme am nächsten Tag waren. Also hieß es: Fieberthermometer in den Tee, Seifen-

wasser in die Augen (zwei Minuten mit aufgerissenen Lidern ohne zu blinzeln reichten meist auch aus, um die notwendige Augenrötung herbeizuführen) und nach einigen, Oscar-verdächtigen Jammerereinlagen ging's ab aufs elterliche Sofa, wo die Flimmerkiste – gerade in Farbe – schon auf ein gut sortiertes Programm wartete: Telekolleg im Dritten, Testbild im Zweiten und Testbild im Ersten.

Und dann hieß es: Sehnsüchtig auf die Sesamstraße um halb zehn warten. Das Testbild, geben Sie es zu, hassten Sie damals sicherlich, wie ich es tat. Und jetzt? Jetzt verbinden wir alle wahrscheinlich kindliche Unbeschwertheit mit exakt dieser Grafik, und wäre es nicht schön, solche Momente nochmals erleben zu können?

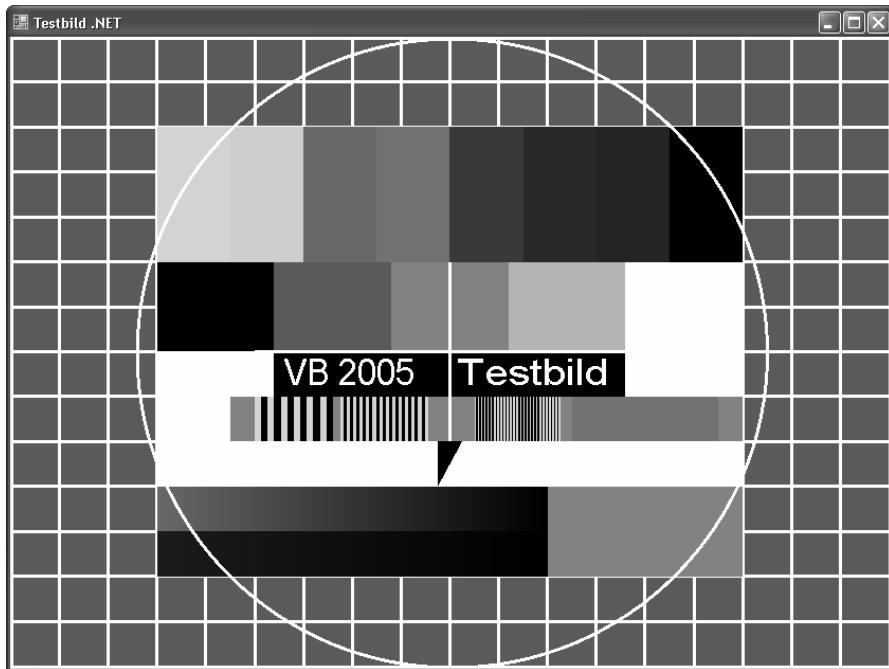
Aber: Warum jammern? Wir haben Rechenpower, wir haben Visual Basic und wir haben GDI+! Holen wir es uns zurück!

---

**BEGLEITDATEIEN:** Sie finden das folgende Projekt unter `\VB 2005 - Entwicklerbuch\G - SmartClient\Kap29\VB-Testbild01`; es trägt den Projektnamen `VBTestbild`, und wenn Sie meiner Generation (so ungefähr 1966–1972) angehören, werden Sie es noch kennen und lieben!

---

Wenn Sie dieses Programm starten, sehen Sie das Testbild, etwa wie in Abbildung 29.2 gezeigt. Sie können dieses Bild vergrößern und verkleinern, und Sie werden feststellen, dass sich sein Inhalt immer an die aktuelle Größe anpasst. Mal ganz abgesehen davon, dass es heftig flimmert (jedenfalls in dieser Version), ist das der Beweis, dass es sich bei der Grafik nicht um eine simple, im Internet geklaute Bitmap handelt, sondern jeder einzelne Strich, jede einzelne Fläche und jedes Polygon farbig und bunt zur Laufzeit gezeichnet wird. Und glauben Sie mir: Salopp gesagt, war das Kreieren dieses Programms eine tierische Fuckelei, aber ich finde, der Aufwand hat sich gelohnt.



**Abbildung 29.2:** Für Erinnerungen an die kindliche Unbeschwertheit – das Testbild

Gelohnt, im Übrigen, nicht nur wegen des Ergebnisses, sondern auch, weil das alte Testbild der Öffentlichen-Rechtlichen (ob man damals schon so weit gedacht hat?) viele unterschiedliche Elemente enthielt, die den Einsatz von vielen verschiedenen GDI+-Funktionen erforderlich macht – Sie werden das gleich sehen.

Einige Worte zur Funktionsweise vorweg: Ich habe das Formular so gestaltet, dass es eher als Komponente denn als Formular fungiert. Aus diesem Grund gibt es einige Member-Variablen, zu denen auch Äquivalente als öffentliche Eigenschaften existieren. Das gibt uns zum einen die Möglichkeit, das komplette Formular später abzuleiten und bestimmte Verhaltensweisen durch gezieltes Über-schreiben von Eigenschaftenprozeduren zu beeinflussen. Zum anderen können wir aus dem Formular auch ein Steuerelement machen, dessen Verhalten sich durch die öffentlichen Eigenschaften gezielt von außen steuern lässt. Aus Platzgründen finden Sie die meisten dieser Eigenschaften in der vorliegenden Version nicht abgedruckt, da sie die Inhalte der geschützten Member-Variablen, die das Zeichnen eigentlich steuern, nur nach oben durchreichen.

Das Codelisting:

```
Imports System.Drawing.Drawing2D  
  
Public Class frmMain  
    Inherits System.Windows.Forms.Form
```

Einige Funktionen des Formulars zum Zeichnen werden aus dem Namespace Drawing2D benötigt; deswegen die entsprechende Imports-Anweisung am Anfang des Codes. Die Formulkarklasse wird aus Form abgeleitet.

```
'Legt fest, ob das Gitter gezeichnet werden soll oder nicht.  
Private myDrawGrid As Boolean  
'Bestimmt, wie das Gitter gezeichnet werden soll.  
Private myGridStyle As GridStyle  
'Bestimmt die Stiftbreite für das Zeichnen des Gitters/Rasters.  
Private myGridLineWidth As Integer  
'Bestimmt die Stiftfarbe für das Zeichnen des Gitters/Rasters.  
Private myGridColor As Color  
'Bestimmt die Hintergrundfarbe.  
Private myBackground As Color  
'Bestimmt die Farben der oberen Balkenreihe.  
Private myBarColors As Color()  
'Bestimmt die Grauschattierungen der darunterliegenden Balkenreihe.  
Private myBarShades As Color()  
'Bestimmt die Grauschattierungen der Balkenreihe, in der sich die Beschriftung befindet.  
Private myBarTitleShades As Color()  
'Definiert die Anzahl der Gitter-/Rasterspalten.  
Private myGridCols As Integer  
'Definiert die Anzahl der Gitter-/Rasterzeilen.  
Private myGridRows As Integer  
'Definiert, in Rasterzeilen gerechnet, den Abstand des "Bildes" von oben.  
Private myUpperOffset As Integer  
'Definiert den Zeichenmodus.  
Private mySmoothingMode As SmoothingMode
```

Die Member-Variablen der Klasse folgen anschließend. Sie bestimmen, in welcher Weise Gitter/Raster und eigentliches Bild gemalt werden sollen. Die Variablen selbst werden im Konstruktor des Programms initialisiert:

```
Public Sub New()
    MyBase.New()

    ' Dieser Aufruf ist für den Windows Form-Designer erforderlich.
    InitializeComponent()

    ' Initialisierungen nach dem Aufruf InitializeComponent() hinzufügen.
    DrawGrid = True
    GridStyle = GridStyle.Lines
    GridLineWidth = 2
    GridColor = Color.White

    '18 Spalten und 14 Zeilen
    GridCols = 18
    GridRows = 14

    'Bild beginnt in der 3. Reihe.
    UpperOffset = 2

    'Hintergrund ca. 70% grau
    Background = Color.FromArgb(90, 90, 90)

    'Farben für die Farbbalken
    BarColors = New Color() {Color.Pink, Color.GreenYellow, Color.Magenta, Color.Olive, _
                           Color.DarkMagenta, Color.DarkRed, Color.Indigo, Color.Black}
    'Farben für die darunterliegenden Grauflächen
    BarShades = New Color() {Color.FromArgb(0, 0, 0), _
                           Color.FromArgb(90, 90, 90), _
                           Color.FromArgb(130, 130, 130), _
                           Color.FromArgb(180, 180, 180), _
                           Color.FromArgb(255, 255, 255)}
    'Farben für die darunterliegenden Grauflächen der Titelzeile
    BarTitleShades = New Color() {Color.White, _
                                 Color.Black, _
                                 Color.Black, _
                                 Color.Black, _
                                 Color.White}

    'Fenstertitel
    Text = "Testbild .NET"
End Sub
```

Einige Anmerkungen zur Angabe von Farben bei der Verwendung des *Graphics*-Objektes: Die *Color*-Struktur bietet eine elegante und einfache Möglichkeit, Farben zu definieren. Sie verfügt über zahlreiche, statische Funktionen, um Farbwerte anhand von Farbnamen zu ermitteln. Die Farb- und Graubalken, die Sie im Testbild sehen, haben zwar feste Ausmaße, aber Sie können die Anzahl der Balken, die sich in dieser Größenvorgabe befinden, variieren, indem Sie den entsprechenden Arrays Elemente hinzufügen oder aus ihnen entfernen.

```

'Wird aufgerufen, wenn das Bild aus irgendeinem Grund neu gezeichnet werden muss.
Protected Overrides Sub OnPaint(ByVal e As System.Windows.Forms.PaintEventArgs)
    paintBackground(e.Graphics)
    paintContent(e.Graphics)
End Sub

```

Diese Routine wird automatisch aufgerufen, wenn ein Neuzeichnen des Bildes erforderlich wird. Sie zeichnet zunächst den Hintergrund des Bildes und anschließend das eigentliche Testbild. Diese Routinen sehen folgendermaßen aus:

```

'Hier wird der eigentliche Inhalt gezeichnet.
Private Sub paintContent(ByVal g As Graphics)

    Dim locPen As Pen
    Dim locBrush As Brush
    Dim locBarWidth As Single
    Dim locX As Single
    Dim locRectF As RectangleF

    g.SmoothingMode = mySmoothingMode

    'Raster malen im Bedarfsfall
    If DrawGrid Then
        paintGrid(g)
    End If

    'Figuren malen:

    'obere Balkenreihe mit Farbblocken...
    locRectF = New RectangleF(GridSize.Width * 3, GridSize.Height * myUpperOffset, _
                           GridSize.Width * 12, GridSize.Height * 3)
    locBarWidth = locRectF.Width / BarColors.Length / 1
    For i As Integer = 0 To BarColors.Length - 1
        locBrush = New SolidBrush(myBarColors(i))
        locRectF.Width = locBarWidth
        g.FillRectangle(locBrush, locRectF)
        locRectF.X += locBarWidth
    Next

    'darunter liegende Balkenreihe mit Grauschattierungen...
    locRectF = New RectangleF(GridSize.Width * 3, GridSize.Height * (UpperOffset + 3), _
                           GridSize.Width * 12, GridSize.Height * 2)
    locBarWidth = locRectF.Width / BarShades.Length / 1
    For i As Integer = 0 To BarShades.Length - 1
        locBrush = New SolidBrush(myBarShades(i))
        locRectF.Width = locBarWidth
        g.FillRectangle(locBrush, locRectF)
        locRectF.X += locBarWidth
    Next

    'darunter liegende Balkenreihe mit Titel...
    locRectF = New RectangleF(GridSize.Width * 3, GridSize.Height * (UpperOffset + 5), _
                           GridSize.Width * 12, GridSize.Height)
    locBarWidth = locRectF.Width / BarShades.Length / 1

```

```

For i As Integer = 0 To BarTitleShades.Length - 1
    locBrush = New SolidBrush(myBarTitleShades(i))
    locRectF.Width = locBarWidth
    g.FillRectangle(locBrush, locRectF)
    locRectF.X += locBarWidth
Next

'darunter komplette zwei Reihen zunächst weiß...
locBrush = New SolidBrush(Color.White)
g.FillRectangle(locBrush, GridSize.Width * 3, GridSize.Height * (UpperOffset + 6), _
    GridSize.Width * 12, GridSize.Height * 2)

'jeweils zwei Farbverlaufsstreifen...
locRectF = New RectangleF(GridSize.Width * 3, GridSize.Height * (UpperOffset + 8), _
    GridSize.Width * 8, GridSize.Height)
locBrush = New LinearGradientBrush(locRectF, Color.Magenta, Color.Black, _
    LinearGradientMode.Horizontal)
g.FillRectangle(locBrush, locRectF)

'der zweite Farbverlaufsstreifen...
locRectF = New RectangleF(GridSize.Width * 3, GridSize.Height * (UpperOffset + 9), _
    GridSize.Width * 8, GridSize.Height)
locBrush = New LinearGradientBrush(locRectF, Color.Blue, Color.Black, LinearGradientMode.Horizontal)
g.FillRectangle(locBrush, locRectF)

'grauer Kasten neben die Farbverläufe...
locBrush = New SolidBrush(Color.FromArgb(130, 130, 130))
g.FillRectangle(locBrush, GridSize.Width * 11, GridSize.Height * (UpperOffset + 8), _
    GridSize.Width * 4, GridSize.Height * 2)

'Geriffele malen...
'Zuerst grauer Kasten als Unterlage
g.FillRectangle(locBrush, GridSize.Width * 4.5F, GridSize.Height * (UpperOffset + 6), _
    GridSize.Width * 10.5F, GridSize.Height)

'dann das linke, größere Geriffele malen...
locRectF = New RectangleF(GridSize.Width * 5, GridSize.Height * (UpperOffset + 6), _
    GridSize.Width * 1.5F, GridSize.Height)

drawAreaCorrugated(g, locRectF, Color.Black, Color.LightGray, 2, GridSize.Width)

'das kleinere Geriffele rechts daneben malen...
locRectF.X = GridSize.Width * 6.75F
locRectF.Width = GridSize.Width * 1.75F
drawAreaCorrugated(g, locRectF, Color.Black, Color.LightGray, 1, GridSize.Width)

'das noch kleinere Geriffele rechts daneben malen...
locRectF.X = GridSize.Width * 9.5F
locRectF.Width = GridSize.Width * 1.75F
drawAreaCorrugated(g, locRectF, Color.Black, Color.LightGray, 0.5F, GridSize.Width)

'das Olivenfarbene daneben...
locRectF.X = GridSize.Width * 11.5F

```

```

    locRectF.Width = GridSize.Width * 3
    locBrush = New SolidBrush(Color.Olive)
    g.FillRectangle(locBrush, locRectF)

    'Zielkreuz Kreis in die Mitte...
    locPen = New Pen(Color.White, 3)
    g.DrawEllipse(locPen, ClientSize.Width \ 2 - ClientSize.Height \ 2, 0, ClientSize.Height, _
                  ClientSize.Height)
    locPen.Width = 1
    g.DrawLine(locPen, GridSize.Width * 9, GridSize.Height * (UpperOffset + 3), _
               GridSize.Width * 9, GridSize.Height * (UpperOffset + 7))
    g.DrawLine(locPen, GridSize.Width * 5, GridSize.Height * (UpperOffset + 5), _
               GridSize.Width * 13, GridSize.Height * (UpperOffset + 5))

    'das kleine Dreieck in den weißen Zwischenraum...
    Dim locPoints(2) As PointF
    locPoints(0) = New PointF(GridSize.Width * 8.75F, GridSize.Height * (UpperOffset + 7))
    locPoints(1) = New PointF(locPoints(0).X, GridSize.Height * (UpperOffset + 8))
    locPoints(2) = New PointF(GridSize.Width * 9.25F, locPoints(0).Y)
    g.FillPolygon(New SolidBrush(Color.Black), locPoints)

    'Texte hineinschreiben.
    drawStringInFrame(g, New SolidBrush(Color.White), "VB.NET", "Arial", _
                      New RectangleF(GridSize.Width * 5.5F, GridSize.Height * (UpperOffset + 5), _
                                     GridSize.Width * 3, GridSize.Height))

    drawStringInFrame(g, New SolidBrush(Color.White), "Testbild", "Arial", _
                      New RectangleF(GridSize.Width * 9, GridSize.Height * (UpperOffset + 5), _
                                     GridSize.Width * 3.5F, GridSize.Height))

```

End Sub

Diese Prozedur bildet den Kern des Programms – sie sorgt dafür, dass das Bild auch tatsächlich auf dem Bildschirm erscheint. Anhand dieser Routine können Sie sehen, wie Füllungen und Linienfiguren mit dem GDI+ gezeichnet werden, und Sie können ebenfalls erkennen, wie einfach die Handhabung von Grafikfunktionen mit dem GDI+ im Grunde genommen ist.

Komplexe Grafikfunktionen, wie beispielsweise das Zeichnen des »geriffelten« Bereiches oder das Platzieren des Textes in der richtigen Größe, sind in verschiedene Unterroutinen ausgelagert, die Sie im Folgenden beschrieben finden:

```

'Geriffelten Bereich malen.
Private Sub drawAreaCorrugated(ByVal g As Graphics, ByVal rectF As RectangleF, _
                               ByVal col1 As Color, ByVal col2 As Color, _
                               ByVal relCellStepWidth As Single, ByVal relCellWidth As Single)

    Dim locPCol1 As New SolidBrush(col1)
    Dim locPCol2 As New SolidBrush(col2)
    Dim locCurrentBrush As Brush
    Dim locAltFlag As Boolean = False
    Dim locStep As Single = relCellWidth / (1 / relCellStepWidth * 15)
    For x As Single = rectF.X To rectF.Right Step locStep
        locCurrentBrush = DirectCast(IIf(locAltFlag, locPCol1, locPCol2), SolidBrush)
        g.FillRectangle(locCurrentBrush, x, rectF.Y, locStep, rectF.Height)
    Next
    locAltFlag = Not locAltFlag

```

```

locAltFlag = Not locAltFlag
Next

End Sub

'Zeichnet den Text so, dass er genau in ein Rechteck passt.
Private Sub drawStringInFrame(ByVal g As Graphics, ByVal brush As Brush, ByVal text As String, _
    ByVal fontName As String, ByVal rectF As RectangleF)

    'Skalierungseinstellungen zum Wiederherstellen speichern.
    Dim locGState As GraphicsState = g.Save
    'Font mit 12 Pt. Höhe aus Fontnamen anlegen.
    Dim locFont As New Font(fontName, 12, FontStyle.Regular)
    'Ausmaße des Strings messen.
    Dim locSize As SizeF = g.MeasureString(text, locFont)
    'Faktoren für die Skalierung ermitteln, sodass der String...
    Dim locScaleV As Single = rectF.Height / locSize.Height
    '....genau in das angegebene Rechteck passt.
    Dim locScaleH As Single = rectF.Width / locSize.Width
    'Koordinatensystem verschieben
    g.TranslateTransform(rectF.X, rectF.Y)
    'KoordinatenSystem neu skalieren
    g.ScaleTransform(locScaleH, locScaleV)
    'String im skalierten Koordinatensystem ausgeben.
    g.DrawString(text, locFont, brush, 0, 0)
    'Alte Skalierungs- und Transformationseinstellungen wiederherstellen.
    g.Restore(locGState)

```

End Sub

Diese letzte Prozedur demonstriert den Einsatz von Text- und Skalierungsfunktionen. Für deren genaues Verständnis ist allerdings ein klein wenig mehr Hintergrundwissen erforderlich, das Ihnen der folgende kurze Exkurs liefert.

## Exaktes Einpassen von Text mit GDI+-Skalierungsfunktionen

Um eine Zeichenkette in ein Graphics-Objekt auszugeben, benötigen Sie neben einem Font-Objekt, das den zu verwendenden Zeichensatz bestimmt, auch ein Brush-Objekt, das die Pinseleigenschaften darstellt und damit Farbe und Füllung definiert, mit denen der Text gezeichnet wird. Für die Größenbestimmung der auszugebenden Zeichenkette ist normalerweise ausschließlich die Größe des Zeichensatzes verantwortlich; GDI+ bietet Ihnen standardmäßig keine Möglichkeit, einen auszugebenden Text automatisch auf eine bestimmte Höhe oder Breite zu skalieren. Wenn Sie einen Text mit der DrawString-Funktion in ein Graphics-Objekt hineinschreiben, hat er damit genau die Größe, die sich durch das angegebene Font-Objekt und die Breite der im auszugebenden Text enthaltenen Buchstaben ergibt.

Allerdings bietet das GDI+ eine Reihe von Skalierungsfunktionen, mit denen das Koordinatensystem in beide Richtungen gedehnt oder gestaucht werden kann. ►

In Zusammenarbeit mit der `MeasureString`-Funktion, die die Ausmaße eines Textes ermitteln kann, ohne ihn dabei wirklich zu auszugeben, erlaubt das die genaue Dehnung/Stauchung des Koordinatensystems, sodass der Text – egal mit welchen Fonteinstellungen gezeichnet – exakt in die Ausmaße eines bestimmten rechteckigen Bereichs eingepasst werden kann. Die Verfahrensweise dabei ist relativ einfach:

Der Text wird zunächst mit `MeasureString` vermessen; dabei werden seine Höhe und seine Breite in Pixel<sup>3</sup> ermittelt. Die Skalierungsfaktoren für den Text ergeben sich jetzt aus einer simplen Division der Ausmaße des Zielrechteckes durch die Ausmaße des den Text tatsächlich umschließenden Rechtecks.

Damit die Skalierung für das `Graphics`-Objekt später wieder auf seine Ursprungseinstellung zurückgestellt werden kann, kann die aktuelle Skalierungseinstellung mit der `Save`-Methode in einem so genannten `GraphicsState`-Objekt gespeichert werden. Erst jetzt werden die errechneten Skalierungsfaktoren mithilfe der Methode `TranslateTransform` für das aktuelle `Graphics`-Objekt bestimmt. Da der zuvor gemessene Text derselbe ist, der nun mit `DrawString` in das `Graphics`-Objekt ausgegeben wird, und die Skalierung durch Anwenden von `TranslateTransform` umgestellt wurde, passt er exakt in das Zielrechteck.

Damit alle weiteren Zeichenfunktionen unskaliert stattfinden können, wird die ursprüngliche Skalierung durch die `Restore`-Methode zu guter Letzt wieder in den Ausgangszustand zurückgesetzt.

Die einzige Zeichenroutine, die jetzt noch fehlt, ist die zum Zeichnen des Gitters. Im folgende Code-listing werden Sie feststellen: Die `paintGrid`-Methode ist so ausgelegt, dass sie wahlweise ein Raster oder ein Gitter zeichnen kann. Ein Rasterpunkt besteht dabei allerdings nicht aus einem einzelnen Pixel, sondern aus einem kleinen Rechteck. Das GDI+ stellt keine Funktion zur Verfügung, mit der ein einzelner Punkt gezeichnet werden kann – das ist der Hintergrund. Alternativ könnten Sie auch eine Linie mit gleichem Anfangs- und Endpunkt zeichnen, das Ergebnis wäre ähnlich:

```
'Zeichnet das Raster oder das Gitter.  
Private Sub paintGrid(ByVal g As Graphics)  
    Dim locPen As New Pen(GridColor, GridLineWidth)  
    Dim locBrush As New SolidBrush(GridColor)  
  
    If GridStyle = GridStyle.Dots Then  
        For x As Single = 0 To ClientSize.Width Step GridSize.Width  
            For y As Single = 0 To ClientSize.Height Step GridSize.Height  
                g.FillRectangle(locBrush, x, y, GridLineWidth, GridLineWidth)  
            Next  
        Next  
  
    Else  
        For x As Single = 0 To ClientSize.Width Step GridSize.Width  
            g.DrawLine(locPen, x, 0, x, ClientSize.Height)  
        Next
```

---

<sup>3</sup> Vorausgesetzt, Sie haben die verwendete Maßeinheit für das aktuelle `Graphics`-Objekt nicht zuvor mit seiner `PageUnit`-Eigenschaft auf einen anderen Wert gesetzt.

```

For y As Single = 0 To ClientSize.Height Step GridSize.Height
    g.DrawLine(locPen, 0, y, ClientSize.Width, y)
Next
End If
End Sub

```

Übrigens: Wenn Sie das Formular vergrößern oder verkleinern, löst das nicht notwendigerweise ein Paint-Ereignis aus. Das Paint-Ereignis wird nur beim Vergrößern des Formulars ausgelöst, und Sie können mit dem Graphics-Objekt, das jetzt übergeben wird, nur den Bereich erneuern, der durch das Vergrößern auch wirklich neu gezeichnet werden müsste. Das ergibt Sinn bei Inhalten, die statisch sind – also nicht durch die Größe des Formulars beeinflusst werden. Bei einer Tabellenkalkulation beispielsweise ist der dargestellte Inhalt eines Fensters nicht von seiner Größe abhängig; nur der dargestellte Ausschnitt verändert sich mit dem Vergrößern des Fensters.

In unserem Beispiel ist das anders. Wenn das Formular vergrößert oder verkleinert wird, muss der *komplette* Formularinhalt neu gezeichnet werden. Aus diesem Grund schaut die Prozedur, die dieses Ereignis verarbeitet, folgendermaßen aus:

```

'Wird aufgerufen, wenn das Formular in seiner Größe verändert wird.
Protected Overrides Sub OnResize(ByVal e As System.EventArgs)
    Me.Invalidate()
End Sub

```

Invalidate bewirkt, dass der gesamte Bereich auf den es angewendet wird, hier `Me` – also das Formular selbst –, als »ungültig« gekennzeichnet wird. Dabei wird dann das Paint-Ereignis ausgelöst. Der Formularinhalt wird also komplett neu gezeichnet.

Und jetzt, nachdem Sie wissen, wie das Programm funktioniert: Stürzen Sie sich hinein in den Quellcode, und experimentieren Sie mit den Einstellungen, die im Konstruktor des Formulars vorgenommen werden. Sie werden sehen, dass Sie durch Ausprobieren der verschiedenen Einstellungen der Objekte, die Graphics anbietet, am besten den Umgang mit dem GDI+ lernen können!

## Flimmerfreie, fehlerfreie und schnelle Darstellungen von GDI+-Zeichnungen

Das Beispielprogramm aus dem vorherigen Abschnitt hat die Leistungsfähigkeit des GDI+ eindrucksvoll demonstriert. Allerdings hat es ebenso gezeigt, dass noch einige Handgriffe notwendig sind, um die Darstellung wirklich zu perfektionieren, denn:

- Die Bilddarstellung flimmert heftig, wenn der Anwender das Testbild vergrößert oder verkleinert.
- Das Formular sollte sich nur proportional vergrößern lassen (Festes Seitenverhältnis in X- und Y-Richtung), damit man feststellen kann, ob der Monitor einen Kreis auch wirklich rund darstellt. Das Seitenverhältnis sollte sich durch eine Eigenschaft einstellen lassen.
- Der Bildaufbau ist gerade beim Vergrößern und Verkleinern des Formulars vergleichsweise langsam.
- Und falls Sie ganz genau hingeschaut haben, werden Sie bemerkt haben, dass die Linienstärke bei den äußeren Linien und bei den Berührungs punkten des Kreises mit den Formularändern berücksichtigt werden sollte.

Diese Liste von Unzulänglichkeiten ist typisch für grafische Inhalte, die in Fenstern unter Windows dargestellt werden. Selbst ein Programm wie der Windows-Explorer, von dem man meinen könnte, es sei nach Jahren wirklich ausgereift, flimmert beim Vergrößern oder Verkleinern munter vor sich hin.<sup>4</sup> Die nächsten Abschnitte sollen Ihnen helfen, die richtige Vorgehensweise zu finden, um Ihren Formularen und (später auch) Steuerelementen ein professionelles Aussehen zu verleihen.

## Zeichnen ohne Flimmern

Bei dem ersten Problem hilft das Framework mit einem Prinzip, nach dem man schon seit Jahren und nicht erst seit Windows verfährt, um Flimmern bei der Darstellung von bewegten Bildern zu vermeiden. Die Technik ist so simpel wie genial: Anstatt ein Bild zu löschen und es verändert neu zu zeichnen, komplettiert man es zunächst in einer unsichtbaren Bitmap im Speicher. Wenn das Bild komplett gezeichnet wurde, kopiert man es als Ganzes in den sichtbaren Anzeigebereich. So ist das eigentliche Entstehen des Bildes unsichtbar und damit flimmerfrei. Darüber hinaus muss das Bild für das Neuzeichnen nicht gelöscht werden (was ein Großteil des Flimmerns verursacht), denn wenn eine entsprechende Instanz das komplett fertige Bild in das zu überschreibende, sichtbare Bild kopiert, wird sowieso jeder einzelne Pixel des alten Bildes gelöscht. Ein Löschen des Hintergrunds wäre also total überflüssig.

Wie schon angedeutet, müssen Sie diese Funktionalität nicht selber implementieren, da sie das Framework schon fix und fertig bereithält. Sie müssen dem Framework lediglich mitteilen, dass Sie die Sonderbehandlung des »Hintergrund-Löschen-Ereignisses«, das von Windows ausgelöst wird, nicht wünschen, sondern die gerade beschriebene Methode – sie nennt sich auf neudeutsch *Double Buffering* (Doppelpufferung) – anwenden möchten.

---

**BEGLEITDATEIEN:** Sie finden das modifizierte Testbild-Projekt im Verzeichnis .\VB 2005 - Entwicklerbuch\G - Smart-Client\Kap29\VBTestbild02\

---

Sie erreichen das mit zwei simplen Zeilen, die Sie im Konstruktor hinterlegen:

```
Public Sub New()
    MyBase.New()

    ' Dieser Aufruf ist für den Windows Form-Designer erforderlich.
    InitializeComponent()

    Me.SetStyle(ControlStyles.DoubleBuffer, True)
    Me.SetStyle(ControlStyles.AllPaintingInWmPaint, True)
```

Wenn Sie das Programm nach dieser Modifizierung starten, ist es vorbei mit dem Flimmern. Sie können das Formular nach Belieben vergrößern oder verkleinern, ohne beim Neuaufbau zuschauen zu müssen.

---

<sup>4</sup> Achten Sie mal beim Vergrößern oder Verkleinern auf das TreeView-Steuerelement des Explorers, das die Laufwerke darstellt.

---

**HINWEIS:** Neu in Visual Basic 2005 bzw. im Framework 2.0 ist eine Eigenschaft der Basisklasse Control namens DoubleBuffered. Das Setzen dieser Eigenschaft bewirkt letzten Endes dasselbe wie Me.SetStyle(ControlStyles.DoubleBuffer, True) zusammen mit Me.SetStyle(ControlStyles.AllPaintingInWmPaint, True). Der neu eingeführte Stil ControlStyles.OptimizedDoubleBuffer verspricht mehr, als er hält: Das Durchführen des internen Double Buffering funktioniert auf die gleiche Weise, wie beim Einstellen des Double Buffering auf die ursprünglichen Art über die beiden SetStyle-Methoden. Es ist also anzunehmen, dass das OptimizedBuffering-Flag nur aus Abwärtskompatibilitätsgründen eingeführt wurde, denn nur auf dieses Flag wird Framework-intern durch die DoubleBuffered-Eigenschaft zurückgegriffen.

---

## Eigenschaften von Formularen und Control-Ableitungen mit SetStyle definieren

SetStyle dient übrigens nicht nur dazu, das *Double Buffering* ein- und auszuschalten. Auch andere Verhaltensweisen einer von Control abgeleiteten Klasse (dazu gehört auch ein Formular) lassen sich mit dieser Methode steuern.

Die grundsätzliche Verwendung von SetStyle funktioniert folgendermaßen:

```
Me.SetStyle(ControlStyles.Style1 or ControlStyle2 or ..., True|False)
```

Als ersten Parameter übergeben Sie SetStyle eine Kombination aus Elementen der ControlStyles-Enum. Der zweite Parameter bestimmt, ob die entsprechenden Flags ein- bzw. ausgeschaltet werden sollen. Welche Flags dabei welche Aufgaben übernehmen, zeigt die folgende Tabelle. Bitte verwenden Sie in diesem Falle nicht die Visual Studio-Online-Hilfe, da sie die Aufgaben der einzelnen Flags recht schwammig, teilweise sogar missverständlich erklärt.

Member-Name	Wert	Beschreibung
ContainerControl	1	Die auf Control basierende Komponente ist ein Container-Control. Dieses Flag ist automatisch gesetzt, wenn Sie eine Klasse aus ScrollableControl ableiten.
UserPaint	2	Definiert, dass die Ereignisse <i>OnPaint</i> und <i>OnBackgroundPaint</i> von <i>WndProc</i> ausgelöst werden. Dieses Flag ist standardmäßig gesetzt. Wenn Sie dieses Flag löschen, müssen Sie <i>WndProc</i> überschreiben und die entsprechenden Nachrichten auswerten, um die notwendigen Maßnahmen zum Zeichnen des Fensterinhaltes zu ergreifen.
Opaque	4	Wenn Sie dieses Flag setzen, wird <i>OnPaintBackground</i> nicht ausgelöst und der Hintergrund damit nicht gezeichnet.
ResizeRedraw	16	Bestimmt, dass <i>OnResize</i> , das bei Größenveränderung des Controls/Formulars automatisch aufgerufen wird, ein <i>Invalidate</i> auslöst und damit automatisch das Neuzeichnen des Client-Bereichs erzwingt.
FixedWidth	32	Formulare lassen sich durch die <i>AutoSize</i> -Eigenschaft so einrichten, dass sie sich automatisch an eine neue Schriftart anpassen, die ihnen durch die <i>Font</i> -Eigenschaft zugewiesen wird. Dadurch wird das Formular selbst und auch alle in ihm enthaltenen Steuerelemente entsprechend der Größe des verwendeten Fonts vergrößert oder verkleinert. <sup>5</sup> Möchten Sie verhindern, dass die Skalierung der Formularbreite stattfindet, setzen Sie dieses Flag. Mit diesem Flag können Sie <i>nicht</i> festlegen, dass ein Formular oder Control in seiner Breite nicht verändert werden darf. ►

---

<sup>5</sup> Beachten Sie, dass dieses Verhalten nur beim Erstellen des Formulars funktioniert, also wenn der Font noch im Konstruktor verändert wird.

Member-Name	Wert	Beschreibung
FixedHeight	64	Es gilt das für <i>FixedWidth</i> Gesagte, nur für die Höhe des Formulars.
StandardClick	256	Dieses Flag ist standardmäßig gesetzt. Wenn Sie möchten, dass das Control oder das Formular kein <i>Click</i> -Ereignis auslösen soll, löschen Sie dieses Flag. WICHTIG: Wenn Sie dieses Flag löschen, löst das Control/Formular auch kein <i>DoubleClick</i> -Ereignis mehr aus.
Selectable	512	Dieses Flag ist für Formulare und Controls standardmäßig gesetzt. Es bestimmt, ob entsprechende Ereignisse von Windows überhaupt zur Fokussierung des Controls/Formulars führen können. Beachten Sie, dass <i>Container Controls</i> ( <i>ScrollableControl</i> , <i>ContainerControl</i> ) den Fokus nicht erhalten können, wenn sie andere Controls beinhalten. Das gilt unabhängig von der Einstellung dieses Flags. Beachten Sie auch, dass von <i>Control</i> abgeleitete Klasseninstanzen standardmäßig die <i>Fokusbenachrichtigung durch Mausklickaktivierung nicht erhalten</i> , wenn das folgende <i>UserMouse</i> -Flag nicht gesetzt ist.
UserMouse	1024	Dieses Flag muss gesetzt werden, damit eine von <i>Control</i> abgeleitete Instanz Fokussierungsnachrichten erhalten kann, wenn diese durch Mausklick ausgelöst wurden. Bitte beachten Sie, dass entgegen den Angaben in der Online-Hilfe das Setzen dieses Flags nicht dazu führt, dass Sie Mausereignisse von Grund auf neu implementieren müssen! <i>OnMouseXXX</i> , <i>OnClick</i> und <i>OnDoubleClick</i> werden nach wie vor ausgeführt!
SupportsTransparentBackColor	2048	Direkte Ableitungen von <i>Control</i> unterstützen normalerweise keine transparenten Hintergrundfarben. Mit Hilfe dieses Flags können Sie die Unterstützung explizit einschalten. Anschließend akzeptiert die <i>Control</i> -Ableitung zum Simulieren von Transparenz eine Hintergrundfarbe mit einer Alpha <sup>6</sup> -Komponente, die kleiner als 255 ist. Für Formulare ist dieses Flag standardmäßig gesetzt; Sie können die Farbe, die die Transparenz bestimmt, mit der <i>TransparencyKey</i> -Eigenschaft definieren.
StandardDoubleClick	4096	Dieses Flag ist standardmäßig gesetzt. Wenn Sie möchten, dass das Control oder das Formular kein <i>DoubleClick</i> -Ereignis auslösen sollen, löschen Sie dieses Flag. Auch wenn Sie das <i>StandardClick</i> -Flag löschen, erhält das Control/Formular kein <i>DoubleClick</i> -Ereignis mehr.
AllPaintingInWmPaint	8192	Normalerweise löst Windows für Controls und Formulare eine <i>WM_ERASEBKND</i> -Nachricht bei der Notwendigkeit des Neuzeichnens eines Client-Bereichs aus. Die Behandlung dieser Nachricht führt dazu, dass der Client-Bereich gelöscht – also mit einer bestimmten Farbe komplett ausgefüllt wird. In vielen Fällen führt das zu unerwünschten Flimmereffekten beim Neuzeichnen des Client-Bereichs. Setzen Sie dieses Flag, ignoriert das Steuerelement die <i>WM_ERASEBKND</i> -Fenstermeldung, um das Flimmen zu verringern.
CacheText	16384	Setzen Sie dieses Flag, bewahrt das Steuerelement eine Kopie des Textes auf, sodass dieser nicht jedes Mal, wenn er benötigt wird, aus dem Windows-Fenster abgerufen werden muss. Dieses Flag ist standardmäßig nicht gesetzt. Dieses Verhalten verbessert die Leistung, erschwert jedoch die Textsynchronisierung.
EnableNotifyMessage	32768	Setzen Sie dieses Flag, wird <i>OnNotifyMessage</i> für jede Meldung aufgerufen, die aus der Nachrichtenwarteschlange an <i>WndProc</i> des Controls bzw. Formulars gesendet wird. Dieses Flag ist standardmäßig nicht gesetzt.

<sup>6</sup> Die Alpha-Komponente bei einer Farbangabe bestimmt die »Durchscheinstärke«. 255 entspricht »voll deckend«, 0 entspricht »voll durchscheinend«.

Member-Name	Wert	Beschreibung
DoubleBuffer	65536	Schaltet das DoubleBuffering ein. Parallel dazu sollten Sie <i>AllPaintingInWmPaint</i> ebenfalls setzen, damit das <i>Double Buffering</i> in Kraft treten kann.
OptimizedDoubleBuffer	131072	Wenn Sie diese Eigenschaft auf True festlegen, müssen Sie auch <i>AllPaintingInWmPaint</i> auf True festlegen, um das Double Buffering beim Zeichnen zu aktivieren. <b>Hinweis:</b> Dieses Flag wird von der im Framework 2.0 neu vorhandenen DoubleBuffered-Eigenschaft verwendet und ist vermutlich nur aus Gründen der Abwärtskompatibilität vorhanden, da beim Zeichnen eines Steuerelements in seiner internen WmPaint-Routine bei gesetztem OptimizedDoubleBuffer-Flag nichts anderes passiert, als beim Setzen des »einfachen« DoubleBuffer-Flags.
UseTextForAccessibility	262144	Gibt an, dass der Wert der Text-Eigenschaft bei Steuerelementen, sofern festgelegt, den Namen und die Tastenkombination für aktive Eingabehilfen des Steuerelements bestimmt.

**Tabelle 29.1:** Die Einstellungen der ControlStyles-Enum

Mit dem Wissen um die Funktionsweisen dieser Flags können wir mit einem kleinen Handgriff das Programm weiter optimieren. Durch das Setzen des Flags *ResizeRedraw* kümmert sich schon die Basisklasse um den Aufruf von *Invalidate* beim Vergrößern oder Verkleinern des Formulars. Durch Einfügen einer weiteren Zeile

```
Me.setStyle(ControlStyles.ResizeRedraw, True)
```

im Konstruktor des Programms wird damit die Behandlung des kompletten *Resize*-Ereignisses überflüssig. Es ist in dieser Version des Programms auch nicht mehr vorhanden.

Was das Thema Geschwindigkeit anbelangt: Wenn Sie das Programm in dieser Version starten, bemerken Sie, dass es gar nicht so langsam läuft, wie es ursprünglich den Anschein hatte. Durch das Eliminieren des Flimmerns erkennen Sie die wahre Geschwindigkeit, mit der man auch auf langsameren Maschinen durchaus leben kann.

## Programmtechnisches Bestimmen der Formulargröße

Es gibt zahlreiche Anwendungen, bei denen es notwendig ist, ein Formular oder ein Steuerelement zur Laufzeit auf eine bestimmte Breite oder eine bestimmte Höhe zu beschränken. In unserem Beispiel ergibt es beispielsweise Sinn, eine Eigenschaft einzuführen, die das Seitenverhältnis reglementiert. Nur wenn das Seitenverhältnis des Formulars auch dem Seitenverhältnis des verwendeten Monitors entspricht, können Sie beurteilen, ob der Monitor einen Kreis auch wirklich als Kreis darstellt.

Mit der *SetBoundsCore*-Methode eines Formulars oder eines Control-Derivats bestimmen Sie dessen Ausmaße. Allerdings können Sie die Größeneinstellungen nicht im *Resize*-Ereignis vornehmen, da das Neupositionieren des Fensters schon vor dem Auslösen des Ereignisses geschieht. Das Ergebnis wäre ein heftiges Flimmern. Was wir bräuchten, wären weitere Ereignisse, mit denen wir erkennen könnten, wann ein bestimmter Vorgang wie das Verschieben oder das Vergrößern bzw. Verkleinern eines Fensters beginnt und wann er abgeschlossen ist.

Seit der Version 2.0 des Frameworks gibt es zwei Ereignisse, die Sie dabei unterstützen. Diese Ereignisse nennen sich *ResizeBegin* und *ResizeEnd*. Wenn das Vergrößern eines Formulars abgeschlossen

wurde, wird das ResizeEnd-Ereignis ausgelöst, und wir können das Formular hier auf das richtige Seitenverhältnis einstellen.

Leider verhält sich diese Sache nicht ganz so einfach. Denn das ResizeEnd-Ereignis wird auch dann ausgelöst, wenn Sie das Formular nur verschoben haben. Warum das so ist, ist mir ehrlich gesagt schleierhaft – im *Product Feedback Center* von Visual Studio gibt es aber bereits einen Vorschlag, das Verhalten dieses Ereignisses zu überarbeiten.

Um dem Programm abzugewöhnen, das Formular auch beim Verschieben anzupassen, behelfen wir und deswegen mit einem kleinen Trick, der deutlich wird, wenn Sie sich die Implementierung der entsprechenden Codezeilen anschauen.

---

**BEGLEITDATEIEN:** Sie finden das modifizierte Testbild-Projekt im Verzeichnis .\VB 2005 - Entwicklerbuch\G - Smart-Client\Kap29\VBTestbild03\

---

```
'Wird zu Beginn einer Größenänderung aufgerufen
Protected Overrides Sub OnResizeBegin(ByVal e As System.EventArgs)
    MyBase.OnResizeBegin(e)
    'Das Flag zurücksetzen, das festhält, dass das Formular verschoben wurde
    myJustMoved = False
End Sub

'Wird aufgerufen, wenn eine Größenänderung abgeschlossen wurde
Protected Overrides Sub OnResizeEnd(ByVal e As System.EventArgs)
    'Nach Abschluss des Resize-Vorgangs Seitenverhältnis berücksichtigen,
    'aber nur, wenn dieses Ereignis nicht durch das Verschieben des Formulars
    'ausgelöst wurde.
    If Not myJustMoved Then
        'Neue Größe des Formulars im richtigen Seitenverhältnis:
        MyBase.SetBoundsCore(Location.X, Location.Y, _
            CInt(Size.Height * XYRatio), Size.Height, BoundsSpecified.Width)
    End If
    MyBase.OnResizeEnd(e)
End Sub

'Wird beim Verschieben des Formulars aufgerufen
Protected Overrides Sub OnMove(ByVal e As System.EventArgs)
    MyBase.OnMove(e)
    'Das Formular wurde verschoben; ResizeEnd wird am Ende auch
    'ausgelöst, darf aber nicht berücksichtigt werden!
    myJustMoved = True
End Sub
```

Mit dieser Änderung können wir die Möglichkeit zur Einstellung des Seitenverhältnisses nun als Eigenschaft in die neue Programmversion einbauen und im nunmehr vorhandenen ResizeEnd-Ereignis dafür sorgen, dass das Formular nach Abschluss des Größenveränderungsvorgangs richtig positioniert wird:

```

Public Property XYRatio() As Single
    Get
        Return myXYRatio
    End Get
    Set(ByVal Value As Single)
        myXYRatio = Value
    End Set
End Property

```

Diese Eigenschaft wird im Konstruktor auf ein Seitenverhältnis von 4 : 3 gesetzt – wie es den meisten Monitoren entspricht:<sup>7</sup>

```

Public Sub New()
    .
    .
    .
    'Seitenverhältnis definieren
    XYRatio = 4 / 3

    'für 19"-TFTs mit 1280x1024:
    'XYRatio = 5 / 4
    .
    .
    .

```

## Was Sie beim Zeichnen von breiten Linienzügen beachten sollten

Wenn Sie das Testbild in Abbildung 29.2 genauer betrachten, werden Sie feststellen, dass die äußereren Linien nicht richtig zu sehen sind. Das gilt für die Linien des Rasters genau so wie für die Berührungspunkte des Kreises. Die Gründe dafür soll das folgende kleine Projekt demonstrieren.

---

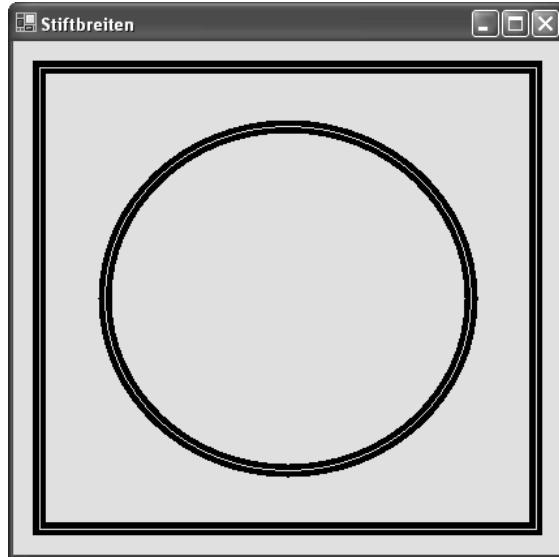
**BEGLEITDATEIEN:** Sie finden dieses Projekt für dieses Beispiel im Verzeichnis .\VB 2005 - Entwicklerbuch\G - Smart-Client\Kap29\PenWidthDemo01\

---

Wenn Sie das Programm starten, sehen Sie im Formular eine Grafik, die in etwa der Abbildung 29.3 entspricht.

---

<sup>7</sup> Ausnahmen bilden 17“, 19“ und 20“-TFT-Displays mit einer Auflösung von 1280 x 1024 Pixel, die ein Seitenverhältnis von 5 : 4 aufweisen.



**Abbildung 29.3:** Wenn Sie Stiftstärken größer als ein Pixel verwenden, dann müssen Sie davon ausgehen, dass die weiteren Pixel einer Linie gleichmäßig um den eigentlich Pixel herum verteilt sind. In dieser Grafik haben die gelben und schwarzen Figuren die gleichen Ausmaße, aber andere Stiftstärken.

Die Abbildung zeigt es deutlich: Die Pixel des breiteren Stiftes der schwarzen Figuren verteilen sich um die ein Pixel breiten Linien des gelben Stiftes, und zwar auf allen Seiten zu genau gleichen Teilen. Wenn Sie also beispielsweise einen Rahmen voll sichtbar in den Client-Bereich eines Formulars zeichnen wollen, dann müssen Sie im Falle des Rechtecks die Hälfte der Linienstärke in die Koordinaten mit einrechnen. Zur Verdeutlichung: Der Code, der diese Figuren ins Formular malt, sieht folgendermaßen aus:

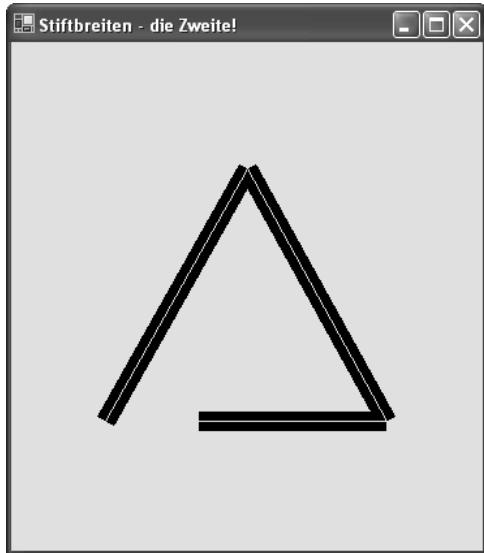
```
'Zeichnet jeweils ein Rechteck in einer dicken, schwarzen und einer dünnen, gelben Umrandung.  
Protected Overrides Sub OnPaint(ByVal e As System.Windows.Forms.PaintEventArgs)  
    Dim locSchwarzerStift As New Pen(Color.Black, 10)  
    Dim locGelberStift As New Pen(Color.Yellow, 1)  
    Dim locFürRechteck As New Rectangle(20, 20,  
                                         ClientSize.Width - 40, ClientSize.Height - 40)  
    Dim locOffsetDrittel As New Size(ClientSize.Width \ 6, ClientSize.Height \ 6)  
    Dim locFürKreis As New Rectangle(locOffsetDrittel.Width,  
                                    locOffsetDrittel.Height,  
                                    ClientSize.Width - 2 * locOffsetDrittel.Width,  
                                    ClientSize.Height - 2 * locOffsetDrittel.Height)  
    e.Graphics.DrawRectangle(locSchwarzerStift, _  
                            locFürRechteck)  
    e.Graphics.DrawRectangle(locGelberStift, _  
                            locFürRechteck)  
    e.Graphics.DrawEllipse(locSchwarzerStift, _  
                          locFürKreis)  
    e.Graphics.DrawEllipse(locGelberStift, _  
                          locFürKreis)  
End Sub
```

Noch problematischer wird es, wenn Sie Linienzüge aufbauen wollen – beispielsweise, wenn Sie ein zu einer Seite offenes Dreieck aus drei verschiedenen Linien zusammensetzen müssen; beim Zeichnen von breiten Linienzügen gibt es nämlich unschöne Überschneidungen, wenn diese aus einzelnen Linien aufgebaut sind, wie das folgende Beispiel zeigt:

---

**BEGLEITDATEIEN:** Sie finden dieses Projekt im Verzeichnis .\VB 2005 - Entwicklerbuch\G - Smart-Client\Kap29\PenWidthDemo02

---



**Abbildung 29.4:** Bei unabhängigen Linien, die Linienzüge bilden sollen, ist das Verhalten bei großen Stiftstärken umso störender

In diesem Beispiel sollen drei Linien zu einer Figur – in diesem Falle zu einem zu einer Seite offenen Dreieck – verbunden werden; leider klappt das nur ansatzweise: Während es bei den inneren, ein Pixel breiten Linien keine Probleme gibt, sind die äußereren Linien als nicht wirklich miteinander verbunden erkennbar. Der Grund: Die jeweils nächste Linie weiß nichts davon, dass eine weitere Linie folgt, und dass die offenen Zwischenräume mit der Stiftfarbe (oder dem Muster, falls der Stift aus einem Pinsel entstanden ist) aufgefüllt werden sollten.

### Geschlossene Figuren mit Polygon und GraphicsPath

Um dieses Manko zu beheben, bietet das GDI+ das so genannte *GraphicsPath*-Objekt an. Das *GraphicsPath*-Objekt erlaubt das Erstellen von Linienzügen, die wirklich miteinander verbunden sind, und seine Verwendung ist denkbar einfach. Zu jeder herkömmlichen Malmethode des GDI+ gibt es ein Äquivalent, mit dem Sie einem *GraphicsPath* einen Linienzug hinzufügen können.

Angenommen, Sie haben mit der Anweisung

```
Dim locLinienverbund As New GraphicsPath
```

ein neues *GraphicsPath*-Objekt erstellt. In diesem Fall verwenden Sie zum Zeichnen einer Linie nicht die *DrawLine*-Methode, die Sie direkt auf das *Graphics*-Objekt anwenden, sondern die *AddLine*-Methode, die Sie auf das *GraphicsPath*-Objekt beziehen. Für andere Malmethoden gibt es ähnliche

AddXXX-Äquivalente. Auf diese Weise erstellen Sie den Linienzug zunächst, ohne ihn konkret zu zeichnen.

Haben Sie den Linienzug komplett aufgebaut, entscheiden Sie sich, ob Sie die Anweisung

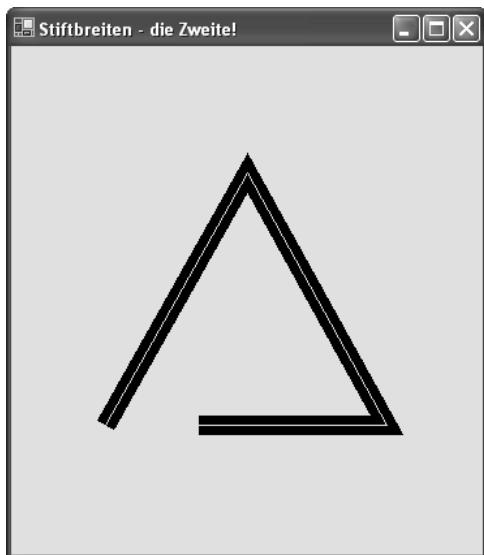
```
locLinienverbund.CloseFigure()
```

anwenden, die den letzten Punkt des Linienzugs automatisch mit dem Anfangspunkt des Linienzugs verbinde (für unser Beispiel nicht erwünscht).

Das eigentliche Zeichnen des Linienzugs passiert erst jetzt mit der Anweisung

```
g.DrawPath(locPen, locLinienverbund)
```

So können Sie die Linienzüge schließen, die wirklich geschlossen werden sollen. Ein Doppelklick in das Formular demonstriert diese Vorgehensweise. Das Ergebnis sehen Sie in Abbildung 29.5:



**Abbildung 29.5:** Mit dem *GraphicsPath*-Objekt erstellen Sie Linienzüge, bei denen die einzelnen Komponenten wirklich miteinander verbunden sind

Der Code, der beide Figuren in das Formular zaubert, sieht dabei auszugsweise folgendermaßen aus:

```
'Zeichnet jeweils ein Rechteck in einer dicken, schwarzen und einer
'dünnen, gelben Umrandung.
Protected Overrides Sub OnPaint(ByVal e As System.Windows.Forms.PaintEventArgs)
    'Nur um Tipparbeit zu sparen,
    Dim g As Graphics = e.Graphics
    'Hier werden die Eckpunkte des offenen Dreiecks gespeichert,
    Dim locDreiecksPunkte(3) As Point
    'Diese Variable dient nur dem Sparen der Tipparbeit,
    Dim locCB As Size = Me.ClientSize
    'Hier werden die Eckpunkte des Dreiecks definiert,
    Dim c As Integer
    'Zählvariable für die Schleife zum Zeichnen der Linien
    locDreiecksPunkte(0) = New Point(locCB.Width \ 5, (locCB.Height \ 4) * 3)
    locDreiecksPunkte(1) = New Point(locCB.Width \ 2, locCB.Height \ 4)
    locDreiecksPunkte(2) = New Point((locCB.Width \ 5) * 4, (locCB.Height \ 4) * 3)
```

```

locDreiecksPunkte(3) = New Point((locCB.Width \ 5) * 2, (locCB.Height \ 4) * 3)

'Zwei mögliche Malverfahren. Das ändert sich bei jedem Doppelklick:
If Not myDoppelklickTrigger Then
    'Einzelne Linien malen.
    Dim locPen As New Pen(Color.Black, 15)
    'Alle Eckpunkte per Linie miteinander verbinden.
    For c = 0 To locDreiecksPunkte.Length - 2
        g.DrawLine(locPen, locDreiecksPunkte(c).X, locDreiecksPunkte(c).Y, _
                   locDreiecksPunkte(c + 1).X, locDreiecksPunkte(c + 1).Y)
    Next

    'Das Gleiche nochmal in gelb und dünn.
    locPen = New Pen(Color.Yellow, 1)
    For c = 0 To locDreiecksPunkte.Length - 2
        g.DrawLine(locPen, locDreiecksPunkte(c).X, locDreiecksPunkte(c).Y, _
                   locDreiecksPunkte(c + 1).X, locDreiecksPunkte(c + 1).Y)
    Next

Else
    '...oder als Linienzug mithilfe des GraphicPaths-Objektes:
    Dim locPen As New Pen(Color.Black, 15)
    Dim locLinienverbund As New GraphicsPath
    For c = 0 To locDreiecksPunkte.Length - 2
        locLinienverbund.AddLine(locDreiecksPunkte(c).X, locDreiecksPunkte(c).Y, _
                               locDreiecksPunkte(c + 1).X, locDreiecksPunkte(c + 1).Y)
    Next

    'Mit dieser Anweisung würden Sie den Endpunkt des Linienzuges
    'mit dem Startpunkt verbinden und so die Figur schließen.
    'locFigur.CloseFigure()
    e.Graphics.DrawPath(locPen, locLinienverbund)
    locPen = New Pen(Color.Yellow, 1)
    e.Graphics.DrawPath(locPen, locLinienverbund)
    e.Graphics.DrawPath(locPen, locLinienverbund)
End If

End Sub

Protected Overrides Sub OnDoubleClick(ByVal e As System.EventArgs)
    myDoppelklickTrigger = Not myDoppelklickTrigger
    Invalidate()
End Sub

```

---

**HINWEIS:** Wenn Sie ein GraphicsPath-Objekt erstellen, dann ist dieses nicht auf nur eine Figur beschränkt. Mit Hilfe der Methode `StartFigure` können Sie innerhalb desselben `GraphicPaths` einen neuen Linienzug beginnen, ohne dass der alte zunächst geschlossen wird. `CloseFigure` hingegen schließt den aktuellen Linienzug (verknüpft also den letzten mit dem ersten Punkt). Einige Methoden, wie beispielsweise `AddEllipse`, fügen dem `GraphicsPath` einen geschlossenen Linienzug direkt hinzu.

---

## Abschließende Änderungen am Testbild

Mit diesem Wissen lassen Sie uns nun noch den letzten Feinschliff am Testbild-Programm vornehmen. Zwei Dinge sind hier noch zu tun.

- Bei der Berechnung des Kreises muss die verwendete Stiftgröße so einkalkuliert werden, dass der komplette Kreiszug zu sehen ist.
- Beim Zeichnen des Rasters gilt dasselbe. Die entsprechenden Änderungen finden Sie in den folgenden Codezeilen wieder. Die geänderten Codezeilen befinden sich zum besseren Vergleich auskommentiert im Codelisting.

---

**BEGLEITDATEIEN:** Sie finden das modifizierte Testbild-Projekt im Verzeichnis `.\VB 2005 - Entwicklerbuch\G - Smart-Client\Kap29\VBTestbild03\`

---

### Platzierung des Kreises:

```
'Zielkreuz und Kreis in die Mitte
locPen = New Pen(Color.White, GridLineWidth)

'Alte Version:
'g.DrawEllipse(locPen, ClientSize.Width \ 2 - ClientSize.Height \ 2, _
'               0, _
'               ClientSize.Height, _
'               ClientSize.Height)

'Liniенstärke bei der Umfangberechnung des Kreises mit einbeziehen.
locRectF = New RectangleF((ClientSize.Width / 2.0! - ClientSize.Height / 2.0!) + GridLineWidth / 2,
                         -
                         GridLineWidth / 2.0!, -
                         ClientSize.Height - GridLineWidth / 2, -
                         ClientSize.Height - (GridLineWidth))
g.DrawEllipse(locPen, locRectF)
```

### Berechnung der Rastergröße:

```
Public ReadOnly Property GridSize() As SizeF
    Get
        'So sah es vorher aus:
        'Return New SizeF(CSng(ClientSize.Width / GridCols),
        '                  CSng(ClientSize.Height / GridRows))'

        'Bei der Berechnung der Zellenausmaße die Linienbreite
        'mit in die Verhältnisrechnung einbeziehen.
        Return New SizeF(CSng((ClientSize.Width - GridLineWidth) / GridCols), -
                         CSng((ClientSize.Height - GridLineWidth) / GridRows))
    End Get
End Property
```

## Zeichnen des Gitters:

```
Private Sub paintGrid(ByVal g As Graphics)

    Dim locPen As New Pen(GridColor, GridLineWidth)
    Dim locBrush As New SolidBrush(GridColor)

    If GridStyle = GridStyle.Dots Then
        For x As Single = 0 To ClientSize.Width Step GridSize.Width
            For y As Single = 0 To ClientSize.Height Step GridSize.Height
                g.FillRectangle(locBrush, x, y, GridLineWidth, GridLineWidth)
            Next
        Next

    Else:
        'Alte Version
        'For x As Single = 0 To ClientSize.Width Step GridSize.Width
        '    g.DrawLine(locPen, x, 0, x, ClientSize.Height)
        'Next

        'For y As Single = 0 To ClientSize.Height Step GridSize.Height
        '    g.DrawLine(locPen, 0, y, ClientSize.Width, y)
        'Next

        'Aufgepasst: Dieses Konstrukt funktioniert nicht, wegen Rundungsfehlern!
        'Dim locStart, locEnd As Single
        'locStart = GridLineWidth / 2
        'locEnd = ClientSize.Width

        'For x As Single = locStart To locEnd Step GridSize.Width
        '    Problem: locEnd würde niemals genau gleich x sein, wg. Rundungsfehler,
        '    die letzte Linie damit nie an die richtige Stelle gemalt.
        '    If x = locStart Or x = locEnd Then
        '        g.DrawLine(locPen, x, GridLineWidth / 2, x, ClientSize.Height)
        '    Else
        '        g.DrawLine(locPen, x - GridLineWidth / 2, GridLineWidth / 2, x - GridLineWidth / 2, _
        '                  ClientSize.Height)
        '    End If
        'Next

        'locStart = GridLineWidth / 2
        'locEnd = ClientSize.Height
        'For y As Single = locStart To locEnd Step GridSize.Height
        '    If y = locStart Or y = locEnd Then
        '        g.DrawLine(locPen, GridLineWidth / 2, y, ClientSize.Width, y)
        '    Else
        '        g.DrawLine(locPen, GridLineWidth / 2, y - GridLineWidth / 2, _
        '                  ClientSize.Width, y - GridLineWidth / 2)
        '    End If
        'Next
```

Zu diesem, auskommentierten Teil des Listings vielleicht noch ein paar Anmerkungen, damit Sie diese typischen Fehler beim Arbeiten mit Grafikkoordinaten, die Sie mit Single-Werten bestimmen, in Projekten von vornherein vermeiden können. Hier startete der Programmierer den Versuch, das Zeichnen der jeweils ersten und letzten Linie des Gitters durch Koordinatenvergleich zu entdecken – doch dieser Vorgang ist bei vielen Single-Werten grundsätzlich zum Scheitern verurteilt: Beim Vergleichen der aktuell verarbeiteten Koordinate mit dem bekannten Wert der jeweils letzten Koordinate durch den Gleichheitsoperator wird der Ausdruck

```
x = locEnd
```

bzw.

```
y = locEnd
```

niemals wahr. Zwar entspricht locEnd dem Wert x bzw. y *fast*, durch die Umwandlung vom Dezimal- in das Binärsystem und die sich daraus kumulierenden Rundungsfehler aber eben nur *fast*. Das bedeutet:

---

**HINWEIS:** Vermeiden Sie es grundsätzlich, Programmzustände auf Grund von Fließkommavergleichen festzustellen. Sie können mit Double- bzw. Single-Werten gefahrlos rechnen und die errechneten Ergebnisse zur Bestimmung von Koordinaten verwenden. Durch kumulierte Rundungsfehler schlagen Vergleiche aber in den meisten Fällen fehl, und Ihr Programm arbeitet nicht wie erwartet. Im hier gezeigten Beispiel könnte locEnd beispielsweise den Wert 477,5 innehaben; x ist durch kumulierte Rundungsfehler im entscheidenden Moment aber nicht 477,5, sondern 477,500001 – und das ist zwar *fast* 477,5 aber nicht ausreichend, um im Vergleichsausdruck True zurückzuliefern und die Sonderbehandlung für die als zuletzt gemalt erkannte Linie einzuleiten.

---

```
Dim locStart, locEnd As Single
locStart = GridLineWidth / 2
locEnd = ClientSize.Width - GridSize.Width

'Erste Linie Sonderfall:
g.DrawLine(locPen, GridLineWidth / 2, 0, GridLineWidth / 2, ClientSize.Height)

'Mittlere Linien malen.
For x As Single = locStart To locEnd Step GridSize.Width
    g.DrawLine(locPen, x - GridLineWidth / 2, GridLineWidth / 2, _
               x - GridLineWidth / 2, ClientSize.Height)
Next

'Letzte Linie Sonderfall.
g.DrawLine(locPen, ClientSize.Width - GridLineWidth / 2, 0, _
           ClientSize.Width - GridLineWidth / 2, ClientSize.Height)

'Horizontale Linien:
locStart = GridLineWidth / 2
locEnd = ClientSize.Height - GridSize.Height

'Erste Linie Sonderfall:
g.DrawLine(locPen, 0, GridLineWidth / 2, _
           ClientSize.Width, GridLineWidth / 2)
```

```
For y As Single = locStart To locEnd Step GridSize.Height
    g.DrawLine(locPen, GridLineWidth / 2, y - GridLineWidth / 2, _
               ClientSize.Width, y - GridLineWidth / 2)
Next

'Letzte Linie Sonderfall:
g.DrawLine(locPen, 0, ClientSize.Height - GridLineWidth / 2, _
           ClientSize.Width - GridLineWidth / 2, ClientSize.Height - GridLineWidth / 2)

End If

End Sub
```

Stattdessen machen Sie es besser wie in dieser Version der *Gitter-Mal-Prozedur* gezeigt. Die Behandlung der Sonderfälle ist absolut codiert und ihre Ausführung obliegt keiner bedingten Programmverzweigung.

# 30 Entwickeln von Steuerelementen

- 
- 894 **Neue Steuerelemente auf Basis vorhandener Steuerelemente implementieren**
  - 903 **Entwickeln von konstituierenden Steuerelementen**
  - 916 **Erstellen von Steuerelementen von Grund auf**
- 

Die konsequente Einhaltung der objektorientierten Programmierung prädestiniert Entwickler geradezu zur Entwicklung von wieder verwendbaren Komponenten wie beispielsweise Funktionsbibliotheken oder Benutzersteuerelementen.

Hier und da konnten Sie im Verlauf der letzten Kapitel schon mehrfach sehen, wie einfach in Visual Basic 2005 das Erstellen neuer Benutzersteuerelemente von statthen geht; denken Sie dabei exemplarisch an ► Kapitel 28 und die »Entwicklung« eines Button-Elements, das auch ein Rechts-Klick-Ereignis zur Verfügung stellt.

Sobald Sie Windows Forms-Anwendungen entwickeln, arbeiten Sie automatisch mit visualisierten Komponenten oder einfach ausgedrückt: mit Steuerelementen. Steuerelemente sind in Sachen RAD<sup>1</sup> eine wichtiges Werkzeug, um Anwendungen mit grafischem Frontend tatsächlich schnell fertig stellen zu können. Aus diesem Grund bietet es sich für wiederkehrende Aufgaben an, eigene Steuerelemente zu entwickeln, und wie das funktioniert, mag Ihnen dieses Kapitel zeigen.

Die Anwendung von Steuerelementen selbst beschränkt sich dabei nicht nur auf eine simple Erweiterung von Bedienungselementen einer Programmoberfläche. Sie können ganze Geschäftslogiken in Steuerelementen zusammenfassen und diese in mehreren Ihrer Anwendungen wieder verwenden. In vielen Anwendungen müssen beispielsweise Kontaktdaten bestimmter Personen erfasst und verwaltet werden. Sie könnten also beispielsweise ein Kontakte-Steuerelement entwickeln, das dem Anwender die Verwaltung und das Neuanlegen von Adressen ermöglicht. Dieses Steuerelement entwickeln Sie nur ein einziges Mal, und Sie verwenden es wieder, indem Sie es wie jedes andere Steuerelement in ein Formular einer neuen Anwendung einfügen, in der Sie diese Funktionalität benötigen.

Um neue Steuerelemente zu entwickeln, stehen Ihnen grundsätzlich drei Wege zur Verfügung:

- Sie erweitern ein vorhandenes Steuerelement. Möchten Sie beispielsweise das TextBox-Steuerelement um zusätzliche Funktionalitäten ergänzen, leiten Sie es in eine neue Klasse ab und implementieren lediglich die Erweiterungen.

---

<sup>1</sup> Rapid Application Development, etwa: zügige Anwendungsentwicklung.

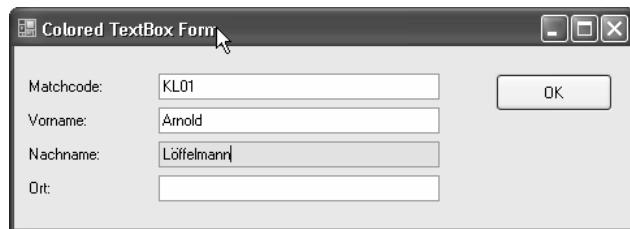
- Sie erstellen ein neues Steuerelement auf Basis mehrerer schon vorhandener Steuerelemente, fassen also die Funktionalität verschiedener vorhandener Steuerelemente zu einem neuen Steuerelement zusammen. Solche Steuerelemente werden auch als *konstituierende Steuerelemente* bezeichnet. Um Steuerelemente dieser Art zu entwerfen, greifen Sie entweder auf Visual Studio selbst zurück und nutzen dessen Designer-Unterstützung, um neue Steuerelemente genau so einfach wie Formulare zu erstellen.<sup>2</sup> Oder Sie leiten Ihre neue Steuerelementklasse von ContainerControl ab und implementieren die Funktionalität ausschließlich über Code.
- Sie implementieren ein Steuerelement komplett neu von Grund auf. In diesem Fall leiten Sie Ihre neue Steuerelementklasse von der Control-Klasse ab. Sie müssen sich dann aber auch um das Zeichnen seiner sichtbaren Komponenten kümmern – feste Trittsicherheit im Umgang mit den Funktionen des GDI+ ist dabei eine wichtige Voraussetzung.

## Neue Steuerelemente auf Basis vorhandener Steuerelemente implementieren

Die einfachste Weise, ein neues Steuerelement zu erstellen, ist, es auf Basis eines bereits vorhandenen Steuerelements zu implementieren. Durch simples Vererben einer vorhandenen Steuerelementklasse schaffen Sie bereits ein neues Steuerelement, das über die gesamte Funktionalität seines Ahnen verfügt. Ihre Aufgabe beschränkt sich anschließend darauf, durch Ergänzen von Funktionen oder Über schreiben von schon vorhandenen Prozeduren die Funktionalität des ursprünglichen Steuerelements zu verändern oder zu ergänzen.

Noch in Visual Basic 2003.NET war es ein vergleichsweise großer Aufwand, ein Steuerelement so zu erstellen, dass Sie es in der Toolbox wieder finden konnten. Sie mussten dazu Ihr Steuerelement in einer eigenen Assembly erstellen, und einige weitere Handgriffe waren nötig, um es letzten Endes in der Toolbox als entsprechendes Symbol anzusehen.

In Visual Studio 2005 ist das so einfach wie das Erstellen einer neuen Klasse geworden. Steuerelemente können zwar nach wie vor in eigenen Assemblies liegen, aber das ist kein Muss mehr, um es in der Toolbox sehen zu können.



**Abbildung 30.1:** Die Einfärbung des jeweils fokussierten Eingabefelds hilft dem Anwender die Übersicht zu bewahren, ist für den Entwickler aber unangenehme Fließbandarbeit

Das folgende Beispiel soll das verdeutlichen. Das Szenario: Sie möchten eine Eingabemöglichkeit für Formulare schaffen, bei der das aktuelle (das »fokussierte«) Eingabefeld, der besseren Übersichtlichkeit halber, gelb unterlegt wird. Damit wird es für den Anwender gerade in großen Formularen einfacher, über eine längere Zeit ermüdfungsfrei zu arbeiten.

---

<sup>2</sup> Steuerelemente dieser Art werden in Visual Studio *Benutzersteuerelemente* genannt.

Normalerweise müssten Sie wie folgt vorgehen, um eine solche Funktionalität in einem Formular zu implementieren:

- Sobald ein TextBox-Steuerelement im Formular den Eingabefokus erhält, müssten Sie seine Hintergrundfarbe in eine auffällige aber dennoch für die Eingabe nicht störende Farbe (beispielsweise gelb) ändern. Das TextBox-Steuerelement stellt dafür das GetFocus-Ereignis zur Verfügung, das Sie durch eine geeignete Ereignisbehandlungsroutine behandeln und dort mit der Backcolor-Eigenschaft des Steuerelements die Hintergrundfarbe ändern.
- Wenn das TextBox-Steuerelement den Fokus wieder verliert, würden Sie seine Hintergrundfarbe wieder mit dem ursprünglichen Farbwert wiederherstellen.
- Diese Implementierung müsste für jedes Eingabefeld erfolgen.

Der Code eines entsprechenden Formulars sähe dann folgendermaßen aus:

```
Public Class Form1

    Private myOriginalBackcolor As Color

    Private Sub btnOK_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnOK.Click
        Me.Close()
    End Sub

    Private Sub txtMatchcode_GotFocus(ByVal sender As Object, ByVal e As System.EventArgs) _
        Handles txtMatchcode.GotFocus
        myOriginalBackcolor = txtMatchcode.BackColor
        txtMatchcode.BackColor = Color.Yellow
    End Sub

    Private Sub txtMatchcode_LostFocus(ByVal sender As Object, ByVal e As System.EventArgs) _
        Handles txtMatchcode.LostFocus
        txtMatchcode.BackColor = myOriginalBackcolor
    End Sub

    Private Sub txtNachname_GotFocus(ByVal sender As Object, ByVal e As System.EventArgs) _
        Handles txtNachname.GotFocus
        myOriginalBackcolor = txtNachname.BackColor
        txtNachname.BackColor = Color.Yellow
    End Sub

    Private Sub txtNachname_LostFocus(ByVal sender As Object, ByVal e As System.EventArgs) _
        Handles txtNachname.LostFocus
        txtNachname.BackColor = myOriginalBackcolor
    End Sub

    Private Sub txtVorname_GotFocus(ByVal sender As Object, ByVal e As System.EventArgs) _
        Handles txtVorname.GotFocus
        myOriginalBackcolor = txtVorname.BackColor
        txtVorname.BackColor = Color.Yellow
    End Sub
```

```

End Sub

Private Sub txtVorname_LostFocus(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles txtVorname.LostFocus
    txtVorname.BackColor = myOriginalBackcolor
End Sub

Private Sub txtOrt_GotFocus(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles txtOrt.GotFocus
    myOriginalBackcolor = txtOrt.BackColor
    txtOrt.BackColor = Color.Yellow
End Sub

Private Sub txtOrt_LostFocus(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles txtOrt.LostFocus
    txtOrt.BackColor = myOriginalBackcolor
End Sub

End Class

```

Und das kann es doch wohl nicht sein, oder? – Der Aufwand rechtfertigt hier eigentlich nicht das Ergebnis.<sup>3</sup>

Ungleich besser wäre es, gäbe es ein TextBox-Steuerelement – nennen wir es FocusColoredTextBox – das eine FocusColor-Eigenschaft sowie eine OnFocusColor-Eigenschaft anbietet. Die FocusColor-Eigenschaft würde dann die Farbe bestimmen, mit der der Hintergrund des Eingabebereichs beim Erhalten des Fokus eingefärbt würde, und die OnFocusColor-Eigenschaft vom Typ Boolean trüfe die Aussage, ob die Einfärbung des Hintergrunds überhaupt stattfände.

## Der Weg eines Steuerelements vom Klassencode in die Toolbox

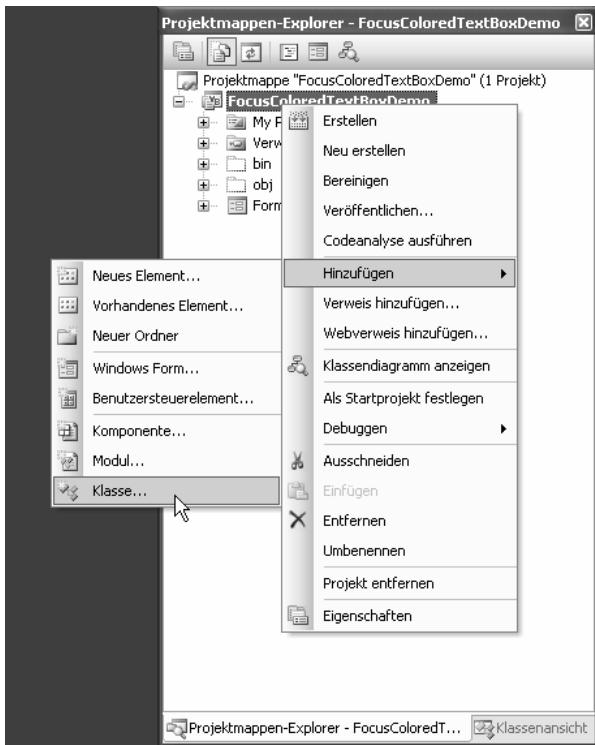
Das Entwickeln und der anschließende Umgang mit selbst definierten oder weiteren Steuerelementen ist nie so einfach gewesen, wie in Visual Basic 2005. Sobald Sie Ihrem Projekt eine Klassendatei hinzufügen, in der sich die Definition einer von Control (oder einer auf Control basierenden) abgeleiteten Komponente befindet, dann sehen Sie die neue Komponente direkt nach dem ersten Kompilieren in der Toolbox.

Probieren Sie es aus:

- Erstellen Sie eine neue Windows Forms-Anwendung.
- Fügen Sie dem Projekt eine neue Klassendatei hinzu, indem Sie aus dem Kontextmenü des Projektmappen-Explorers den Menüpunkt *Hinzufügen* und *Klasse* wählen.

---

<sup>3</sup> Aufmerksamen Lesern ist es sicherlich nicht entgangen, dass man diesen Code schon ein wenig kompakter formulieren könnte, würde man das Ein- und Ausfärben in nur zwei Ereignisbehandlungsroutinen platzieren, die an alle TextBox-Steuerelemente gebunden wären – Handels erlaubt ja schließlich die Verdrahtung mit mehreren Ereignisquellen. Aber auch dieses Verfahren würde nicht der OOP und dem Kapseln von Funktionalitäten in geschlossenen Komponenten entsprechen.



**Abbildung 30.2:** Um ein neues Steuerelement zu erstellen, fügen Sie Ihrem Projekt zunächst eine ganz normale Klassendatei hinzu ...

- Geben Sie im Dialog, der jetzt erscheint, einen neuen Klassencode-Dateinamen für die Codedatei Ihres zukünftigen Steuerelements an – beispielsweise *FocusedColoredTextBox*.
- Doppelklicken Sie auf die neu erstellte Klassencode-Datei im Projektmappen-Explorer, um sie im Codeeditor zu öffnen. Ändern Sie den Klassencode zum Beispiel folgendermaßen ab:

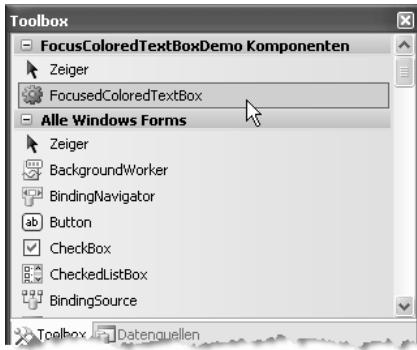
```

FocusedColoredTextBox.vb [Form1.vb [Entwurf]] Startseite
(Allgemein)
Public Class FocusedColoredTextBox
    Inherits TextBox
End Class

```

**Abbildung 30.3:** ... ergänzen die Klassendefinition mit *Inherits*, um die Ableitung von einem vorhandenen Steuerelement (mindestens von *Control*), ...

- Doppelklicken Sie auf die Formulardatei *Form1.vb* im Projektmappen-Explorer, um sie im Designer darzustellen.
- Falls die Toolbox nicht permanent angezeigt wird, holen Sie diese in den Vordergrund, und klicken Sie auf das kleine Reißzweckensymbol an der rechten oberen Ecke des Fensters, um sie permanent darzustellen.
- Wählen Sie aus dem Menü *Erstellen* den Menüpunkt *Projektmappe neu erstellen* (bzw. *[Projektname] neu erstellen*), und beobachten Sie, was in der Toolbox passiert.



**Abbildung 30.4:** ... und sofort nach dem ersten Erstellen Ihres Projektes befindet sich das neue Steuerelement auch schon in der Toolbox zur sofortigen Verwendung in Ihren Formularen

Wie in Abbildung 30.4 zu sehen, befindet sich das neue Steuerelement zur sofortigen Verwendung in der Toolbox.

Sie haben also jetzt ein neues Steuerelement namens FocusColoredTextBox. Da der Klassencode außer der Klassenableitung noch keine weitere Funktionalität aufweist, kann dieses neue Steuerelement exakt das, was ein TextBox-Steuerelement auch kann – aber ich finde, selbst das ist schon beeindruckend, oder nicht?

Probieren Sie es aus: Ziehen Sie das neue Steuerelement im Formular auf, und schauen Sie sich im Eigenschaftenfenster an, welche Eigenschaften dieses Steuerelement zu bieten hat.

---

**BEGLEITDATEIEN:** Zum Nachvollziehen der nächsten Abschnitte verwenden Sie am besten das bereits fertige Beispielprojekt, das Sie im Verzeichnis .\VB 2005 - Entwicklerbuch\G - SmartClient\Kap30\FocusColoredTextBoxDemo01\ finden.

---

## Implementierung von Funktionslogik und Eigenschaften

Die Implementierung der Funktionslogik ist keine aufwändige Angelegenheit, wie Sie anhand des folgenden Listings sehen können, bei dem die XML-Dokumentierungszeilen alleine fast zur Hälfte des Gesamtcodes beitragen.

```
Public Class FocusedColoredTextBox
    Inherits TextBox

    Private myFocusColor As Color      ' Fokus-Einfärbfarbe (intern)
    Private myOriginalBackcolor As Color ' Zwischenspeicher der ursprünglichen Farbe
    Private myOnFocusColor As Boolean   ' Flag, dass Autofärben bei Fokussierung bestimmt (intern)

    ''' <summary>
    ''' Erstellt eine neue Instanz dieser Klasse
    ''' </summary>
    ''' <remarks></remarks>
    Sub New()
        'NIE VERGESSEN, gerade bei Steuerelementen!
        'Den Basiskonstruktor aufrufen!
        MyBase.New()
    End Sub
```

```

'Voreingestellter Wert ist gelb.
myFocusColor = Color.Yellow

'Voreingestellt ist: Bei Fokus wird eingefärbt.
myOnFocusColor = True
End Sub

'Überschrieben: Färbt im Bedarfsfall mit FocusColor ein,
'wenn das Steuerelement den Fokus erhält.
Protected Overrides Sub OnGotFocus(ByVal e As System.EventArgs)
    MyBase.OnGotFocus(e)
    If OnFocusColor Then
        myOriginalBackcolor = Me.BackColor
        Me.BackColor = FocusColor
    End If
End Sub

'Überschrieben: Stellt die Ursprungshintergrundfarbe im
'Bedarfsfall wieder her, wenn das Steuerelement den Fokus verliert.
Protected Overrides Sub OnLostFocus(ByVal e As System.EventArgs)
    MyBase.OnLostFocus(e)
    If OnFocusColor Then
        Me.BackColor = myOriginalBackcolor
    End If
End Sub

''' <summary>
''' Bestimmt oder ermittelt die Farbe, die das Steuerelement automatisch erhält,
''' wenn es fokussiert wird, und wenn die OnFocusColor-Eigenschaft gesetzt wurde.
''' </summary>
''' <value>Die Farbe, die beim Fokussieren verwendet werden soll.</value>
''' <returns></returns>
''' <remarks></remarks>
Public Property FocusColor() As Color
    Get
        Return myFocusColor
    End Get
    Set(ByVal value As Color)
        myFocusColor = value
    End Set
End Property

''' <summary>
''' Bestimmt oder ermittelt, ob das Steuerelement beim Erhalten des Fokus
''' automatisch mit der in FocusColor eingestellten Farbe eingefärbt werden soll.
''' </summary>
''' <value>True, wenn die Autoeinfärbung aktiviert werden soll.</value>
''' <returns></returns>
''' <remarks></remarks>
Public Property OnFocusColor() As Boolean
    Get
        Return myOnFocusColor
    End Get

```

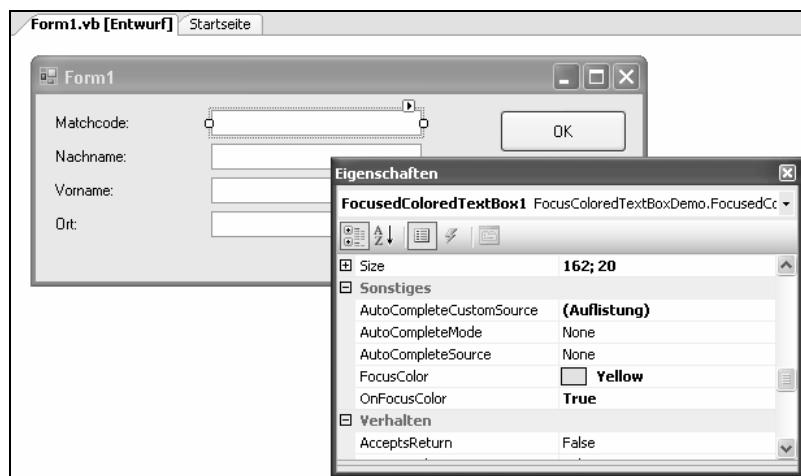
```

Set(ByVal value As Boolean)
    myOnFocusColor = value
End Set
End Property
End Class

```

Wenn Sie das Beispiel kompilieren und starten, können Sie die neuen Steuerelemente, auf denen das Formular des Beispiele basiert, direkt ausprobieren.

Sobald Sie das Programm beenden und eine der vorhandenen Instanzen des neuen Steuerelements selektieren, können Sie die beiden neuen Eigenschaften im Eigenschaftenfenster entdecken:



**Abbildung 30.5:** Die neuen Eigenschaften der *FocusedColoredTextBox* sind im Eigenschaftenfenster zu finden – allerdings unter der Kategorie *Sonstiges*, und die Werte der Eigenschaften sind auf eine Weise markiert, die implizieren, dass sie verändert wurden

Sie sehen dabei allerdings ebenfalls, dass sie in einer Kategorie platziert wurden (»Sonstiges«), in der sie eigentlich gar nicht hingehören. Darüber hinaus werden die Werte der Eigenschaften auch so dargestellt, als wären sie abweichend von den Standardeigenschaften eingestellt worden – denn nur in diesem Fall werden die Werte von Eigenschaften im Fenster fett hervorgehoben.

Der nächste Abschnitt zeigt, wie Sie diese Verhaltensweisen normalisieren.

## Steuern von Eigenschaften im Eigenschaftenfenster

Zur Implementierung einer neuen Eigenschaft, die im Eigenschaftenfenster zur Entwurfszeit angezeigt werden soll, genügt es, wenn Sie eine Eigenschaft im Sinne von Klassen implementieren, also mit entsprechenden Property-Prozeduren.

Das .NET-Framework kennt allerdings gerade für die Steuerung von Eigenschaften im Eigenschaftenfenster eine Reihe von Attributen, die Sie in der folgenden Tabelle beschrieben finden.

Attribute	Beschreibung
Browsable	Bestimmt, ob die Eigenschaft, der dieses Attribut zugeordnet ist, zur Entwurfszeit im Eigenschaftenfenster sichtbar ist oder nicht.
Description	Definiert den Beschreibungstext, der unterhalb des Eigenschaftenfensters zur Entwurfszeit angezeigt wird, wenn die Eigenschaft ausgewählt ist.
DefaultValue	Bestimmt, welchen Typ und Wert der Standardwert der Eigenschaft aufweist. Wenn der Standardwert der Eigenschaft zur Entwurfszeit definiert ist, wird kein Serialisierungscode vom Designer erzeugt. Aus diesem Grund muss die Steuerelementklasse dafür Sorge tragen, dass während der Initialisierung des Steuerelements (am besten schon im Konstruktor) der entsprechende Wert gesetzt ist. Im Eigenschaftenfenster wird eine Eigenschaft, die ihren Standardwert besitzt, in Normalschrift angezeigt; veränderte Werte werden in Fett-schrift angezeigt. <b>Wichtig:</b> Falls es – durch komplexere Datentypen oder Auswertungslogiken – nicht möglich ist, einen Standardwert mit diesem Attribut zu bestimmen, implementieren Sie eine Funktion mit dem Namen der Eigenschaft und dem Präfix »ShouldSerialize...«, die True zurückgibt, wenn die Eigenschaft im aktuellen Zustand nicht den Standardwert zurückliefert. Lautet eine Eigenschaft also Beispielsweise SampleProperty, so sollte die entsprechende Funktion, die den Code Serialisierer für diese Eigenschaft steuert, ShouldSerializeSampleProperty lauten und True zurückliefern, wenn die Wertdefinition für diese Eigenschaft <i>nicht</i> dem Standardwert entspricht. Um in diesem Fall die <i>Zurücksetzen</i> -Funktionalität des Eigenschaftenfensters zur Verfügung zu stellen, implementieren Sie eine weitere Funktion mit dem Präfix »Reset...« (also beispielsweise ResetSampleProperty); diese Funktion muss den Standardwert der Eigenschaft wieder herstellen.
Category	Legt fest, unter welcher Kategorie im Eigenschaftenfenster die Eigenschaft eingeordnet werden soll. Wenn Sie dieses Attribut nicht definieren, wird die Eigenschaft automatisch unter einer neuen Kategorie namens <i>Sonstiges</i> eingeordnet.

**Tabelle 9.1:** Diese Attribute sind wichtig für die Steuerung der Anzeige im Eigenschaftenfenster

**BEGLEITDATEIEN:** Die Version des Beispiels, die den Einsatz dieser Attribute demonstriert, finden Sie Verzeichnis .\VB2005 - Entwicklerbuch\G - SmartClient\Kap30\FocusColoredTextBoxDemo02\.

Der folgende Codeausschnitt zeigt beispielhaft, wie diese Attribute für Eigenschaften zur Steuerung ihres Verhaltens im Eigenschaftenfenster zum Einsatz kommen:

```

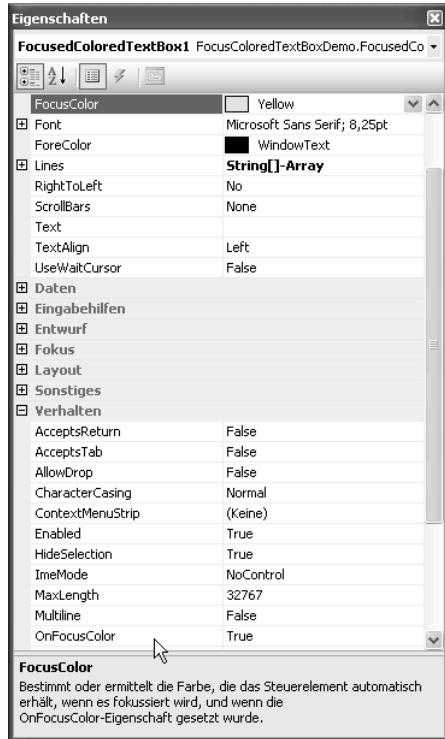
''' <summary>
''' Bestimmt oder ermittelt die Farbe, die das Steuerelement automatisch erhält,
''' wenn es fokussiert wird, und wenn die OnFocusColor-Eigenschaft gesetzt wurde.
''' </summary>
''' <value>Die Farbe, die beim Fokussieren verwendet werden soll.</value>
''' <returns></returns>
''' <remarks></remarks>
<Category("Darstellung"),
 DefaultValue(GetType(Color), "Yellow"),
 Description("Bestimmt oder ermittelt die Farbe, die das Steuerelement automatisch erhält, " &
 "wenn es fokussiert wird, und wenn die OnFocusColor-Eigenschaft gesetzt wurde."), _
 Browsable(True)>
Public Property FocusColor() As Color
    Get
        Return myFocusColor
    End Get

```

```

Set(ByVal value As Color)
    myFocusColor = value
End Set
End Property
''' <summary>
''' Bestimmt oder ermittelt, ob das Steuerelement bei Erhalt des Fokus
''' automatisch mit der in FocusColor eingestellten Farbe eingefrbt werden soll.
''' </summary>
''' <value>True, wenn die Autoeinfrbung aktiviert werden soll.</value>
''' <returns></returns>
''' <remarks></remarks>
<Category("Verhalten"), _
DefaultValue(True), _
Description("Bestimmt oder ermittelt, ob das Steuerelement beim Erhalten des Fokus " &
            "automatisch mit der in FocusColor eingestellten Farbe eingefrbt werden soll."), _
Browsable(True)>
Public Property OnFocusColor() As Boolean
    Get
        Return myOnFocusColor
    End Get
    Set(ByVal value As Boolean)
        myOnFocusColor = value
    End Set
End Property

```



**Abbildung 30.6:** Die neuen Eigenschaften gliedern sich durch den Einsatz entsprechender Attribute so sauber in die vorhandenen ein, dass ein Unterschied nicht mehr erkennbar ist. Sowohl die Anzeige der Standardwerte funktioniert bei beiden neuen Eigenschaften (siehe *FocusColor*, ganz oben, sowie *OnFocusColor*, ganz unten) als auch das Einblenden der helfenden Erklärung der Eigenschaften, die im unteren Teil des Eigenschaftenfensters zu sehen ist. Und auch die Platzierung der Eigenschaften in den richtigen Kategorien funktioniert überdies.

---

**HINWEIS:** Die DefaultValueAttribut-Klasse verfügt über nicht weniger als 11 Überladungen. Folgende Regel sollte deswegen bei ihrer Anwendung gelten: Für primitive Datentypen geben Sie den Standardwert einfach als einziges Argument in Klammern an (siehe OnFocusColor im vorherigen Listingauszug). Komplexeren Strukturen wie beispielsweise Color, Size, Location und andere Typen benötigen so genannte Eigenschaften-Deskriptoren-Klassen, die in der Lage sind, eine Instanz des Wertetyps aus einer Zeichenkette zu bilden. Alle wichtigen im .NET-Framework vorhandenen Wertetypen verfügen über solcherlei Eigenschaften-Deskriptoren. In diesem Fall reicht es allerdings nicht aus, einen einzelnen Standardwert der DefaultValueAttribut-Klasse zu übergeben; stattdessen geben Sie zwei Parameter an: Den Typ der Struktur als auch seine Wertrepräsentation als Zeichenkette, so, wie es im vorherigen Listingauszug bei der FocusColor-Eigenschaft gezeigt wird.

---

**TIPP:** Gerade bei Steuerelementen, die Sie immer wieder verwenden wollen, ergibt es Sinn, eine eigene Assembly als Komponentenbibliothek zu erstellen. ► Abschnitt »Anlegen eines Projekts für die Erstellung einer eigenen Steuerelement-Assembly« ab Seite 904 zeigt, wie das geht, wenn auch mit anderen Steuerelementtypen. Die gezeigten Schritte für das Erstellen einer Assembly können Sie aber genauso gut auf wie in diesem Kapitel besprochenen Steuerelementtypen anwenden, die von anderen Steuerelementen abgeleitet sind.

---

## Entwickeln von konstituierenden Steuerelementen

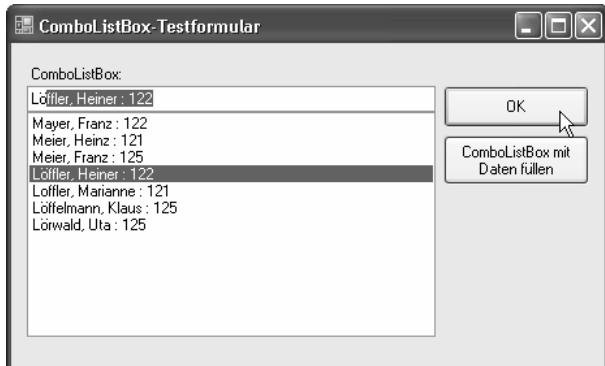
Bei konstituierenden Steuerelementen handelt es sich, wie eingangs bereits erwähnt, um solche, die aus mehreren vorhandenen ein neues mit erweiterter Funktionalität bilden. Wenn Sie mit Visual Studio konstituierende Steuerelemente entwickeln, haben Sie generell zwei Möglichkeiten:

- Sie gehen so vor, wie Sie es im vorangegangenen Abschnitt kennen gelernt haben, leiten Ihr Steuerelement allerdings von ContainerControl<sup>4</sup> ab. Um das Steuerelement auf mehreren vorhandenen Steuerelementklassen basieren zu lassen, fügen Sie der Controls-Auflistung im Konstruktor neue Instanzen der vorhandenen Steuerelementklassen hinzu. Die Positionierung der Controls definieren Sie durch das Setzen der Location-, Size-, Dock- bzw. Anchor-Eigenschaften im Code.
- Sie nehmen den Visual Studio-Designer zu Hilfe und definieren das neue Steuerelement wie ein Formular. Der Vorteil: Sie sparen wertvolle Zeit (nämlich die des Codeschreibens zum Instanziieren und Positionieren der Basisstreuerelemente) und können sich auf die eigentliche Funktionalitätsimplementierung des Steuerelements konzentrieren.

Im folgenden Beispiel verwenden wir die letzte der beiden angesprochenen Vorgehensweisen, um ein Steuerelement zu erstellen. In diesem Beispiel handelt es sich um die Kombination eines TextBox- und eines ListBox-Steuerelements. Die Liste kann wie eine herkömmliche ListBox mit entsprechenden Einträgen ausgestattet werden. Wenn der Anwender in der TextBox einen Text eingibt, versucht das Steuerelement auf Grund der vorhandenen Listeneinträge den eingegebenen Text zu komplettieren (siehe Abbildung 30.7).

---

<sup>4</sup> Theoretisch könnten Sie ein konstituierendes Steuerelement auch von Control ableiten, da es ebenfalls über die erforderliche ControlCollection (über seine Controls-Eigenschaft) verfügt. Allerdings hat ContainerControl einige weitere Vorzüge – so kann es beispielsweise für eingebundene Steuerelemente, die nicht in den Anzeigebereich passen, automatisch eine Rollbalkenfunktionalität zur Verfügung stellen, mit der der Anwender den sichtbaren Ausschnitt darstellen lassen kann (die AutoScroll-Eigenschaft ist dafür zuständig).



**Abbildung 30.7:** Mit der *ComboBox* vereinen Sie die Funktionalität von *TextBox* und *ListBox*

Der Aufwand für das Erstellen eines konstituierenden Steuerelements ist ein wenig intensiver, weil .NET die so genannte Mehrfachvererbung nicht zulässt: Eine Klasse kann nicht aus zwei Klassen erben. Aus diesem Grund werden die bestehenden Steuerelemente als Klassen-Member angelegt, und das Offenlegen entsprechender Eigenschaften, Ereignisse und Methoden kann nur durch »Delegierung« (zu den eingebundenen Klassen) erfolgen. Das heißt im Klartext: Im Gegensatz zur echten Vererbung müssen Elemente komplett neu implementiert werden, die sich bei Anwendung der richtigen Vererbung in der neuen Klasse ergeben würden.

---

**HINWEIS:** Im Gegensatz zum vorherigen Beispiel, bei dem das Steuerelement Bestandteil einzelnen Projektes war, zeigt dieses Beispiel einen etwas aufwändigeren aber dafür sinnvolleren Weg bei der Implementierung des Steuerelements. Das Steuerelement wird nämlich hier in einer eigenen Assembly abgelegt, und wenn Sie die neue Komponente erst einmal entwickelt und ausgiebig getestet haben, können Sie sie durch einfaches Zurückgreifen auf das erstellte Projekt es ohne Probleme immer wieder verwenden.

---

Natürlich finden Sie auch das folgende Beispiel in den Begleitdateien zum Buch. Dennoch ergeht an dieser Stelle wieder der Hinweis, dass es durchaus Sinn ergibt, die Erstellung des folgenden Projektes »manuell« nachzuvollziehen, um für spätere, eigene Entwicklungen vorbereitet zu sein.

---

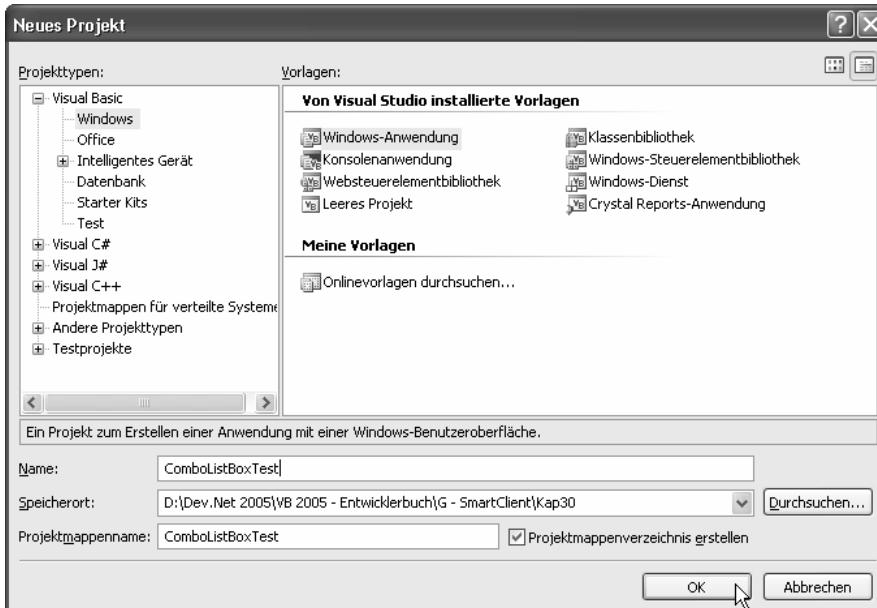
**BEGLEITDATEIEN:** Sie finden die Projektmappe zu diesem Beispiel im Verzeichnis `.\VB 2005 - Entwicklerbuch\G - SmartClient\Kap30\ComboBoxTest\ComboBoxTest.sln`

---

## Anlegen eines Projekts für die Erstellung einer eigenen Steuerelement-Assembly

Die folgende Schritt-für-Schritt-Anleitung zeigt, wie Sie vorgehen müssen, um ein Steuerelement in einem Projekt unterzubringen, aus dem nach dem Austesten und Erstellen eine eigene Assembly hervorgeht. Wenn Sie entscheiden, dass das Steuerelement einen finalen Versionsstand erreicht hat, und keine Modifikationen mehr notwendig sind, dann können Sie diese Assembly zukünftig durch das Einrichten eines simplen Verweises in Sekundenschnelle anderen Projekten hinzufügen; die Quellcode-Dateien des Steuerelements benötigen Sie dann in zukünftigen Projekten nicht einmal mehr.

- Um eine neue Projektmappe zu erstellen, die sowohl das Projekt für die Steuerelement-Assembly als auch ein notwendiges Testprojekt enthält, legen Sie zunächst erst einmal eine simple Windows-Anwendung an. Wählen Sie dazu aus dem Menü *Datei* die Menüpunkte *Neu/Projekt*.
- Wählen Sie *Visual Basic/Windows* im Bereich *Projekttypen* und als Vorlage *Windows-Anwendung*.
- Bestimmen Sie das Projektmappen-Verzeichnis und geben Sie für das Windows-Anwendungsprojekt einen aussagekräftigen Namen ein – etwa *ComboBoxTest*.
- Klicken Sie auf *Projektmappen-Verzeichnis erstellen*, damit eine weitere Verzeichnisebene eingerichtet wird, in der sich später *beide* Projekte in separaten Verzeichnissen befinden.
- Klicken Sie auf *OK*, um sowohl Projektmappe als auch Projekt zu erstellen.



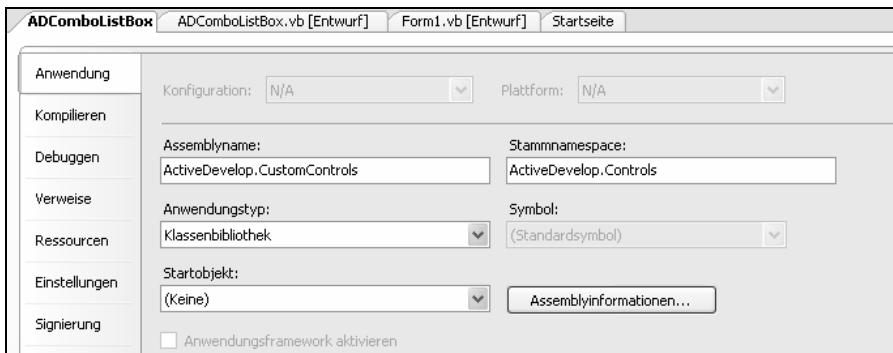
**Abbildung 30.8:** Legen Sie mit diesem Dialog zunächst die Projektmappe und das Projekt zum Testen des Steuerelements an

- Klicken Sie im Projektmappen-Explorer mit der rechten Maustaste auf die Projektmappe *ComboBoxTest* (nicht auf das Projekt!), und wählen Sie den Menübefehl *Hinzufügen/Neues Projekt*.
- Wählen Sie *Visual Basic/Windows* im Bereich *Projekttypen* und als Vorlage *Windows-Steuerelementbibliothek*.
- Geben Sie für das Windows-Anwendungsprojekt einen aussagekräftigen Namen ein – etwa *AD-ComboBox*.
- Klicken Sie auf *OK*, um das Projekt neu zu erstellen und der Projektmappe hinzuzufügen.
- Klicken Sie im Projektmappen-Explorer mit der rechten Maustaste über der Klassencodedatei *UserControl1.vb* des Projekts *ADComboBox*, und benennen Sie sie in *ADComboBox.vb* um.

## Einrichten von Namespace, Assembly-Namen und Projektverweisen

Im folgenden Schritt bestimmen Sie den Namespace sowie den Namen der Assembly-DLL, unter der das Steuerelement abrufbar wird.

- Klicken Sie dazu im Projektmappen-Explorer mit der rechten Maustaste auf den Projektnamen *ADComboBox*, und wählen Sie im sich jetzt öffnenden Kontextmenü den Menüpunkt *Eigenschaften*.



**Abbildung 30.9:** Der *Assemblyname* ist auch der Name der später aus dem Projektkompilat hervorgehenden DLL; über den Stammnamespace-Namen greifen Sie auf die Klasse(n) selbst zu

- Unter *Assemblyname* bestimmen Sie den Namen der Assembly, die später die Klasse Ihres Steuerelements beinhalten soll. Diesen Namen wird auch die DLL-Datei tragen, die die Assembly mit Ihrem Steuerelement enthält. Für dieses Beispiel habe ich eine Kombination aus Firmennamen und »CustomControls« verwendet – denn diese Assembly soll eventuell später einmal dazu gedacht sein, weitere Benutzersteuerelemente zu beinhalten. Der Name der DLL lautet also nach dem Kompilieren *ActiveDevelop.CustomControls.dll*.
- Unter *Stammnamespace* bestimmen Sie, unter welchem Namespace die Steuerelementklasse (oder später einmal die Klassen, falls mehrere Klassen dem Projekt hinzugefügt werden sollen) abrufbar ist (bzw. sind). Für dieses Beispiel habe ich eine Kombination aus Firmennamen und *Controls* gewählt. Dazu ein wenig mehr Hintergrundinformationen: Genau so, wie Sie beispielsweise die Zeile

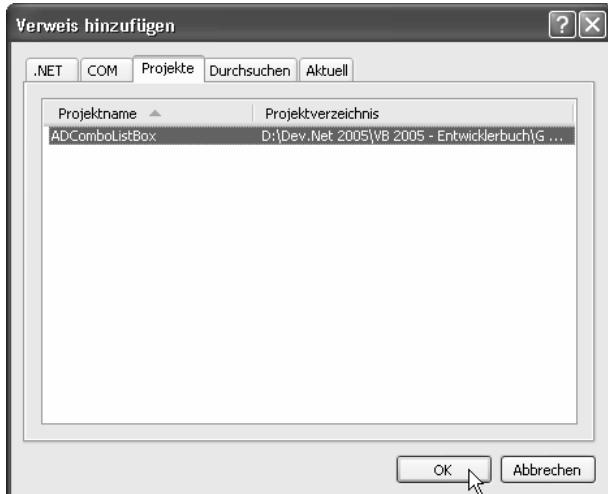
```
Imports System.IO
```

vor eine Klassencodedatei einfügen, um auf die I/O-Funktionen des Frameworks zugreifen zu können, benötigen Sie später, im *ComboBoxTest*-Programm, die folgende Anweisung

```
Imports ActiveDevelop.Controls
```

um auf die Klasse zugreifen zu können, die das *ADComboBox*-Steuerelement darstellt.
- Die *Imports*-Anweisung alleine reicht allerdings nicht. Damit die eigentlichen Funktionsnamen, die Sie erst über die Einbindung des Namespaces *ActiveDevelop.Controls* für den Compiler erkennbar machen, auch später tatsächlich im IML-Code gefunden und aufgelöst werden können, müssen Sie zusätzlich einen Verweis auf die eigentliche Assembly (das Projekt) einrichten. Dazu öffnen Sie das Kontextmenü des Projektes *ComboBoxTest* im Projektmappen-Explorer und wählen dort den Eintrag *Verweis hinzufügen*.

- Wechseln Sie im Dialog, der jetzt erscheint (siehe Abbildung 30.10), auf die Registerkarte *Projekte*, doppelklicken Sie auf den Eintrag *ADComboBox*, und klicken Sie anschließend auf *OK*.

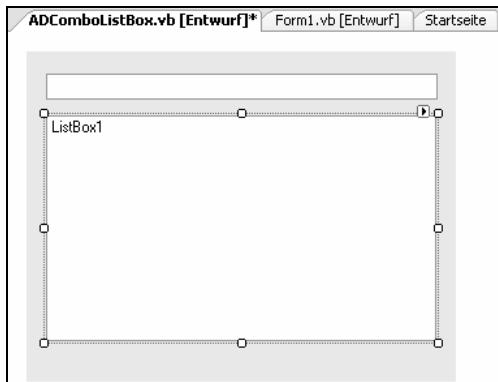


**Abbildung 30.10:** Wenn Sie mit Klassen anderer Projekte Ihrer Projektmappe arbeiten wollen, müssen Sie ebenfalls Verweise auf die benötigten Projekte hinzufügen

### Anordnen von Steuerelementen im benutzerdefinierten Steuerelement

Das Steuerelement, das wir in diesem Beispiel entwickeln, besteht aus zwei im .NET-Framework schon vorhandenen Steuerelementen: Einer `TextBox` und einer `ListBox`. Sie können diese beiden Steuerelemente im Benutzersteuerelement genau wie in einem Formular anordnen und mit den entsprechenden Eigenschaften ausstatten.

- Öffnen Sie dazu den Designer zur Klassendatei *ADComboBox.vb* aus dem Projekt *ADComboBox* per Doppelklick auf den entsprechenden Eintrag im Projektmappen-Explorer.
- Ziehen Sie jetzt jeweils ein `TextBox` - und ein `ListBox` -Steuerelement im Steuerelementcontainer auf, etwa wie in Abbildung 30.11 zu sehen (bitte in dieser Reihenfolge, da es sonst Probleme bei der späteren Einstellung der *Dock*-Eigenschaft geben kann).



**Abbildung 30.11:** In einem Benutzersteuerelement können Sie Steuerelemente genau wie in einem Formular einfügen

- Setzen Sie die Dock-Eigenschaft der TextBox auf Top und die Dock-Eigenschaft der ListBox auf Fill. Die Dock-Eigenschaft bewirkt, dass sich die Ränder eines Steuerelements an die entsprechenden Innenseiten ihres Containers fügen. Wird der Container vergrößert oder verkleinert, passen sich die gedockten Steuerelemente innerhalb des Containers an. So gesehen funktioniert die Dock-Eigenschaft fast wie die Anchor-Eigenschaft, mit dem Unterschied, dass gedockte Steuerelemente immer am äußersten Rand des sie umgebenden Containers verankert werden.

---

**TIPP:** Die so genannte Z-Reihenfolge der Steuerelemente ist wichtig, wenn Sie mit umfangreichen, gedockten Steuerelementen arbeiten. Die Z-Reihenfolge bestimmt, welches Steuerelement über oder unter einem anderen liegt. Durch das Öffnen des Kontextmenüs eines Steuerelements im Designer können Sie dessen Position in der Z-Reihenfolge mit den Funktionen *In den Hintergrund* oder *In den Vordergrund* verändern. Auf diese Weise ändern Sie auch deren Docking-Verhalten untereinander.<sup>5</sup>

---

- Benennen Sie die Steuerelemente um, damit sie den bisherigen Standards der Namensvergebung von Klassen-Membern entsprechen: Nennen Sie das ListBox-Steuerelement myListBox und das TextBox-Steuerelement myTextBox.

---

**HINWEIS:** Sobald Sie Änderungen an den Eigenschaften eines Steuerelements im Designer vorgenommen haben und das Steuerelementprojekt bereits erstellt wurde, zeigt Visual Studio eine Warnmeldung in der Aufgabenliste und weist Sie mit dieser darauf hin, dass Sie das Projekt neu erstellen müssen, damit sich die Änderungen auf alle Instanzen auswirken, die das Steuerelement einbinden.

---

Die Grundeinstellungen für das Steuerelementprojekt sind damit abgeschlossen. Sie können jetzt das Projekt das erste Mal erstellen lassen und so die Assembly für das Steuerelement generieren.

---

**WICHTIG:** Wenn Sie konstituierende Benutzersteuerelemente in der Visual Studio-Umgebung erstellen, wird ein entsprechendes Toolbox-Symbol automatisch eingefügt. Sie können auf das neue Steuerelement ohne weitere Maßnahmen direkt zugreifen. Das Steuerelementprojekt muss dazu allerdings mindestens einmal erstellt worden sein.

---

- Das neue Steuerelement ist nun zum Einsatz bereit, und Sie können es im vorerst letzten Schritt testweise im Formular des *ADComboBoxTest*-Projektes platzieren.

---

**BEGLEITDATEIEN:** Um die Implementierung beschreibungstechnisch abzukürzen, möchte ich für die nächsten Erklärungen auf das bereits fertige Projekt verweisen, das Sie, wie schon eingangs erwähnt, im Verzeichnis *|VB 2005 - Entwicklerbuch|G - SmartClient|Kap30\ComboBoxTest\ComboBoxTest.sln* finden.

---

## Initialisieren des Steuerelements

Im Gegensatz zum vorherigen Beispiel, bei dem das neue Steuerelement aus einem vorhandenen erbte, müssen Sie bei diesem Beispielprojekt viele Eigenschaften erneut implementieren. Das gilt nicht nur für die Eigenschaften, sondern auch für die wichtigsten Ereignisse. Aus diesem Grund ist

---

<sup>5</sup> Das kann in einigen Fällen in eine ziemliche Fummel ausarten. Sie sollten sich daher überlegen, ob Sie mehrere Steuerelemente in solchen Fällen nicht in weiteren Containern zusammenfassen können und nur die Container docken. In vielen Fällen könnte auch die Verwendung der Anchor-Eigenschaft die bessere Alternative darstellen.

der Deklarationsteil sowie der Konstruktor dieses Beispiels ein wenig umfangreicher, wie der folgende Codeausschnitt zeigt:

```
Imports System.ComponentModel

Public Class ADComboBox
    Inherits System.Windows.Forms.UserControl

    Private mySenderIsThis As Boolean
    Private myAutoComplete As Boolean
    Private myAutoSelect As Boolean

    Public Event SelectedIndexChanged(ByVal sender As Object, ByVal e As EventArgs)
    Public Event SelectedValueChanged(ByVal sender As Object, ByVal e As EventArgs)

    Public Sub New()
        MyBase.New()

        ' Dieser Aufruf ist für den Windows Form-Designer erforderlich.
        InitializeComponent()

        ' Eigenschaften vorselektieren.
        myAutoComplete = True
        myAutoSelect = True

        ' Die Listbox soll nicht mit der Tabulatortaste fokussiert werden dürfen!
        myListBox.TabStop = False
    End Sub
```

Die beiden Steuerelemente, mit denen unser Benutzersteuerelement arbeitet, sind nichts anderes als Instanzen einer bestimmten Klasse, die vom Konstruktor der Steuerelementklasse erzeugt werden (während nicht direkt, sondern durch `InitializeComponent`, aber immer noch im Rahmen des Konstruktorkodes). Wollten Sie ein konstituierendes Steuerelement ohne Designer-Unterstützung erstellen, würden Sie im Prinzip genau so vorgehen, wie es der Designer in diesem Beispiel für Sie gemacht hat – den entsprechenden Code finden Sie in der Codedatei `ADComboBox.Designer.vb`, wenn Sie für das Projekt `ADComboBox` im Projektmappen-Explorer alle Dateien anzeigen lassen, und er ist im Folgenden zu sehen:

```
<Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()> _
Partial Class ADComboBox
    Inherits System.Windows.Forms.UserControl

    'UserControl1 überschreibt den Löschvorgang, um die Komponentenliste zu bereinigen.
    <System.Diagnostics.DebuggerNonUserCode()>
    Protected Overrides Sub Dispose(ByVal disposing As Boolean)
        If disposing AndAlso components IsNot Nothing Then
            components.Dispose()
        End If
        MyBase.Dispose(disposing)
    End Sub
```

```

'Wird vom Windows Form-Designer benötigt.
Private components As System.ComponentModel.IContainer

'Hinweis: Die folgende Prozedur ist für den Windows Form-Designer erforderlich.
'Das Bearbeiten ist mit dem Windows Form-Designer möglich.
'Das Bearbeiten mit dem Codeeditor ist nicht möglich.
<System.Diagnostics.DebuggerStepThrough()> _
Private Sub InitializeComponent()
    Me.myTextBox = New System.Windows.Forms.TextBox
    Me myListBox = New System.Windows.Forms.ListBox
    Me.SuspendLayout()
    '
    'myTextBox
    '
    Me.myTextBox.Dock = System.Windows.Forms.DockStyle.Top
    Me.myTextBox.Location = New System.Drawing.Point(0, 0)
    Me.myTextBox.Name = "myTextBox"
    Me.myTextBox.Size = New System.Drawing.Size(185, 20)
    Me.myTextBox.TabIndex = 0
    '
    'myListBox
    '
    Me myListBox.Dock = System.Windows.Forms.DockStyle.Fill
    Me myListBox.FormattingEnabled = True
    Me myListBox.Location = New System.Drawing.Point(0, 20)
    Me myListBox.Name = "myListBox"
    Me myListBox.Size = New System.Drawing.Size(185, 95)
    Me myListBox.TabIndex = 1
    '
    'ADComboBox
    '
    Me.AutoScaleDimensions = New System.Drawing.SizeF(6.0!, 13.0!)
    Me.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font
    Me.Controls.Add(Me myListBox)
    Me.Controls.Add(Me myTextBox)
    Me.Name = "ADComboBox"
    Me.Size = New System.Drawing.Size(185, 121)
    Me.ResumeLayout(False)
    Me.PerformLayout()
    Me.PerformLayout()

End Sub
Friend WithEvents myTextBox As System.Windows.Forms.TextBox
Friend WithEvents myListBox As System.Windows.Forms.ListBox

End Class

```

Es gilt dabei dasselbe wie für Formulare: Sobald Sie eine von Control abgeleitete Klasseninstanz der Controls-Auflistung hinzugefügt haben, erscheint es im Steuerelement (gültige Positionseinstellungen vorausgesetzt).

Ebenfalls im Gegensatz zum vorherigen Beispiel müssen wir in unserem Code zumindest die beiden Ereignisse SelectedIndexChanged und SelectedValueChanged neu definieren, denn: Sie werden von myListBox ausgelöst und können bestenfalls durch das Benutzersteuerelement eingebunden werden.

Da es keine Vererbung der `ListBox` gibt, ist mit der Ereigniskette beim Benutzersteuerelement aber auch bereits Schluss. Würden wir die Ereignisse nicht einbinden, unser Steuerelement darüber hinaus selbst mit diesen Ereignissen ausstatten und diese zu gegebener Zeit wieder auslösen, hätte der Entwickler, der unser Steuerelement verwendet, keine Chance, vom Ereigniseintritt zu erfahren – daher die neue Definition.

---

**TIPP:** Wenn Sie komplexere konstituierende Steuerelemente erstellen, und die Steuerelemente, auf denen es basiert, mit Werten vorbelegen müssen, ist der Konstruktor nach `InitializeComponent` die richtige Stelle, um das zu erreichen.

---

## Methoden und Ereignisse delegieren

Was für Ereignisse und Eigenschaften notwendig ist, gilt für Methoden gleichermaßen. Es ist anzunehmen, dass der Entwickler, der Ihr Steuerelement verwendet, bestimmte Funktionen der `TextBox` und der `ListBox` benötigt. Ihnen bleibt auch dann nichts weiter übrig, als die entsprechenden Methoden zu implementieren und die Parameter zum entsprechenden Steuerelement hinabzudelegieren, wie im Beispiel:

```
'Diese Funktionen nach unten durchrouten, weil diese Instanz...
Public Function FindString(ByVal s As String) As Integer
    Return myListBox.FindString(s)
End Function

'...aber auch andere Instanzen sie häufig...
Public Function FindString(ByVal s As String, ByVal startIndex As Integer) As Integer
    Return myListBox.FindString(s, startIndex)
End Function

'...benötigen.
Public Function FindStringExact(ByVal s As String) As Integer
    Return myListBox.FindStringExact(s)
End Function

Private Sub myTextBox_TextChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles myTextBox.TextChanged
    'Löst, durch die Veränderung der Text-Eigenschaft, OnTextChange des UserControls aus
    Me.Text = myTextBox.Text
    Console.WriteLine(Me.Text)
End Sub

'Für unsere Zwecke müssen wir nur dieses Ereignis nach oben routen, aber...
Private Sub myTextBox_KeyDown(ByVal sender As Object, ByVal e As System.Windows.Forms.KeyEventArgs) _
    Handles myTextBox.KeyDown
    Me.OnKeyDown(e)
End Sub

'andere Anwendungen benötigen vielleicht auch dieses...
Private Sub myTextBox_KeyPress(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.KeyPressEventArgs) _
    Handles myTextBox.KeyPress
    Me.OnKeyPress(e)
End Sub
```

```

'...und dieses Ereignis.
Private Sub myTextBox_KeyUp(ByVal sender As Object, ByVal e As System.Windows.Forms.KeyEventArgs) _
    Handles myTextBox.KeyUp
    Me.OnKeyUp(e)
End Sub

'Der Entwickler, der unser Steuerelement verwendet, muss mitbekommen, wenn sich die Auswahl ändert.
'Aus diesem Grund müssen wir diese beiden Ereignisse nach oben durchreichen.
Private Sub myListBox_SelectedIndexChanged(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles myListBox.SelectedIndexChanged
    OnSelectedIndexChanged(e)
End Sub

```

---

**HINWEIS:** Rein theoretisch könnten Sie die Steuerelemente, aus denen das Benutzersteuerelement besteht, natürlich auch *Public* deklarieren – doch davon ist dringend abzuraten! Der Entwickler könnte in diesem Fall Eigenschaften der basierenden Steuerelemente verändern, und das Benutzersteuerelement würde nichts von den Veränderungen mitbekommen – die Funktionalität könnte dadurch stark beeinträchtigt werden – bis hin zum völligen Versagen!

---

## Implementieren der Funktionslogik

Die Implementierung der Funktionslogik ist der des vorherigen Beispiels sehr ähnlich. Der einzige Unterschied: Durch die Natur des Steuerelements ist die Liste ständig sichtbar, aber beide Steuerelemente sind in keiner Weise miteinander verbunden. Die Vorselektierung des Eintrags in der *ListBox*, der zur bisherigen Eingabe in die *TextBox* passt, muss deswegen programmiert werden. Dieses Verhalten lässt sich übrigens mit der *AutoSelect*-Eigenschaft des Benutzersteuerelementes steuern. Bis auf diesen Unterschied werden Sie in der Implementierung überwiegend Gleichheiten zum vorherigen Beispiel bemerken:

```

Protected Overridable Sub OnSelectedIndexChanged(ByVal e As EventArgs)
    'Den Listeneintrag in die TextBox kopieren,
    'dabei ungewollte Rekursion verhindern,
    mySenderIsThis = True
    Me.Text = myListBox.SelectedItem.ToString
    mySenderIsThis = False
    'Ereignis auslösen, damit auch andere Instanzen vom Ereignis erfahren.
    RaiseEvent SelectedIndexChanged(Me, e)
End Sub

Private Sub myListBox_SelectedIndexChanged(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles myListBox.SelectedIndexChanged
    OnSelectedIndexChanged(e)
End Sub

Protected Overridable Sub OnSelectedValueChanged(ByVal e As EventArgs)
    RaiseEvent SelectedValueChanged(Me, e)
End Sub

'Wird aufgerufen, *weil* der Handler myTextBox_TextChanged der TextBox die Text-Eigenschaft
'des UserControls verändert.
'Wird damit aufgerufen, *sobald* irgendeine Veränderung im Eingabefeld vorgenommen wird
'(ganz gleich, ob programmtechnisch oder durch den Anwender).

```

```

Protected Overrides Sub OnTextChanged(ByVal e As System.EventArgs)
    'Dieses Flag wird benötigt, damit eine Eingabebereichsveränderung, die
    'programmtechnisch durch das Steuerelement selbst vorgenommen wird, das Ereignis
    'nicht erneut auslöst, und das Programm in einer Endlosschleife hängen bleibt.
    Console.WriteLine("OnTextChanged")
    If Not mySenderIsThis Then
        'Verhindern, dass das Ereignis durch eigene Manipulation erneut ausgelöst wird.
        mySenderIsThis = True
        'Auto-Vervollständigen durchführen
        PerfAutoCompletion()
        mySenderIsThis = False
    End If
    'Basis-Funktion aufrufen nicht vergessen!
    MyBase.OnTextChanged(e)
End Sub

Private Sub PerfAutoCompletion()

    'Zwischenspeichern, damit die Originaleingabe erhalten bleibt.
    Dim locTemp As String = Me.Text
    'Index des Eintrags finden, der der bisherigen Texteingabe entspricht.
    Dim locIndex As Integer = Me.FindString(locTemp)
    'Nur wenn AutoComplete eingeschaltet ist, überhaupt etwas machen.
    If AutoComplete Then
        If locIndex > -1 Then
            'Eintrag gefunden, zum einfacheren Handling in Variable kopieren.
            Dim locFoundEntry As String = myListBox.Items(locIndex).ToString
            If locIndex > -1 Then
                'Eintrag ins Eingabefeld kopieren.
                myTextBox.Text = locFoundEntry
                'Den noch nicht eingegebenen Teil markieren, damit
                'er einfach überschrieben werden kann.
                myTextBox.Select(locTemp.Length, locFoundEntry.Length - locTemp.Length)
            End If
        End If
    End If
End Sub

'AutoSelect-Eigenschaft im Bedarfsfall umsetzen:
If AutoSelect Then
    'Passenden Eintrag gefunden, ...
    If locIndex > -1 Then
        '...dann selektieren.
        myListBox.SelectedIndex = locIndex
    End If
End If
End Sub

'Wird ausgelöst, wenn eine Taste in der TextBox gedrückt wird.
Protected Overrides Sub OnKeyDown(ByVal e As System.Windows.Forms.KeyEventArgs)
    Console.WriteLine("OnKeyDown")
    'Cursor-nach-unten, dann...
    If e.KeyCode = Keys.Down Then
        'Testen ob die Listbox nicht schon fokussiert ist, und ob bereits Einträge vorhanden sind.

```

```

If Not myListBox.Focused AndAlso myListBox.Items.Count > 0 Then
    '...ListBox fokussieren,
    myListBox.Focus()
    'Wenn noch kein Eintrag in der Listbox ausgewählt war,
    If myListBox.SelectedIndex = -1 Then
        'ersten Eintrag selektieren.
        myListBox.SelectedIndex = 0
    End If
    e.Handled = True
End If
Else
    'Wenn Delete oder Backspace gedrückt wird, dann keine Modifizierungen vornehmen.
    mySenderIsThis = (e.KeyCode = Keys.Delete) Or (e.KeyCode = Keys.Back)
End If
 MyBase.OnKeyDown(e)
End Sub

```

## Implementierung der Eigenschaften

Bei der Implementierung der neuen Eigenschaften verfahren wir fast genauso wie im vorherigen Beispiel – doch da wir es mit einem konstituierenden Steuerelement zu tun haben, ist es damit wieder nicht getan. Einige Eigenschaften müssen wieder nach oben hoch gereicht werden – in diesem Beispiel habe mich auf die Items-Eigenschaft der ListBox und die Text-Eigenschaft der TextBox beschränkt:

```

'Hier kommen die Eigenschaften:
<Description("Bestimmt oder ermittelt, ob die Auto-Ergänzen-Funktion verwendet wird."), _
Category("Verhalten"), _
DefaultValue(GetType(Boolean), "True"), _
Browsable(True)>
Public Property AutoComplete() As Boolean
    Get
        Return myAutoComplete
    End Get
    Set(ByVal Value As Boolean)
        myAutoComplete = Value
    End Set
End Property

<Description("Bestimmt oder ermittelt, ob mit dem Text übereinstimmende Einträge automatisch selektiert werden."), _
Category("Verhalten"), _
DefaultValue(GetType(Boolean), "True"), _
Browsable(True)>
Public Property AutoSelect() As Boolean
    Get
        Return myAutoSelect
    End Get
    Set(ByVal Value As Boolean)
        myAutoSelect = Value
    End Set
End Property

```

```

'Die Texteigenschaft wird vom Designer für das UserControl und deren Ableitungen unterdrückt.
'Mit Browsable(True) machen wir es im Eigenschaftenfenster wieder darstellbar.
<Browsable(True)>
Public Overrides Property Text() As String
    Get
        Return MyBase.Text
    End Get
    Set(ByVal Value As String)
        'Neuen Text der TextBox zuordnen.
        myTextBox.Text = Value
        'WICHTIG: Wenn diese Anweisung fehlt, wird OnTextChanged nicht aufgerufen,
        'wenn die Texteigenschaft neu zugewiesen wird!
        MyBase.Text = Value
    End Set
End Property

<Description("Die Elemente im Listenfeld."), _
Category("Daten"), _
Browsable(True)>
Public ReadOnly Property Items() As ListBox.ObjectCollection
    Get
        Return myListBox.Items
    End Get
End Property

```

End Class

Das Aufrufen der Basisroutine der Text-Eigenschaft ist übrigens in diesem Beispiel besonders wichtig, da andernfalls das OnTextChanged-Ereignis nicht ausgelöst wird. Die folgende Ereigniskette würde dann nämlich unterbrochen, und weder Autokomplettieren noch Vorselektieren in der Liste würden funktionieren:

- Der Anwender gibt ein Zeichen in der TextBox ein.
- Das TextChange-Ereignis wird ausgelöst, das von myTextBox\_TextChanged behandelt wird.
- Diese Ereignisbehandlungsroutine setzt die Text-Eigenschaft des Benutzersteuerelementes mit dem (nun geänderten) Inhalt der TextBox – der Set-Part der Text-Eigenschaftsprozedur wird ausgeführt. Da der Text, der hier myTextBox wieder zugewiesen wird, nicht nur der gleiche, sondern der selbe ist, wird kein erneutes TextChange-Ereignis ausgelöst. Die Gefahr einer unfreiwilligen Rekursion besteht hier also nicht.
- Durch MyBase.Text (vorletzte Zeile im fett markierten Code) ändert sich die Text-Eigenschaft des Benutzersteuerelements, was zur Ausführung von OnTextChanged des Benutzersteuerelements führt. Hier kann jetzt die eigentliche Funktionslogik des Steuerelements ausgeführt werden.

# Erstellen von Steuerelementen von Grund auf

Wenn Sie Steuerelemente erstellen wollen, die nicht auf Funktionalitäten von einem oder mehreren bereits vorhandenen Steuerelementen basieren sollen, dann verfahren Sie prinzipiell so, wie es der ► Abschnitt »Neue Steuerelemente auf Basis vorhandener Steuerelemente implementieren« ab Seite 894 erklärt. Der einzige Unterschied: Ihre neue Steuerelementklasse basiert auf der Control-Klasse. Anstatt also die Klasse mit

```
Public Class ADComboBox  
    Inherits ComboBox
```

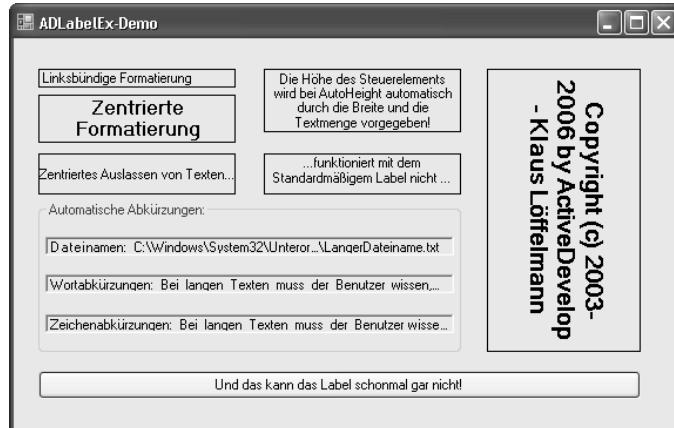
zu definieren, verwenden Sie die Anweisungen

```
Public Class GanzNeuesControl  
    Inherits Control
```

Was das Anlegen einer Projektmappe für ein Steuerelement anbelangt, das Sie von Grund auf neu erstellen wollen, bleibt alles andere genau gleich – deswegen möchte ich die erforderlichen Schritte aus Platzgründen hier nicht wiederholen. Um wirklich professionelle Steuerelemente entwickeln zu können, ist beim eigentlichen Ausformulieren des Codes allerdings viel mehr Handarbeit erforderlich, als bei Steuerelementen, die auf anderen basieren. Haben Sie jedoch erst einmal einen Blick hinter die Kulissen gewagt und verstanden, welche Techniken angewendet werden müssen, um stabile Komponenten dieser Art zu entwerfen, werden Sie diesen Vorzug des Frameworks nicht mehr missen wollen. Mit dieser Methode, wieder verwendbare Komponenten entwerfen zu können, sparen Sie auf Dauer nicht nur eine Menge Arbeit, Sie werden den enormen Komfort, den Sie sich mit dem anfänglich zusätzlichen Aufwand erkaufen müssen, schnell schätzen lernen.

## Ein Label, das endlich alles kann

O.k., »alles« ist vielleicht ein wenig übertrieben, und man mag kaum glauben, dass ein Steuerelement, das auf den ersten Blick nichts weiter macht, als Elemente einer Benutzerumgebung zu beschriften, tatsächlich verbesserungswürdig ist. Doch ich bin mir sicher, dass Sie Ihre Meinung ändern, wenn Sie einen Blick auf das folgende Beispiel geworfen haben.



**Abbildung 30.12:** Dieses Demoprogramm demonstriert die Leistungsstärke des Steuerelementes *ADLabelEx*

---

**BEGLEITDATEIEN:** Sie finden die Projektmappe zu diesem Beispiel im Verzeichnis .\VB 2005 - Entwicklerbuch\G - SmartClient\Kap30\LabelExDemo\LabelExDemo.sln

---

Wenn Sie dieses Programm starten, sehen Sie einen Dialog, etwa wie in Abbildung 30.12 zu sehen. Schon in diesem Dialog werden Sie die Verbesserungen im Vergleich zum herkömmlichen Label-Steuerelement bemerken:

- ADLabelEx ermöglicht es, Beschriftungen mit um 90 Grad gedrehtem Text durchzuführen.
- Mit bestimmten Eigenschaften können automatische Abkürzungen von Text realisiert werden – demonstriert durch die ADLabelEx-Elemente im Rahmen des Formulars. Das Standard-Label vom .NET-Framework kann das dank seiner AutoEllipses-Eigenschaft zwar auch, versagt aber bei der Platzierung des Textes, wenn dieser in der Mitte eines Steuerelements angeordnet werden soll. Außerdem beschränkt es sich auf eine Auslassungsform. ADLabelEx kennt die TextTrimming-Eigenschaft, mit der Sie das Auslassungsverfahren einstellen können.
- Auch hier zu sehen: ADLabelEx verfügt über eine AutoHeight-Eigenschaft, mit der sich das Label in der Höhe automatisch an den Textumfang anpasst.
- Zu sehen, wenn Sie auf die Schaltfläche klicken: Mit der Flash-Eigenschaft können Sie jede ADLabelEx-Instanz zum Blinken bringen. Diese Möglichkeit kann die Benutzerfreundlichkeit von Anwendungen extrem steigern, gerade wenn es um die Darstellung wichtiger Hinweise und Warnungen oder die Kennzeichnung von Eingabefehlern in Formularen geht.

Neugierig geworden? Die folgenden Abschnitte zeigen, wie die verschiedenen Funktionen in der Steuerelementklasse ADLabelEx implementiert wurden.

## Vorüberlegungen und Grundlagenerarbeitung

Die Grundidee für dieses Steuerelement ist eigentlich aus einer realen Notwendigkeit heraus entstanden. Bei der Gestaltung von Formularen mit umfangreichen Eingabefeldern fiel mir auf, dass sich rechtsbündig formatierte Texte im Framework 1.1 nicht wirklich bündig untereinander platzieren ließen. Je länger der Text eines Labels wurde, desto weiter entfernte er sich vom rechten Rand. Da rechtsbündig vor dem Eingabefeld platzierte Labels für die Benutzerfreundlichkeit aber am förderlichsten sind, galt es, eine Lösung zu finden – und diese Bestand in der Entwicklung dieses Steuerelements. Dieses Manko wurde mit dem Framework 2.0 (und insbesondere der UseCompatibleTextRendering-Eigenschaft) zwar behoben, aber viele weitere Features rechtfertigen dieses Demo-Steuerelement natürlich immer noch.

### Textausgabe mit DrawString und dem GDI+

DrawString des GDI+ stellt eigentlich schon alle Funktionen zur Verfügung, die man zur Realisierung eines Label-Steuerelements benötigt. Mit DrawString können Sie bestimmen, wie Texte umgebrochen oder abgekürzt werden sollen, wenn sie nicht in einen definierten Bereich passen. DrawString erlaubt darüber hinaus mit dem StringFormat-Objekt das Einstellen individueller Ausrichtungen (linksbündig, oben; mittig; rechtsbündig unten), wenn der String in einem rechteckigen Bereich ausgegeben werden soll. Allerdings produziert DrawString unter bestimmten Umständen auch Darstellungsfehler, gerade bei zentrierter und rechtsbündiger Formatierung. Der Grund dafür ist Auflösungsunabhängigkeit bei der Verwendung von DrawString:

Die Buchstaben eines Zeichensatzes sind vergleichsweise kleine Objekte, gemessen an der relativ geringen Auflösung, die ein Bildschirm darzustellen in der Lage ist. GDI+ kann deswegen gerade bei kleinen Fonts auf nur relativ wenig Pixel zurückgreifen, um einen Buchstaben auf dem Bildschirm anzuzeigen. Die einzelnen Buchstaben sind aus diesem Grund nicht so lang, wie sie es rein rechnerisch eigentlich sein sollten. Dennoch werden sie in vielen Fällen direkt nebeneinander platziert, sodass sich eine Lücke am Ende des Strings bildet. Und das hat dann genau diesen Effekt: Die theoretisch berechnete Stringlänge weicht von der Länge des tatsächlich dargestellten Textes ab; und dieser Fehler wird umso deutlicher, wenn der auszugebende Text aus vielen Buchstaben besteht. Allerdings – und das ist merkwürdig – zeigt sich dieses Verhalten nur dann, wenn `DrawString` einzeilige Texte auf dem Bildschirm ausgibt. Bei mehrzeiligen Texten ist dieses Verhalten nicht zu beobachten. Und genau das ist auch des Problems Lösung: Aus diesem Grund hängt die `OnPaint`-Funktion des Steuerelements, die den Text in den sichtbaren Bereich des Steuerelements malt, einfach ein Carriage Return (ASCII 13) hinter den eigentlich auszugebenden Text, und schon erscheinen rechtsbündige Texte im vorgegebenen Rahmen wirklich am rechten Rand ausgerichtet.

### **Verwenden von Timer-Objekten zum Auslösen von Ereignissen in Intervallen**

Auch was den Blinkmechanismus anbelangt, sind einige Vorüberlegungen anzustellen, denn: Natürlich muss das Blinken in irgendeiner Form getriggert werden, und dazu bietet sich das `Timer`-Objekt an. Mit einem `Timer`-Objekt haben Sie die Möglichkeit, ein Intervall zu bestimmen (mit seiner `Interval`-Eigenschaft), nach deren Ablauf ein Ereignis (das `Tick`-Ereignis) ausgelöst wird. Anstatt nun für jede in einem Programm vorhandene `ADLabelEx`-Instanz ein eigenes `Timer`-Objekt zu instanzieren – was Verschwendug von Rechenzeit und Ressourcen bedeuten würde – besteht natürlich die ungleich bessere Möglichkeit, nur ein einziges `Timer`-Objekt zu verwenden, das *alle* verwendeten `Label` triggert. Das hat den zusätzlichen Vorteil, dass Texte, wenn sie blinken, synchronisiert blinken. Mit diesen Vorüberlegungen im Hinterkopf können wir uns nun an das eigentliche Codieren des Steuerelements heranwagen:

## **Klasseninitialisierungen und Einrichten der Windows-Darstellungsstile des Steuerelements**

Damit das Steuerelement ordnungsgemäß funktionieren kann, müssen bei seiner Instanzierung zwei grundlegende Dinge gewährleistet sein:

- Klassen-Member-Variablen müssen initialisiert werden.
- Der grundsätzliche Windows-Stil für das Steuerelement muss definiert werden.

Der zweite Punkt bedarf hier einer genaueren Erklärung: Wie Sie in ► Kapitel 28 schon erfahren konnten, basieren sämtliche Bedienungselemente von Benutzeroberflächen unter dem Windows-Betriebssystem auf einem nativen Element, das sich (sehr ungünstig für Erklärungen) *Window* nennt. Ein *Window* im Betriebssystem-Sinn kann die Basis für eine Schaltfläche, einen Tooltip, ein »echtes« Dokumentenfenster, ein .NET-Formular und natürlich auch für ein Steuerelement sein. *Window* ist dabei aber niemals gleich *Window*: Es gibt welche mit und ohne Rollbalken, solche mit Rahmen oder eingelassenem Rahmen im 3D-Lock, Fenster, die grundsätzlich über anderen Fenstern liegen usw.

Wenn ein auf `Control` basierendes Objekt in .NET erstellt wird, hat es durch Überschreiben der `CreateParams`-Eigenschaft die Möglichkeit, diese grundsätzlichen Stile für das zugrunde liegende *Window* zu bestimmen. Auf diese Weise wird beispielsweise für jedes Steuerelement dessen `Borderline`-Eigenschaft umgesetzt; das Zeichnen des entsprechenden Rahmens wird hierbei – entgegen vielen

Vermutungen – nicht durch GDI+-Funktionen, sondern durch das Windows-Betriebssystem selbst realisiert.

Der Klassencode, der zum einen die Member-Variablen mit sinnvollen Werten vorbelegt und zum anderen dafür sorgt, dass das basierende »Windows-Window« korrekt eingerichtet wird, sieht daher folgendermaßen aus:

```
<Designer("ActiveDev.ADLabelExDesigner")> _
Public Class ADLabelEx
    Inherits Control

    Private Const WS_BORDER As Integer = &H800000
    Private Const WS_EX_CLIENTEDGE As Integer = &H200
    Private Const myFlashInterval As Integer = 400

    Private Shared myFlashTimer As Timer

    Private myBorderStyle As BorderStyle
    Private myTextAlign As ContentAlignment
    Private myUseMnemonic As Boolean
    Private myDirectionVertical As Boolean

    Private myAutoHeight As Boolean
    Private myRequestedHeight As Integer
    Private myTextWrap As Boolean
    Private myTextTrimming As StringTrimming

    Sub New()
        MyBase.new()
        'Eigenschaften initialisieren.
        myBorderStyle = BorderStyle.None
        myTextAlign = ContentAlignment.TopLeft
        myUseMnemonic = True
        myTextWrap = True
        myTextTrimming = StringTrimming.None
        myFlashBackColor = Color.Empty
        myFlashForeColor = Color.Empty

        'Windows-Stile setzen.
        SetStyle(ControlStyles.AllPaintingInWmPaint, True)
        SetStyle(ControlStyles.ResizeRedraw, True)
        SetStyle(ControlStyles.DoubleBuffer, True)

        'Initialwert für die Höhe merken.
        myRequestedHeight = Me.Height

        'Flash-Ereignishandler einrichten.
        AddHandler FlashOn, AddressOf FlashOnHandler
        AddHandler FlashOff, AddressOf FlashOffHandler
    End Sub
```

```

'Definiert die Parameter für das Anlegen des "Windows-Window".
Protected Overrides ReadOnly Property CreateParams() As CreateParams

    Get
        Dim params1 As CreateParams
        Dim style1 As BorderStyle
        params1 = MyBase.CreateParams

        'Möglichweise eingeschaltete BorderStyles ausschalten.
        params1.ExStyle = (params1.ExStyle And Not WS_EX_CLIENTEDGE)
        params1.Style = (params1.Style And Not WS_BORDER)

        'Herausfinden, welcher Borderstyle eingeschaltet werden soll.
        style1 = Me.myBorderstyle
        Select Case style1 - 1

            Case 0
                'Simpler Rand
                params1.Style = (params1.Style Or WS_BORDER)

            Case 1
                'Drei-D-Rand
                params1.ExStyle = (params1.ExStyle Or WS_EX_CLIENTEDGE)
        End Select
        Return params1
    End Get
End Property

```

---

**HINWEIS:** CreateParams ist eine Eigenschaft, die das geschützte .NET-Zuhause schon fast verlässt, und mit deren Benutzung Sie sich nahezu inmitten des Windows-Betriebssystem befinden. Durch das CreateParams-Objekt können Sie großen Einfluss auf das Grundaussehen eines Steuerelements oder eines Formulars nehmen. Wenn Sie mehr aus dieser Eigenschaft herausholen wollen, sollten Sie sich mit der Programmierung des Windows-Betriebssystems, insbesondere der nativen Fenstersteuerung, einigermaßen vertraut machen. Die Visual Studio-Hilfe zum Suchbegriff CreateWindowEx (eine native Windows-Funktion) verrät Ihnen mehr zu diesem Thema.

---

Übrigens: Wenn Sie wollen, dass das dem Steuerelement zugrunde liegende »Windows-Window« neu erstellt werden soll, verwenden Sie die Methode UpdateStyles, die eine erneute Abfrage von CreateParams veranlasst.

## Zeichnen des Steuerelements

Das Ausformulieren des Codes zum Zeichnen des Steuerelements bereitet bei dieser Klasse eigentlich die geringsten Schwierigkeiten, da sämtliche umzusetzenden Eigenschaften mit entsprechenden Funktionen des GDI+ realisiert werden können, wie das folgende Codelisting zeigt. Etwas aufwändig ist das Finden der Parameter für das StringFormat-Objekt, das bei DrawString die Textausrichtung steuert – letzten Endes ist das aber nur eine reine Fleißarbeit.

Das eigentliche Zeichnen des Textes geschieht, wie bei allen Steuerelementen und Formularen, in der überschriebenen OnPaint-Methode. Sie wird entweder dann aufgerufen, wenn ein anderes Window-Objekt das Steuerelement verdeckt hatte und es anschließend wieder sichtbar wurde, oder wenn eine

Instanz von außen ein Neuzeichnen des Inhalts erforderlich machte – beispielsweise, wenn sich eine Eigenschaft geändert hat, die die grafische Gestaltung des Steuerelementinhalts in irgendeiner Form beeinflusst.

```
*****  
*** Alles für das Zeichnen *****  
*****  
  
Protected Overrides Sub OnPaint(ByVal e As System.Windows.Forms.PaintEventArgs)  
    'DrawString arbeitet mit RectangleF, ClientRectangle mit Rectangle;  
    'deswegen die Werte ins andere "Format" konvertieren.  
    Dim locRectf As New RectangleF(0, 0, ClientSize.Width, ClientSize.Height)  
    'StringFormat-Objekt für die Ausgabe des Strings erzeugen  
    Dim locSf As StringFormat = CreateStringFormat()  
  
    'Diese "Version" malen, wenn nicht geblinkt wird, oder gerade die Aus-Phase stattfindet.  
    If Not Flash Or Not myFlashState Then  
        'Bei der Ausgabe des Strings "CR" anhängen, damit wird bei rechtsbündiger und  
        'zentrierter Formatierung die richtige Stringlänge berücksichtigt.  
        e.Graphics.DrawString(Text + vbCr, Font, New SolidBrush(ForeColor), locRectf, locSf)  
    Else  
        'Sonst die An-Phase zeichnen, mit FlashBackColor und FlashForeColor.  
        e.Graphics.Clear(FlashBackColor)  
        e.Graphics.DrawString(Text + vbCr, Font, New SolidBrush(FlashForeColor), locRectf, locSf)  
    End If  
    'Keinen Speicher verschwenden!  
    locSf.Dispose()  
End Sub  
  
'Bastelt aus der Einstellung für ContentAlignment das StringFormat-Objekt zusammen,  
'über das diese Eigenschaft bei der Ausgabe mit DrawString umgesetzt wird.  
Protected Overridable Function StringFormatForAlignment(ByVal textAlign As ContentAlignment) _  
As StringFormat  
  
    Dim locStringFormat As New StringFormat  
    If (textAlign And ContentAlignment.BottomLeft) = ContentAlignment.BottomLeft Or  
        (textAlign And ContentAlignment.MiddleLeft) = ContentAlignment.MiddleLeft Or  
        (textAlign And ContentAlignment.TopLeft) = ContentAlignment.TopLeft Then  
        locStringFormat.Alignment = StringAlignment.Near  
    ElseIf (textAlign And ContentAlignment.BottomRight) = ContentAlignment.BottomRight Or  
        (textAlign And ContentAlignment.MiddleRight) = ContentAlignment.MiddleRight Or  
        (textAlign And ContentAlignment.TopRight) = ContentAlignment.TopRight Then  
        locStringFormat.Alignment = StringAlignment.Far  
    ElseIf (textAlign And ContentAlignment.BottomCenter) = ContentAlignment.BottomCenter Or  
        (textAlign And ContentAlignment.MiddleCenter) = ContentAlignment.MiddleCenter Or  
        (textAlign And ContentAlignment.TopCenter) = ContentAlignment.TopCenter Then  
        locStringFormat.Alignment = StringAlignment.Center  
    End If  
  
    If (textAlign And ContentAlignment.TopLeft) = ContentAlignment.TopLeft Or  
        (textAlign And ContentAlignment.TopRight) = ContentAlignment.TopRight Or  
        (textAlign And ContentAlignment.TopCenter) = ContentAlignment.TopCenter Then  
        locStringFormat.LineAlignment = StringAlignment.Near
```

```

ElseIf (textAlign And ContentAlignment.MiddleLeft) = ContentAlignment.MiddleLeft Or
    (textAlign And ContentAlignment.MiddleRight) = ContentAlignment.MiddleRight Or
    (textAlign And ContentAlignment.MiddleCenter) = ContentAlignment.MiddleCenter Then
    locStringFormat.LineAlignment = StringAlignment.Center
ElseIf (textAlign And ContentAlignment.BottomLeft) = ContentAlignment.BottomLeft Or
    (textAlign And ContentAlignment.BottomCenter) = ContentAlignment.BottomCenter Or
    (textAlign And ContentAlignment.BottomRight) = ContentAlignment.BottomRight Then
    locStringFormat.LineAlignment = StringAlignment.Far
End If
Return locStringFormat

End Function

'Baut das StringFormat-Objekt zusammen und berücksichtigt nicht nur ContentAlignment,
'sondern auch andere Eigenschaften des AdLabelEx-Steuerelements.
Protected Overrides Function CreateStringFormat() As StringFormat

    Dim locStringFormat As StringFormat
    'Grundsätzliche Einstellungen aufgrund des ContentAlignment holen.
    locStringFormat = StringFormatForAlignment(Me.TextAlign)
    'RightToLeft-Einstellung für Arabische Sprachen berücksichtigen
    If Me.RightToLeft = RightToLeft.Yes Then
        locStringFormat.FormatFlags = locStringFormat.FormatFlags Or _
            StringFormatFlags.DirectionRightToLeft
    End If

    'Zugriffstastenanzeige berücksichtigen.
    If Not Me.UseMnemonic Then
        locStringFormat.HotkeyPrefix = System.Drawing.Text.HotkeyPrefix.None
    Else
        If (Me>ShowKeyboardCues) Then
            locStringFormat.HotkeyPrefix = System.Drawing.Text.HotkeyPrefix.Show
        Else
            locStringFormat.HotkeyPrefix = System.Drawing.Text.HotkeyPrefix.Hide
        End If
    End If

    'If Me.AutoSize Then
    '    locStringFormat.FormatFlags = locStringFormat.FormatFlags Or _
        StringFormatFlags.MeasureTrailingSpaces
    'End If

    'Möglichst genaue Formatierung.
    locStringFormat.FormatFlags = locStringFormat.FormatFlags Or StringFormatFlags.FitBlackBox
    'LineLimit wird nicht berücksichtigt, wenn der Text nicht in den Rahmen passt
    locStringFormat.FormatFlags = locStringFormat.FormatFlags And Not StringFormatFlags.LineLimit

    'Text um 90 Grad im Uhrzeigersinn drehen?
    If DirectionVertical Then
        locStringFormat.FormatFlags = locStringFormat.FormatFlags Or StringFormatFlags.DirectionVertical
    End If

    'Textwrapping eingeschaltet?

```

```

If Not TextWrap Then
    locStringFormat.FormatFlags = locStringFormat.FormatFlags Or StringFormatFlags.NoWrap
End If

'Das Trimming definieren.
locStringFormat.Trimming = TextTrimming

'Wert zurückgeben.
Return locStringFormat
End Function

```

Die fett hervorgehobenen Codezeilen im oben gezeigten Listing zeichnen übrigens den Text in den Clientbereich des Steuerelements. Dabei muss natürlich auch ein möglicherweise eingeschaltetes Blinken (Flash = True) berücksichtigt werden. Wie das Blinken des Steuerelements implementiert ist, erfahren Sie in einem späteren Abschnitt.

---

**TIPP:** Das Verändern bestimmter Eigenschaften führt oft dazu, dass sich das Aussehen eines Steuerelements ändert und es daher neu gezeichnet werden soll. Wenn Sie wollen, dass sich ein Steuerelement neu zeichnet, stehen Ihnen dazu prinzipiell drei verschiedene Vorgehensweisen zur Verfügung, nämlich mit Refresh, Invalidate und Update. Der folgende Abschnitt macht die Unterschiede in der Anwendung deutlich:

---

## Der Unterschied zwischen Refresh, Invalidate und Update

Aus vielen Diskussionen im Usenet wird klar, dass es Unklarheiten über die richtige Anwendung und Funktionsweise dieser drei Funktionen gibt, da sie alle drei Ähnliches bewirken. Fakt ist, dass sie tatsächlich viel miteinander zu tun haben, sich teilweise auch gegenseitig aufrufen. Invalidate verwenden Sie, wenn Sie einen bestimmten oder den ganzen Bereich des Clientbereiches eines Fensters oder Steuerelements neu zeichnen lassen wollen. Invalidate löst dann das Aufrufen der OnPaint-Methode aus, die für das Neuzeichnen des Clientbereichs Sorge tragen muss. Allerdings macht sie das nicht sofort, sondern erst, wenn die Nachrichtenwarteschlange die nächste Gelegenheit bekommt, ausstehende Nachrichten zu verarbeiten. Das ist aber erst dann der Fall, wenn Ihre Anwendung in den so genannten *Idle*<sup>6</sup>-Modus wechselt. In einigen Fällen kann es aber notwendig sein, dass das Steuerelement (oder Formular) seinen Clientbereich sofort neu zeichnen soll. In diesem Fall ist ein anschließendes Update notwendig, das die OnPaint-Methode und das dazugehörige Ereignis auslöst.

Refresh wiederum fasst beide Methoden unter einem Dach zusammen und macht noch ein bisschen mehr: Es bewirkt mit einem Invalidate(True), dass auch alle untergeordneten Steuerelemente neu gezeichnet werden und erzwingt mit einem anschließenden Update darüber hinaus das unmittelbare Neuzeichnen.

---

<sup>6</sup> Auf Deutsch etwa »brachliegend«, »unbeschäftigt«, »untätig« (interessanterweise auch »faul«, »nutzlos«, »müßig«). Kleiner Tipp: Wenn Sie per Ereignis benachrichtigt werden wollen, dass Ihre Applikation in den *Idle*-Modus gewechselt ist, binden Sie via AddHandler das Application.Idle-Ereignis ein. Hinweis: Sie können auch per Anweisung dafür sorgen, dass die Anwendung die Nachrichtenwarteschlange abarbeitet, sie also bewusst für einen kurzen Moment in den Idle-Modus bringen: Der unter den VB6 Entwicklern so beliebte Befehl DoEvents steht auch in .NET als Methode des Application-Objekts zur Verfügung.

## Größenbeeinflussung durch andere Eigenschaften

Bestimmte Steuerelemente können nicht nur durch die `Size`-Eigenschaft, sondern auch durch andere Eigenschaften größtmäßig beeinflusst werden. Sie haben das bestimmt schon selbst bei der `TextBox` erlebt: Wenn Sie eine `TextBox` in einem Formular platzieren, so kann sie in ihrer Höhe nicht verändert werden – standardmäßig ist sie nämlich für den einzeiligen Betrieb vorgesehen. Die `TextBox` merkt sich die Höhe, in der sie hätte platziert werden sollen, aber sehr wohl: Sobald Sie nämlich ihre `MultiLine`-Eigenschaft auf `True` setzen, »springt« sie förmlich auseinander, und zwar nimmt sie dann genau die Größe an, die sie schon beim ersten Aufziehen im Formular hätte haben sollen.

Wenn die Größe eines Steuerelements oder Formulars verändert werden soll, ist die Methode `SetBoundsCore` der `Control`-Basisklasse für diese Aufgabe zuständig. Sie sorgt dafür, dass die entsprechenden Windows-Nachrichten an das Betriebssystem gesendet werden, um das Steuerelement auf die richtige Größe zu bringen. Um diesen Vorgang zu reglementieren (also beispielsweise zu verhindern, dass eine bestimmte Höhe überschritten wird), kann eine vererbte Klasse diese Methode überschreiben und veränderte Parameter für die Steuerelementausmaße an die Basismethode übergeben.

Um zu gewährleisten, dass ein ursprünglich zugewiesener Wert für die Höhe oder die Breite erhalten bleibt (die `TextBox` beispielsweise beim Setzen von `MultiLine` auf die ursprüngliche Höhe wieder zurückspringt), implementiert jedes Steuerelement, das dieses Verhalten an den Tag legt, eine oder zwei Variablen,<sup>7</sup> die die Ausgangsausmaße für die jeweilige Dimension zwischenspeichern.

Diese Zwischenspeicher dürfen aber nur dann neu gesetzt werden, wenn eine Eigenschaft die entsprechende Dimension gezielt überschrieben hat. Wurde beispielsweise nur die `Height`-Eigenschaft des Steuerelements verändert, darf sich das nicht auf den zwischengespeicherten Wert für die Breite (`requestedWidth`) auswirken. Aus diesem Grund wird `SetBoundsCore` neben Parametern, die die kompletten neuen (oder eben alten) Ausmaße des Steuerelements enthalten, ein weiterer Parameter namens `specified` übergeben, der bestimmt, welcher Ausmaßparameter (X, Y, Breite, Höhe) gezielt verändert wurde.

Unser Beispielsteuerelement implementiert dieses Verhalten übrigens auch. Wenn die `AutoHeight`-Eigenschaft des Steuerelements auf `True` gesetzt wird, passt es sich höhenmäßig an die Ausmaße des Textes an, vergrößert bzw. verkleinert sich automatisch. Das funktioniert auch dann, wenn Sie anschließend die Breite des Steuerelements verschieben. Setzen Sie die `AutoHeight`-Eigenschaft anschließend wieder zurück, nimmt das Steuerelement seine ursprüngliche Größe an.

Sobald die `AutoHeight`-Eigenschaft gesetzt wurde, lässt sich die Höhe des Steuerelements nicht mehr verändern. Für dieses Verhalten ist aber nicht die eigentliche Steuerelementklasse verantwortlich, sondern eine weitere, die Sie im Projektmappen-Explorer unter `ADLabelExDesigner` finden. Im ► Abschnitt »Designer-Reglementierungen« ab Seite 931 finden Sie mehr zu diesem Thema.

Das folgende Codelisting zeigt wie die Größensteuerung des Steuerelements implementiert wurde.

```
'Die neue Höhe einstellen. Diese Methode wird aufgerufen, wenn sich eine Eigenschaft  
'geändert hat, die die Höhe des Steuerelements beeinflusst, und wenn AutoHeight eingeschaltet ist.  
Private Sub AdjustHeight()  
  
    Dim locRequestedHeightTemp As Integer
```

---

<sup>7</sup> Es hat sich eingebürgert, diese Variablen `requestedHeight` und `requestedWidth` zu nennen.

```

locRequestedHeightTemp = myRequestedHeight
Try
    If AutoHeight Then
        MyBase.Size = New Size(Me.Size.Width, PreferredHeight)
    Else
        MyBase.Size = New Size(Me.Size.Width, locRequestedHeightTemp)
    End If
Finally
    myRequestedHeight = locRequestedHeightTemp
End Try
End Sub

'Ermittelt die Höhe des Textes bei einer bestimmten Breite. Diese Funktion
'wird von AdjustHeight für die automatische Höhenanpassung des Steuerelements verwendet.
Public Overridable ReadOnly Property PreferredHeight() As Integer
    Get
        Dim locHeightToReturn As Integer
        If Me.Text = "" Then
            locHeightToReturn = Me.FontHeight
        Else
            Dim locG As Graphics
            Dim locSf As StringFormat
            Dim locSizeF As SizeF
            locG = Graphics.FromHwnd(Me.Handle) ' Graphics-Objekt erzeugen.
            locSf = CreateStringFormat()          ' Gleiches StringFormat wie beim Ausgeben.
            'Texthöhe automatisch ermittelt. Das erreichen Sie, wenn Sie für die Höhe 0 übergeben.
            locSizeF = locG.MeasureString(Text, Font, New SizeF(ClientSize.Width, 0), locSf)
            'Immer nach unten abrunden!
            locHeightToReturn = CInt(Math.Ceiling(locSizeF.Height))
        End If
        'Falls es einen Borderstyle gibt, 2 Pixel draufrechnen, damit es nicht
        'zu gequetscht wird.
        If BorderStyle <> BorderStyle.None Then
            locHeightToReturn += 2
        End If
        Return locHeightToReturn
    End Get
End Property

'Setzt alle Ausmaße des Steuerelements oder nur bestimmte Größenkomponenten,
'die von Specified bestimmt werden.
Protected Overrides Sub SetBoundsCore(ByVal x As Integer, ByVal y As Integer, ByVal width As Integer,
                                     ByVal height As Integer, ByVal specified As System.Windows.Forms.BoundsSpecified)
    Dim locRect As New Rectangle

    'Falls AutoHeight eingeschaltet ist...
    If AutoHeight Then
        '...und die Breite bestimmt werden soll...
        If (specified And BoundsSpecified.Width) = BoundsSpecified.Width Then
            '...dann die neue Breite im Steuerelement setzen...
            MyBase.SetBoundsCore(x, y, width, height, specified)
            '...jetzt muss aber auch die Höhe neu errechnet werden...

```

```

AdjustHeight()
'...und wenn die zwischengespeicherte Höhe gesetzt war...
If myRequestedHeight > 0 Then
    'dann bricht der Vorgang hier ab. Andernfalls wurde myRequestedHeight nicht
    'initialisiert, und zwar dadurch, dass Height noch 0 war, als das
    'Steuerelement erstellt wurde. Erst die erste Zuweisung der Size-Eigenschaft
    'bestimmt die Höhe, die aber selbst mit SetBoundsCore gesetzt wird. Aus diesem
    'Grund kann myRequestedHeight beim ersten Durchlauf keinen anderen Wert als
    '0 haben und muss entsprechend initialisiert werden.
    Return
End If
End If

'Aktuelle Ausmaße zwischenspeichern.
locRect = Me.Bounds
If (specified And BoundsSpecified.Height) = BoundsSpecified.Height Then
    'myRequestedHeight wird neu definiert, wenn die Höhe (zum Beispiel durch Size)
    'explizit zugewiesen wird. Am vom SetBoundsCore "verlangten" Height
    'ändert sich nur dann was...
    myRequestedHeight = height
End If

'....wenn AutoHeight eingeschaltet ist. Dann wird die Höhe des Steuerelements auf die
'gemessene Höhe des Textes festgeschrieben.
If (Me.AutoHeight AndAlso (locRect.Height <> height)) Then
    height = Me.PreferredHeight
End If

'Basis aufrufen
 MyBase.SetBoundsCore(x, y, width, height, specified)
End Sub

```

## Implementierung der Blink-Funktionalität

Wie in der Einführung schon erwähnt, kam es bei der Implementierung der Blink-Funktionalität darauf an, möglichst wenige Ressourcen zu verwenden und das Blinken der einzelnen ADLabelEx-Instanzen untereinander zu synchronisieren. Aus diesem Grund gibt es nur einen einzigen Timer, der für das Triggern des Blinkens zuständig ist. Und da es nur einen Timer gibt, ist es fast schon selbstverständlich, dass er als statische Instanz im Steuerelement implementiert wurde.

Der Timer selbst wird erst dann instanziert, wenn das erste ADLabelEx seine Flash-Eigenschaft auf True setzt und damit beginnen will zu blinken. Eine statische Zählvariable sorgt dafür, dass der Timer weiß, wie viele Instanzen von ihm Gebrauch machen. Wenn die Zählvariable wieder den Wert 0 erreicht hat, wird die Dispose-Methode des Timers ausgeführt, um ihn ordnungsgemäß und rückstandsfrei zu entsorgen.

Damit jede ADLabelEx-Instanz über das Eintreten des Tick-Ereignisses im Bedarfsfall informiert wird, meldet sie ihre Ereignisbehandlungsrouterien FlashOnHandler und FlashOffHandler schon in ihrem Konstruktor mit AddHandler an. Diese Ereignisse werden durch die statische Ereignisbehandlungsroutine FlashTimeHandler für alle ADLabelEx-Instanzen ausgelöst. Damit dieser zentrale Ereignisverteiler

selbst vom Ablauf des Timers durch das Eintreten seines Tick-Ereignisses erfährt, sorgt die Routine StartFlashHandlerOnDemand für das Zuweisen der Ereignisbehandlungsroutine ebenfalls durch eine AddHandler-Anweisung. Der typische Weg der Ereigniskette ist also folgender:

- Eine ADLabelEx-Instanz wird instanziert, und sie bindet die Ereignisbehandlungs routinen mit AddHandler schon im Konstruktor ein.
- Ihre Flash-Eigenschaft wird auf True gesetzt, und das bewirkt den Aufruf von StartFlashHandler- OnDemand, das das Timer-Objekt instanziert, startet und die zentralen Ereignisbehandlungsroutine FlashTimeHandler einrichtet.
- Der Timer läuft irgendwann ab und löst sein Tick-Ereignis aus. Das wiederum löst den Aufruf von FlashTimeHandler aus, und diese Routine löst, in Abhängigkeit vom Phasen-Flag myFlashState, entweder das FlashOn- oder das FlashOff-Ereignis aus.
- Da jede ADLabelEx-Instanz diese Ereignisse im Konstruktor registriert hat, werden die Ereignisbe handlungs routinen FlashOnHandler und FlashOffHandler jeder Instanz aufgerufen.
- Wenn die Flash-Eigenschaft einer Instanz nicht gesetzt ist, erfolgt ein sofortiger Rücksprung aus der Behandlungsroutine – es passiert gar nichts. Andernfalls lösen beide Routinen mit Invalidate ein Neuzeichnen des Steuerelementeinhals mit der dem jeweiligen Phasenzustand entsprechenden Farbeinstellung aus. In der Aus-Phase werden FlashBackColor und FlashForeColor für das Zeichnen verwendet, in der An-Phase BackColor und ForeColor. Durch geschicktes Wählen der Farben entsteht der Eindruck eines rhythmischen Blinkens.
- Wenn eine Steuerelementinstanz entsorgt wird und ihre Flash-Eigenschaft zuvor verwendet wurde, meldet sie sich in ihrer Dispose-Methode durch Aufruf der Prozedur StopFlashHandlerOnDemand wieder ab. Der *Instanzzähler* wird heruntergezählt, und der Timer wird erst dann ordnungsgemäß entsorgt, wenn die letzte ADLabelEx-Instanz ihn nicht mehr benötigt.

Der Code, der diese Funktionalität implementiert, sieht folgendermaßen aus:

```

'*****
'*** Flash-Handling
'*****
Private Shared myFlashState As Boolean
Private Shared myFlashTimerUseCounter As Integer
Private myFlashTimerUsed As Boolean
Private myFlash As Boolean
Private myFlashBackColor As Color
Private myFlashForeColor As Color
Public Shared Event FlashOn(ByVal sender As Object, ByVal e As EventArgs)
Public Shared Event FlashOff(ByVal sender As Object, ByVal e As EventArgs)

'Es gibt einen einzigen Timer für alle blinkenden ADLabelEx-Instanzen. Alles andere
'wäre Verschwendun von Ressourcen.
'Und auch der eine Timer wird erst dann angeworfen, wenn das erste ADLabelEX
'blinken will.
Private Shared Sub StartFlashHandlerOnDemand()
    If myFlashTimerUseCounter = 0 Then
        myFlashTimer = New Timer
        myFlashTimer.Interval = myFlashInterval
        myFlashTimer.Start()
        'Hier wird die Ereignisbehandlungsroutine eingebunden, die beim
        'Ablauen von myFlashInterval-Millisekunden (also alle 300) aufgerufen wird.

```

```

        AddHandler myFlashTimer.Tick, AddressOf FlashTimeHandler
    End If
    'Damit die Steuerelement-Klasse weiß, wie viele Instanzen blinken,
    'gibt es einen Zähler...
    myFlashTimerUseCounter += 1
End Sub

Private Shared Sub StopFlashHandlerOnDemand()
    myFlashTimerUseCounter -= 1
    '....damit das Timer-Objekt ordnungsgemäß entladen werden kann,
    'wenn es nicht mehr benötigt wird.
    If myFlashTimerUseCounter = 0 Then
        myFlashTimer.Stop()
        myFlashTimer.Dispose()
    End If
End Sub

'Dispose wird benötigt, damit der letzte das Licht (den Timer) ausmachen kann!
Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
    If disposing Then
        If myFlashTimerUsed Then
            RemoveHandler FlashOn, AddressOf FlashOnHandler
            RemoveHandler FlashOff, AddressOf FlashOffHandler
            StopFlashHandlerOnDemand()
        End If
    End If
    MyBase.Dispose(disposing)
End Sub

'Dieser private Ereignishandler löst zwei neue Ereignisse aus, die öffentlich empfangen werden
'können. Es gibt jeweils für beginnende An- und Aus-Phase ein Ereignis.
Private Shared Sub FlashTimeHandler(ByVal sender As Object, ByVal e As EventArgs)
    myFlashState = Not myFlashState
    If myFlashState Then
        RaiseEvent FlashOn("ADLabelEx.FlashHandler", e.Empty)
    Else
        RaiseEvent FlashOff("ADLabelEx.FlashHandler", e.Empty)
    End If
End Sub

Protected Overridable Sub FlashOnHandler(ByVal sender As Object, ByVal e As EventArgs)
    'Da die Ereignis-Handler schon im Konstruktor eingebunden werden (war einfacher ;-)
    'treten die Ereignisse auch auf, wenn ein anderes ADLabelEx blinken will.
    'Deswegen muss diese Instanz testen, ob sie blinken darf.
    If Not myFlash Then Return
    'Alles weitere regelt OnPaint...
    Me.Invalidate()
End Sub

'Dasselbe in blau/schwarz.
Protected Overridable Sub FlashOffHandler(ByVal sender As Object, ByVal e As EventArgs)
    If Not myFlash Then Return
    Me.Invalidate()
End Sub

```

## Designercode-Generierung und Zurücksetzen von werteerbenden Eigenschaften mit ShouldSerializeXXX und ResetXXX

Besondere Beachtung verdienen die beiden Eigenschaften FlashBackColor und FlashForeColor, da es sich bei ihnen um so genannte »werteerbende Eigenschaften« handelt. Solange Sie ihnen keine speziellen Werte zuweisen, liefern sie den Wert der BackColor-Eigenschaft zurück. Ändern Sie den Wert der BackColor-Eigenschaft, ändern sich auch die Werte von FlashBackColor und FlashForeColor. Der fett markierte Bereich zeigt, wie diese Implementierung vonstatten geht.

Nun gibt es keinen fest definierten Standardwert für diese beiden Eigenschaften, weil sie vom Wert einer anderen abhängig sind. Damit diese beiden Eigenschaften nicht unnötigerweise serialisiert werden (d.h., entsprechender Initialisierungscode in InitializeComponent für die Instanz erzeugt wird, die das ADLabelEx einbindet), versagt das schon bekannte DefaultValue-Attribut an dieser Stelle. In solchen Fällen implementieren Sie eine Funktion, die genauso lautet wie die eigentliche Eigenschaft und zusätzlich das Präfix ShouldSerialize trägt. Durch Reflection ermittelt der Designer vor der Codegenerierung das Vorhandensein einer solchen Funktion und ruft sie auf, um herauszufinden, ob die Serialisierung der Eigenschaft notwendig ist. Die Funktion bestimmt also anhand eines bestimmten Algorithmus und nicht auf Grund eines konstanten Wertes, ob die Code Serialisierung notwendig ist.

Damit die Reset-Funktion des Eigenschaftenfensters für eine derartige Eigenschaft funktionieren kann, implementieren Sie eine weitere Funktion, die den Namen der Eigenschaft und das Präfix Reset trägt. Diese Funktion stellt dann im Bedarfsfall (der Anwender wählt aus dem Kontextmenü der Eigenschaft im Eigenschaftenfenster *Zurücksetzen*) den gültigen Ausgangswert wieder her.

```
<DefaultValue(GetType(Boolean), "False"), _
Category("Darstellung"), _
Description("Bestimmt, ob der Label-Text blinked angezeigt werden soll."), _
Browsable(True)>
Public Property Flash() As Boolean
    Get
        Return myFlash
    End Get
    Set(ByVal Value As Boolean)
        myFlash = Value
        'Im Entwurfsmodus wird nicht geblinkt!
        If Not DesignMode Then
            If Value Then
                'Erst das erste Setzen initialisiert den Blink-Timer,
                'aber nur beim ersten Mal!
                If Not myFlashTimerUsed Then
                    StartFlashHandlerOnDemand()
                End If
                myFlashTimerUsed = True
            Else
                'Alten Zustand wiederherstellen
                Invalidate()
            End If
        End If
    End Set
End Property
```

```

<Category("Darstellung"), _
Description("Bestimmt die Hintergrundfarbe beim Blinken, wenn sich das Steuerelement in der An-Phase befindet."), _
Browsable(True)>
Public Property FlashBackColor() As Color
    Get
        'Hier läuft's anders mit den Standardwerten. Wenn keine Farbe definiert ist,
        '"erbt" diese Eigenschaft von BackColor. Dadurch muss nur BackColor verändert
        'werden, um auch FlashBackColor zu verändern. Allerdings gibt es damit keinen
        'festen Standardwert...
        If myFlashBackColor.Equals(Color.Empty) Then
            Return BackColor
        Else
            Return myFlashBackColor
        End If
    End Get

    Set(ByVal Value As Color)
        If Value.Equals(BackColor) Then
            myFlashBackColor = Color.Empty
        Else
            myFlashBackColor = Value
        End If
    End Set
End Property

'...deswegen muss mit einer Funktion ermittelt werden, ob der aktuelle Wert der Standardwert ist.
'Nur wenn er es nicht ist, wird serialisiert (Code für die Eigenschaft in der sie einbindenden
'Instanz erzeugt).
Public Function ShouldSerializeFlashBackColor() As Boolean
    Return Not myFlashBackColor.Equals(Color.Empty)
End Function

'Damit wird die Reset-Funktion für diese Eigenschaft im Eigenschaftenfenster
'(Kontext-Menü über der Eigenschaft) aktiviert.
Public Sub ResetFlashBackColor()
    myFlashBackColor = Color.Empty
End Sub

<Category("Darstellung"), _
Description("Bestimmt die Vordergrundfarbe beim Blinken, wenn sich das Steuerelement in der An-Phase befindet."), _
Browsable(True)>
Public Property FlashForeColor() As Color
    Get
        If myFlashForeColor.Equals(Color.Empty) Then
            Return BackColor
        Else
            Return myFlashForeColor
        End If
    End Get

    Set(ByVal Value As Color)
        If Value.Equals(BackColor) Then
            myFlashForeColor = Color.Empty

```

```

    Else
        myFlashForeColor = Value
    End If
End Set
End Property

Public Function ShouldSerializeFlashForeColor() As Boolean
    Return Not myFlashForeColor.Equals(Color.Empty)
End Function

Public Sub ResetFlashForeColor()
    myFlashForeColor = Color.Empty
End Sub
End Class

```

## Designer-Reglementierungen

Wie Sie vielleicht wissen, verfügt jedes Steuerelement über seinen eigenen Designer. Visual Studio .NET hat also keine Funktionalität zum interaktiven Verändern der einzelnen Steuerelemente implementiert, sondern fungiert nur als so genannter *Designer Host*. Für das Zur-Verfügung-Stellen eines Designers, mit dem der Anwender zum Entwurfsmodus das Steuerelement beispielsweise in seinen Ausmaßen verändern kann, ist also jedes Steuerelement selbst verantwortlich.

Natürlich brauchen Sie nicht für jedes Steuerelement, das Sie neu erstellen, eine vollständige Designer-Logik von Grund auf zu entwerfen. Im günstigsten Falle müssen Sie überhaupt nichts machen. Wenn Sie keinen speziellen Designer für Ihr Steuerelement definiert haben, verwendet jeder Designer-Host automatisch den Standard-Control-Designer, mit dem Ihr Steuerelement zur Entwurfszeit bearbeitet werden kann. Soll Ihr Steuerelement jedoch zur Entwurfszeit design-technisch in irgend-einer Form von der Norm abweichen, müssen Sie selbst Hand anlegen.

In unserem Fall ist das notwendig, da das Steuerelement nicht in der Höhe verändert werden darf, wenn seine AutoHeight-Eigenschaft gesetzt ist. Der folgende Code zeigt, wie eine solche Implementierung vonstatten geht.

---

**HINWEIS:** Damit ein individueller Designer und nicht der *Standard Control Designer* vom Designer Host verwendet wird, müssen Sie Ihre Steuerelementklasse mit einem Attribut namens *Designer* ausstatten. Dieses Attribut muss über der Klassendefinition der eigentlichen Steuerelementklasse platziert werden, etwa folgendermaßen für dieses Beispiel:

---

```

<Designer("ActiveDevelop.Controls.ALabelExDesigner")> _
Public Class ALabelEx
    Inherits Control
    .
    .

```

Wenn Sie einen eigenen Designer implementieren, erben Sie ihn aus der Klasse *ControlDesigner*. Die Funktionen, die Sie verändern möchten, überschreiben Sie in dieser Klasse. Ebenfalls wichtig: Damit Sie auf die Designerfunktionalität des Frameworks zurückgreifen können, müssen Sie einen Verweis auf die Assembly *System.Design.Dll* in Ihr Projekt einfügen. Das Importieren des Namespaces *System.Windows.Forms.Design* ist darüber hinaus erforderlich.

```

Imports System.Windows.Forms
Imports System.Drawing
Imports System.ComponentModel
Imports System.Windows.Forms.Design

'WICHTIG: Wenn Sie einen ControlDesigner einfügen,
'müssen Sie den System.Windows.Forms.Design-Namespace einbinden,
'und System.Design.dll als Verweis dem Projekt hinzufügen!
Public Class ADLabelExDesigner
    Inherits ControlDesigner

    'Muss überschrieben werden, damit bei einem Steuerelement mit fixer Größe in
    'einer vertikaler Richtung tatsächlich nur eine vertikale Größenänderung möglich wird.
    'Die vertikalen Anfasspunkte sind dann ausgeblendet
    Public Overrides ReadOnly Property SelectionRules() As System.Windows.Forms.Design.SelectionRules
        Get
            Dim locThisComponent As Object
            Dim locSelectionRules As SelectionRules
            locThisComponent = Me.Component
            Try
                'In Abhängigkeit von ConsiderFixedSize (die sich beispielsweise durch Multiline ändert)
                If Convert.ToBoolean(TypeDescriptor.GetProperties(locThisComponent).Item("AutoHeight").GetValue(locThisComponent)) _
                    Then
                        'Nur vertikale Größenveränderungen...
                        locSelectionRules = SelectionRules.Moveable Or SelectionRules.Visible Or _
                            SelectionRules.LeftSizeable Or SelectionRules.RightSizeable
                    Else
                        '...oder komplette Größenveränderungen ermöglichen
                        locSelectionRules = SelectionRules.Moveable Or SelectionRules.Visible Or _
                            SelectionRules.AllSizeable
                    End If
                    Return locSelectionRules
                Catch ex As Exception
                    Debug.WriteLine("Designermassage:" & ex.Message)
                    Return MyBase.SelectionRules
                End Try
            End Get
        End Property
    End Class

```

# 31

# Mehreres zur gleichen Zeit erledigen – Threading in .NET

---

- 936 Threads durch ein Thread-Objekt initiieren
  - 939 Synchronisieren von Threads
  - 953 Verwenden von Steuerelementen in Threads
  - 954 Managen von Threads
  - 959 Datenaustausch zwischen Threads durch Kapseln von Threads in Klassen
  - 970 Verwenden des Thread-Pools
  - 975 Thread-sichere Formulare in Klassen kapseln
  - 978 Threads durch den Background-Worker initiieren
  - 981 Threads durch asynchrone Aufrufe von Delegaten initiieren
- 

Dass Gleichzeitigkeit eigentlich eine Illusion ist, hat Einstein mit seiner Relativitätstheorie bewiesen.<sup>1</sup> Wenn auch aus anderen Gründen, besteht dennoch der Eindruck, dass ein normaler Computer Dinge wirklich gleichzeitig erledigen könnte. Auch wenn er im Hintergrund Robbie Williams spielt, eine seiner Festplatten defragmentiert und mich gleichzeitig diese Zeilen mit Word schreiben lässt, so zerlegt er diese drei Sachen in klitzekleine Aufgaben und arbeitet sie im Grunde genommen nacheinander ab. Aus der Geschwindigkeit, mit der er diese kleinen Dinge hintereinander macht, entsteht dann der Eindruck, er mache sie wirklich gleichzeitig.

Ausnahmen davon bilden Multiprozessor- oder Multicore-Systeme, die bestimmte Aufgaben tatsächlich gleichzeitig erledigen können. Anmerkung am Rande: Die so genannten *Hyperthreading*-Prozessoren von Intel nehmen dabei eine Art Zwitterstellung ein – sie nutzen Pausen, die der Prozessor von Zeit zur Zeit einlegen muss, wenn er beispielsweise auf Daten aus dem Hauptspeicher wartet, um schon mal Teile anderer Threads zu verarbeiten, sodass dieser Prozessor auf unterster Ebene betrachtet Gleichzeitigkeit ziemlich perfekt vortäuscht. In der Tat führt das zu einer Leistungssteigerung von durchschnittlich 10 % bis zu ca. maximal und unter günstigsten Umständen 20 %, doch im Grunde genommen arbeiten auch Hyperthreading-Prozessoren die zu erledigenden Threads auch nur nacheinander ab. Echte Gleichzeitigkeit ist allein Multiprozessor- bzw. Multicore-Systemen vorbehalten.

---

<sup>1</sup> Siehe auch die Folge von Alpha Centauri (vom 10.6.2001), zu erreichen über den IntelliLink G3101.

Doch gerade die letzten Systemtypen machen genau in diesem Moment, in dem die Zeilen entstehen, Schluss mit dem Taktfrequenzrennen der verschiedenen Prozessorhersteller. Da die Grenze des physikalisch Möglichen quasi erreicht ist, können Prozessoren nicht mehr wesentlich schneller werden: man kann Ihnen nur beibringen, mehrere Dinge wirklich gleichzeitig zu machen, und genau dahin geht der Trend. Dummerweise bedeutet es nicht automatisch, dass ein Prozessor, der im Grunde genommen zwei oder vier Prozessoren in sich vereint, auch automatisch das zwei- bzw. vierfache an Leistung bringt. Nur, wenn eine Software auch in der Lage ist, die beiden Prozessorkerne auch wirklich zu nutzen, indem sie eine oder mehrere Aufgaben (so genannte *Tasks*) in verschiedene Threads verlagert, die dann wiederum unabhängig auf mehreren Prozessorkernen laufen können, können Sie als Anwender (und als Entwickler zeitkritischer Systeme) auch wirklich von dieser neuen Technologie profitieren.

Sie sehen also, dass dieses Kapitel eines der wirklich wichtigen Themen der kommenden Zeit behandelt.

Unter dem Namen »Multitasking« hat wohl jeder, der sich nur ein wenig mit Computern beschäftigt, diese Fähigkeit schon einmal kennen gelernt. Multitasking ist die Kombination der englischen Wörter »multi« – für »viel« – und »task« – für »Aufgabe« – und bedeutet im Deutschen das, was Frauen beneidenswerter- und normalerweise eher können als Männer: nämlich mehrere Dinge zur gleichen Zeit erledigen.

Ein weiterer, ähnlicher Begriff, stammt ebenfalls aus dem Englischen, aber er ist nicht ganz so bekannt wie »Multitasking«. Gemeint ist »Multithreading«<sup>2</sup>, wieder abgeleitet von »multi« – für »viel« – nur im zweiten Teil des Wortes diesmal von »thread« – für »Faden«. Man könnte eine Multithreading-fähige Anwendung also als »mehrädiges« Programm bezeichnen, wollte man den Ausdruck unbedingt übersetzen.

Und was bedeutet dieser Ausdruck jetzt genau? Dazu folgender Hintergrund: wenn Sie eine Windows-Applikation starten, dann besteht sie aus mindestens einem so genannten Thread. In diesem Zusammenhang ist »Thread« eine abstrakte Bezeichnung für einen bestimmten Programmverlauf. Ein Thread startet, löst eine bestimmte Aufgabe und wird beendet, wenn diese Aufgabe erledigt ist. Wenn diese Aufgabe sehr rechenintensiv ist, dann bedeutet das in der Regel für das Programm, dass es nicht weiter bedienbar ist. Der aktuelle Thread beansprucht die gesamte Rechenleistung eines Prozessorkerns (und wenn der Computer nur über einen Prozessorkern verfügt, seine komplette Rechenleistung), sodass für die Behandlung der Bedienungselemente nichts mehr übrig bleibt, wie das folgende kleine Beispielprogramm eindrucksvoll zeigt:

---

**BEGLEITDATEIEN:** Sie finden dieses Projekt im Verzeichnis *.\VB 2005 - Entwicklerbuch\G - SmartClient\Kap31\Single-Threading*.

---

Dieses Programm macht nichts anderes, als eine Zählvariable bis auf einen bestimmten Wert hinaufzuzählen und den aktuellen Wert im Label-Steuerelement anzuzeigen.

---

<sup>2</sup> Ausgesprochen etwa »Maltifrädding«, und wenn Sie es ganz perfekt machen wollen, liseln Sie das scharfe S.

```

Public Class Form1

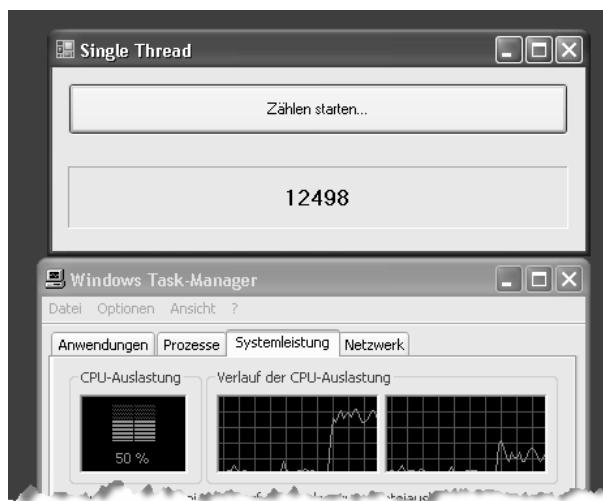
    Private Sub btnZählenStarten_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
        Handles btnZählenStarten.Click
        For z As Integer = 0 To 100000
            lblAusgabe.Text = z.ToString
            lblAusgabe.Refresh()
        Next
    End Sub
End Class

```

Wenn Sie dieses Programm gestartet haben, klicken Sie auf die Schaltfläche *Zählen starten*.

Ein Blick in den Task-Manager offenbart, dass es fast alles an Prozessorleistung verschlingt. Da es darüber hinaus keine Anstalten macht, das Formular die Warteschleife verarbeiten zu lassen, lässt es sich, während es läuft, auch nicht bedienen – Sie können das Programmfenster nicht verschieben oder schließen; obwohl die Zählung läuft, scheint es zu hängen.

Sie sehen: In dem Moment, in dem Sie Ihr Programm eine umfangreiche Verarbeitung von Daten durchführen lassen müssen, sollten Sie auf eine andere Technik ausweichen, damit andere Funktionen des Programms nach wie vor zur Verfügung stehen können. Und hier kommt die Thread-Programmierung ins Spiel. Threads sind Programmteile, die unabhängig voneinander und quasi gleichzeitig operieren können. So könnte beispielsweise die eigentliche Zählschleife des Programms in einem eigenen Thread laufen. Sie würde in diesem Fall parallel zum eigentlichen Programm ausgeführt werden. Das Programm könnte sich dann nicht nur um seine Nachrichtenwarteschlange kümmern und das Programm so bedienbar halten, sondern sich zusätzlich um die Ausführung weiterer Aufgaben kümmern.



**Abbildung 31.1:** Sobald Sie das Programm starten, verschlingt es die komplette Prozessorleistung eines Prozessorkerns. Darüber hinaus lässt die Schleife keine Nachrichtenauswertungen zu, sodass sich das Programm während des Zählens nicht anderweitig bedienen lässt.

Beispiele, wann Threads »angesagt« sind, gibt es viele:

- Ihr Programm muss eine umfangreiche Auswertung von Daten zusammenstellen und drucken. Diese Aufgabe könnte in einem eigenen Thread im Hintergrund passieren, ohne dass es den weiteren Arbeitsablauf des Programms stören würde.

- Ihr Programm muss umfangreiche Datensicherungen durchführen. Ein Thread könnte Momentaufnahmen dieser Dateien erstellen und sie anschließend sozusagen im Hintergrund auf eine andere Ressource kopieren.
- Ihr Programm muss Zustände bestimmter Hardwarekomponenten aufzeichnen – beispielsweise Produktionsdaten von Maschinen abrufen. Auch diese Funktion könnte im Hintergrund ablaufen; Auswertungsfunktionen könnten dennoch zu jeder Zeit parallel laufen und vom Anwender verwendet werden, wenn die Produktionsdatenerfassung als Thread im Hintergrund läuft.
- Bei komplizierten Berechnungen oder Auswertungen könnten zwei oder mehrere Threads diese Aufgabe übernehmen. In diesem Fall würde auf Hyperthreading- oder Multiprozessorsystemen eine Auslastung mehrerer Prozessoren durch *jeweils* einen eigenen Thread die Verarbeitungsgeschwindigkeit der gesamten Aufgabe deutlich erhöhen.

Anwendungen gibt es also viele, um Threads einzusetzen. Allerdings gibt es bei Threads auch ein paar Dinge, die beachtet werden müssen, denn: Sie dürfen sich nicht gegenseitig ins Gehege kommen. Doch dazu später mehr.

Betrachten wir zunächst die Grundlagen, also wie wir das Framework überhaupt dazu bewegen können, dass bestimmte Teile einer Anwendung als eigener Thread ausgeführt werden können.

---

**HINWEIS:** Sollten Sie einfach nur die Anforderung haben, eine bestimmte Aufgabe in einem anderen Thread erledigen zu lassen, und möchten Sie dazu nicht tiefer in die Materie einsteigen, empfehle ich Ihnen den Einsatz der `BackgroundWorker`-Komponente, die Sie im ► Abschnitt »Threads durch den Background-Worker initiieren« ab Seite 978 beschrieben finden.

---

## Threads durch ein Thread-Objekt initiieren

Das folgende Beispiel zeigt am einfachsten Beispiel, wie Sie eine Prozedur Ihrer Klasse mithilfe der `Thread`-Klasse als Thread parallel zum aktuellen Programm ablaufen lassen können. Alle weiteren Techniken des Threading wird dieses Kapitel übrigens anhand dieser Klasse besprechen.

---

**BEGLEITDATEIEN:** Sie finden dieses Beispiel im Verzeichnis `.\VB 2005 - Entwicklerbuch\G - Smart-Client\Kap31\SimpleThread01`.

---

**WICHTIG:** In den Beispielen dieses Kapitels finden Sie des Öfteren eine Klasse, die die Ausgabe von Zeilen in einem speziellen Dialog ermöglicht. Diese Klasse nennt sich `ADThreadSafeInfoBox`, und sie stellt die Methoden `TSWrite` und `TSWriteLine` zur Verfügung. Sie ist für die Verwendung in Windows Forms-Anwendungen gedacht, und um sie zu verwenden, müssen Sie sie nicht instanzieren, sondern können ihre Funktionen wie die der `Console`-Klasse direkt verwenden, da sie statisch sind. Ihre Verwendung ist notwendig, da die Aktualisierung von Steuerelementen in Formularen aus Thread-Prozeduren eine besondere Vorgehensweise erforderlich macht. Auf dieses Thema werde ich jedoch am Ende dieses Kapitels genauer eingehen. Für den Moment arbeiten Sie mit der Klasse einfach so, als wäre sie fest im Framework vorhanden.

---

Wenn Sie dieses Programm starten, sehen Sie einen simplen Dialog, der lediglich aus zwei Schaltflächen besteht. Sobald Sie die Schaltfläche *Thread starten* anklicken, öffnet sich ein weiteres Fenster, in dem ein Wert von 0 bis 50 hoch gezählt wird. Soweit ist das noch nichts Besonderes. Allerdings können Sie die Schaltfläche ein weiteres Mal anklicken, um einen weiteren Thread zu starten. Auch der zweite Thread führt die Zählung durch, und das Ausgabefenster zeigt dabei die Ergebnisse beider Zahlenfolgen an – etwa wie in Abbildung 31.2 zu sehen:



**Abbildung 31.2:** Zwei Threads laufen parallel und teilen sich das Ausgabefenster, um Ergebnisse ihres Schaffens anzuzeigen

Soweit, so gut. Nun lassen Sie uns als nächstes den Code betrachten, der dieses Ergebnis im Ausgabefenster zustande bringt:

```
Imports System.Threading

Public Class frmMain

    'Member-Variablen, damit die Threads durchnummieriert werden können.
    'Dient nur zur späteren Unterscheidung des laufenden Threads, wenn
    'er Ergebnisse im Ausgabefenster darstellt.
    Private myArbeitsThreadNr As Integer = 1

    'Dies ist der eigentliche Arbeits-Thread (Worker Thread), der das
    'Hochzählen und die Werteausgabe übernimmt
    Private Sub UmfangreicheBerechnung()
        For c As Integer = 0 To 50
            'Dient zur Ausgabe des Wertes. TSWriteLine ist eine statische
            'Prozedur, die für die Darstellung des Fensters selbst sorgt,
            'sobald sie das erste Mal verwendet wird.
            ADThreadSafeInfoBox.TSWriteLine(Thread.CurrentThread.Name + ":: " + c.ToString)
```

```

'Aktuellen Thread um 100ms verzögern, damit die ganze
'Geschichte nicht zu schnell vorbei ist.
Thread.Sleep(100)
Next
End Sub

Private Sub btnThreadStarten_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnThreadStarten.Click
'Dieses Objekt kapselt den eigentlichen Thread
Dim locThread As Thread
'Dieses Objekt benötigen Sie, um die Prozedur zu bestimmen,
'die den Thread ausführt.
Dim locThreadStart As ThreadStart

'Threadausführende Prozedur bestimmen
locThreadStart = New ThreadStart(AddressOf UmfangreicheBerechnung)
'ThreadStart-Objekt dem Thread-Objekt übergeben
locThread = New Thread(locThreadStart)
'Thread-Namen bestimmen
locThread.Name = "Arbeits-Thread: " + myArbeitsThreadNr.ToString
'Thread starten
locThread.Start()
'Zähler, damit die Threads durch ihre Namen unterschieden werden können
myArbeitsThreadNr += 1

End Sub

Private Sub btnBeenden_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles _
    btnBeenden.Click
'Einfach so geht's normalerweise nicht
Me.Close()
End Sub

End Class

```

## Starten von Threads

Wie aus dem Listing des vorherigen Beispielprogramms ersichtlich, benötigen Sie zwei Objekte, um einen Thread tatsächlich zum Laufen zu bringen: das Thread- und das ThreadStart-Objekt. Das ThreadStart-Objekt dient lediglich dazu, die Adresse der Prozedur aufzunehmen, die als Thread ausgeführt werden soll. Sie können es als Delegaten mit besonderen Eigenschaften betrachten, der darauf ausgerichtet ist, mit dem eigentlichen Thread-Objekt zusammenzuarbeiten. Nachdem Sie das ThreadStart-Objekt instanziert und ihm dabei die Thread-Prozedur mit AddressOf zugespielt haben, übergeben Sie die generierte Instanz dem Thread-Konstruktor. Haben Sie auch dieses Objekt erzeugt, starten Sie den Thread mit der Start-Methode.

## Grundsätzliches über Threads

Jedes Thread-Objekt, das Sie auf die beschriebene Weise erzeugt haben, speichert spezifische Informationen über den eigentlichen Thread. Das ist notwendig, da Windows auf Basis des so genannten *Preemptive Multitasking*<sup>3</sup> arbeitet, bei dem Prozessorzeit den verschiedenen Threads durch das Betriebssystem zugewiesen wird.<sup>4</sup> Wenn das Betriebssystem bestimmt, dass es Zeit für die Ausführung eines bestimmten Threads wird, müssen beispielsweise der komplette Zustand der CPU-Register für den laufenden Thread gesichert und der ursprüngliche Zustand der Register für den als nächstes auszuführenden Thread wiederhergestellt werden. Diese Daten werden unter anderem in einem bestimmten Speicherbereich gesichert, den das Thread-Objekt kapselt. Sie werden für gewöhnlich auch als *Thread Context* bezeichnet.

Thread-Prozeduren sind im Grunde genommen nichts Besonderes. Eine Thread-Prozedur ist grundsätzlich – wie jede andere Prozedur auch – ein Teil einer Klasse oder eines Moduls (das ja im Grunde genommen auch nicht weiter als eine Klasse mit nur statischen Funktionen ist). Die lokalen Variablen, die die Thread-Prozedur verwendet, sind für den jeweils ausgeführten Thread unterschiedlich. Anders ausgedrückt, können lokale Variablen eines Threads also nicht denen eines anderen ins Gehege kommen. Anders ist das beim Zugriff auf Klassen-Member: Für jeden Thread gilt dieselbe Instanz einer Member-Variablen, und dabei können sich besondere Probleme ergeben:

Stellen Sie sich vor, ein Thread greift auf eine Member-Variablen der Klasse zu, um sie beispielsweise neu zu berechnen und anschließend auszugeben. Genau in dem Moment, in dem der erste Thread sie berechnet hat, aber noch bevor er dazu gekommen ist, das berechnete Ergebnis tatsächlich auf dem Bildschirm auszugeben, hat ein zweiter Thread die Berechnung mit dem Member abgeschlossen. In der Member-Variablen steht nun ein aus Sicht des ersten Threads völlig falsches Ergebnis, und das ausgegebene Ergebnis des ersten Threads ist ebenso falsch.

## Synchronisieren von Threads

Damit Zugriffs- bzw. Synchronisationsprobleme verhindert werden können, gibt es eine ganze Reihe von Techniken, die in diesem Abschnitt besprochen werden sollen.

Die einfachste Methode erlaubt das automatische Synchronisieren eines Codeblocks auf Grund einer verwendeten Objektvariablen. Der folgende Abschnitt erläutert das Problem und dessen Lösung anhand eines konkreten Beispiels.

## Synchronisieren der Codeausführung mit SyncLock

Das nächste Beispielprogramm lehnt sich an das des vorherigen Beispiels an – es ist nur ein wenig »eloquenter«. Lediglich um zu zeigen, inwieweit nicht synchronisierte Vorgänge beim Threading richtig daneben gehen können, modifiziert es die Ausgaberoutine der Thread-Routine auf folgende Weise:

<sup>3</sup> Etwa »bevorrechtigt«, »präventiv«.

<sup>4</sup> Im Gegensatz dazu gibt es das so genannte *Cooperative Multitasking*, bei dem ein Thread selber bestimmt, wie viel Prozessorzeit er benötigt. Dadurch kann ein Thread, der in einer nicht enden wollenden Operation fest hängt, die Stabilität des gesamten Systems gefährden.

---

**BEGLEITDATEIEN:** Sie finden dieses Beispiel im Verzeichnis .\VB 2005 - Entwicklerbuch\G - Smart-Client\Kap31\SimpleThread02 (SyncLock).

---

```
'Member-Variablen, mit der demonstrativ Mist gebaut wird...
Private myThreadString As String

'Dies ist der eigentliche Arbeits-Thread (auch "Worker Thread" genannt),
'der das Hochzählen und die Werteausgabe übernimmt.
Private Sub UmfangreicheBerechnung()

    Dim strTemp As String

    For c As Integer = 0 To 50
        strTemp = Thread.CurrentThread.Name + ":: " + c.ToString

        'Nehmen Sie die Auskommentierung von SyncLock zurück,
        'um den "Fehler" des Programms zu beheben.
        'SyncLock Me
        myThreadString = ""

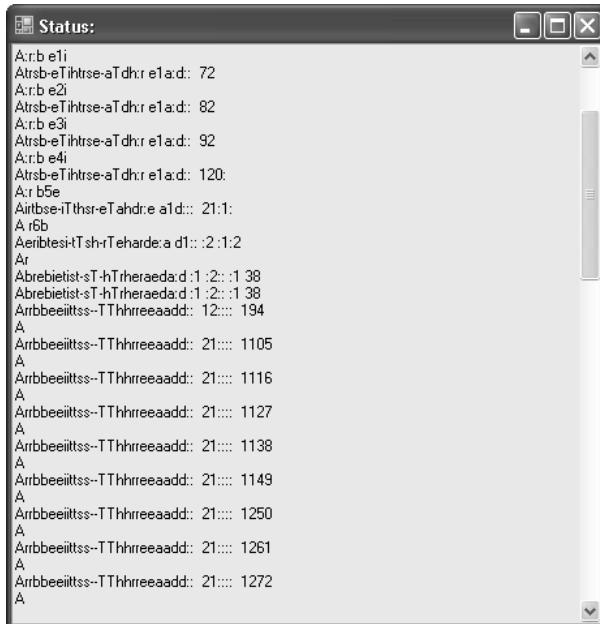
        For z As Integer = 0 To strTemp.Length - 1
            myThreadString += strTemp.Substring(z, 1)
            Thread.Sleep(5)
        Next

        ADThreadSafeInfoBox.TSWriteLine(myThreadString)
        'End SyncLock

    Next
End Sub
```

Hier passiert folgendes: `myThreadString` ist eine Member-Variable der Klasse, und sie wird für jedes Zeichen, das im Ausgabefenster für einen Zählungseintrag erscheinen soll, Zeichen für Zeichen zusammengesetzt. Die `TSWriteLine`-Methode gibt diesen String anschließend aus, wenn das Zusammenbasteln des Strings für einen Eintrag erledigt ist.

Wenn Sie das Programm starten, und die Schaltfläche nur ein einziges Mal anklicken, sodass auch nur ein einziger Thread läuft, bleibt alles beim Alten. Doch wehe, sie starten, schon während der erste Threads läuft, auch nur einen weiteren, dann nämlich ist Chaos angesagt, wie auch in Abbildung 31.3 zu sehen.



**Abbildung 31.3:** Zwei Threads im Kampf um eine Member-Variable – und das daraus resultierende und nicht wirklich zufrieden stellende Ergebnis

Das Problem ist, nicht genau voraussehen zu können, wann eine Thread-Prozedur zum Zug kommt – denn beim *Preemptive Multitasking* bestimmt diesen Zeitpunkt das Betriebssystem. Im hier vorliegenden Fall »meint« das Betriebssystem, dass ein anderer Thread am besten zum Zuge kommen kann, wenn der eine Thread gerade zu warten beginnt (was beide durch die *Sleep*-Methode in regelmäßigen Abständen machen).

Dieses Problem können Sie durch das Blocken von Codeabschnitten in Abhängigkeit von verwendeten Objekten aus der Welt schaffen. Die *SyncLock*-Anweisung ist hier der Schlüssel zur Lösung. Wenn Sie den Code folgendermaßen umbauen, kann sich das Ergebnis wieder sehen lassen:

```
'Dies ist der eigentliche Arbeits-Thread (Worker Thread), der das
'Hochzählen und die Werteausgabe übernimmt.
Private Sub UmfangreicheBerechnung()
    Dim strTemp As String

    For c As Integer = 0 To 50
        strTemp = Thread.CurrentThread.Name + ":: " + c.ToString
        SyncLock Me
            myThreadString = ""
            For z As Integer = 0 To strTemp.Length - 1
                myThreadString += strTemp.Substring(z, 1)
                Thread.Sleep(5)
            Next
            'ADThreadSafeInfoBox.TSWriteLine(myThreadString)
            Console.WriteLine(myThreadString)
        End SyncLock
    Next
End Sub
```

SyncLock verwendet ein Objekt, um den nachfolgenden Code für alle anderen Threads zu schützen, die einen Verweis auf dasselbe Objekt halten. Dieses Objekt sollte daher für die Dauer des Zugriffs nicht verändert werden, da der Schutz sonst nicht oder nicht mehr zuverlässig funktioniert.

---

**WICHTIG:** Im Gegensatz zu vielen vorherrschenden Meinungen schützt SyncLock nicht das Objekt selbst vor Veränderungen, sondern den Code gegen den gleichzeitigen Zugriff von einem anderen Thread. Natürlich kann jeder andere Thread das Objekt verändern und so die ganze Bemühung zur Synchronisation des Programms an dieser Stelle zunichte machen. Außerdem sollte am Rande erwähnt werden: Da SyncLock intern einen Try/Catch-Block verwendet und damit auch im Falle einer Ausnahme der Schutz des Blocks wieder aufgehoben werden kann, dürfen Sie nicht mit der Goto-Anweisung in einen SyncLock-Block springen.

---

Was passiert also im Detail bei der Ausführung dieser Routine? Nun, sobald der erste Thread den Block erreicht, sperrt er den Zugriff auf den Code für jeden weiteren Thread durch Zuhilfenahme des Objektes. Für dieses Objekt müssen folgende Bedingungen gelten:

- Es muss sich um ein Objekt handeln, das auf eine gültige Adresse im Managed Heap verweist – darf also nicht Nothing sein.
- Es muss eine Member-Variable sein, also jedem Thread den Zugriff darauf ermöglichen.
- Es muss zwingend ein Referenztyp sein (logisch, denn Wertetypen liegen nicht auf dem Managed Heap).
- Es darf durch die geschützte Routine nicht verändert werden.

Im Beispiel wird das einfach durch die Verwendung der eigenen Instanz Me erreicht. Me erfüllt diese Bedingungen stets. Bei statischen Routinen, bei denen Me natürlich nicht anwendbar ist, können Sie das Typ-Objekt der verwendeten Klasse verwenden, um es als Hilfe zum Schutz einer Routine zu verwenden, etwa:

```
.  
. .  
SyncLock (GetType(Klassename))  
  
'Hier steht der Code, der zu schützenden Routine  
  
End SyncLock  
. .
```

Wenn nun ein zweiter Thread den geschützten Programmteil erreicht, dann wird er so lange in die Knie gezwungen, bis der Thread, der den Programmteil bereits durchläuft, End SyncLock erreicht hat.

In unserem Beispiel kann also der Member-String ohne Sorge zu Ende aufgebaut und anschließend verarbeitet werden. Dass ein anderer Thread den Member-String an dieser Stelle in irgendeiner Form verändert, ist ausgeschlossen, weil er auf alle Fälle zu warten hat.

---

**HINWEIS:** Einen Programmteil, der besonderen Synchronisationsschutz benötigt bzw. erfährt, nennt man *kritischer Abschnitt* oder neudeutsch: *Critical Section*.

---

## Mehr Flexibilität in kritischen Abschnitten mit der Monitor-Klasse

Die SyncLock-Funktion hat einen großen Nachteil: Sie ist unerbittlich. Wenn ein anderer Thread bereits einen kritischen Abschnitt durchläuft, dann warten alle anderen Threads am Anfang des kritischen Abschnitts – ob sie wollen oder nicht. Sie können dagegen nichts tun. Die Monitor-Klasse bietet an dieser Stelle die größere Flexibilität, da sie ein paar zusätzliche Methoden im Angebot hat, mit der ein Thread auch nachschauen kann, ob er warten müsste, wenn er einen kritischen Bereich beträte. Ein weiteres Beispiel soll das verdeutlichen:

---

**BEGLEITDATEIEN:** Betrachten Sie den Code des folgenden Beispielprogramms, das Sie im Verzeichnis .\VB 2005 - Entwicklerbuch\G - SmartClient\Kap31\SimpleThread03 (Monitor)\.

---

```
'Dies ist der eigentliche Arbeits-Thread (Worker Thread), der das
'Hochzählen und die Werteausgabe übernimmt.
Private Sub UmfangreicheBerechnung()

    Dim strTemp As String

    For c As Integer = 0 To 50
        strTemp = Thread.CurrentThread.Name + ":: " + c.ToString
        'Nehmen Sie die Auskommentierung von SyncLock zurück,
        'um den "Fehler" des Programms zu beheben.
        If Not Monitor.TryEnter(myLock, 1) Then
            ADThreadSafeInfoBox.TSWriteLine(Thread.CurrentThread.Name + " meldet: Gerade besetzt!")
            Thread.Sleep(50)
        Else
            myThreadString = ""
            For z As Integer = 0 To strTemp.Length - 1
                myThreadString += strTemp.Substring(z, 1)
                Thread.Sleep(5)
            Next
            ADThreadSafeInfoBox.TSWriteLine(myThreadString)
            Monitor.Exit(myLock)
        End If
    Next
End Sub
```

In dieser Version des Arbeits-Thread erfolgt die Synchronisation durch die Monitor-Klasse. Allerdings ist der Thread, der auf seinen Vorgänger warten muss, ein wenig ungeduldig. Bekommt er nicht innerhalb von einer Millisekunde Zugriff auf den kritischen Abschnitt, verarbeitet er den Code für die entsprechenden Werterhöhung nicht, sondern gibt nur lapidar die Meldung *Threadname meldet: Gerade besetzt!* aus.



**Abbildung 31.4:** Ein Thread, der auf den kritischen Abschnitt nicht zugreifen kann, wartet maximal eine Millisekunde, bevor er mit einer lapidaren Meldung den Wartevorgang abbricht

Möglich wird das durch die Verwendung der `TryEnter`-Methode der `Monitor`-Klasse – die im Übrigen statisch ist; Sie brauchen die `Monitor`-Klasse also nicht zu instanzieren (Sie könnten es auch gar nicht). Diese Methode überprüft, ob der Zugriff auf einen kritischen Abschnitt, der durch ein angebautes Objekt genau wie bei `Synclock` gesteuert wird, freigegeben ist. Ist er freigegeben, liefert sie `True` als Funktionsergebnis zurück. Ist er nicht freigegeben, ergibt sie `False`.

Auf diese Weise gibt es beim Ablauf des Programms Effekte, wie Sie sie auch in Abbildung 31.4 beobachten können.

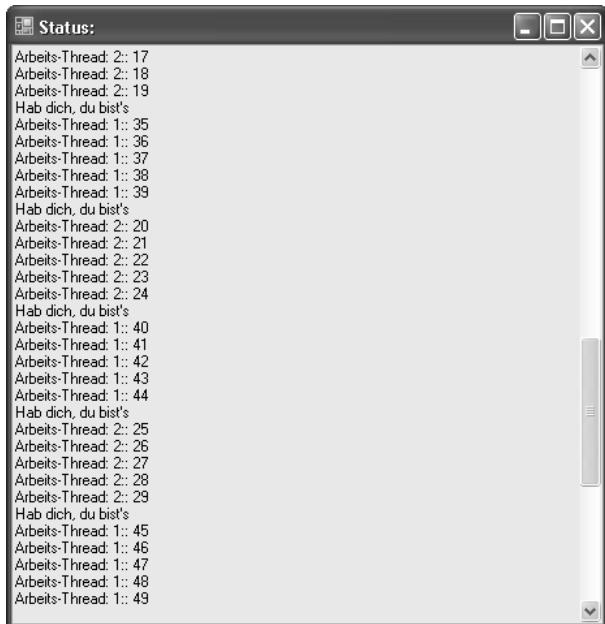
Doch die `Monitor`-Klasse kann noch mehr, wie das folgende Beispiel zeigt.

Angenommen Sie möchten, dass im bislang gezeigten Beispielprogramm die durchgeführten Operationen in Fünferportionen durchgeführt werden. In diesem Fall benötigen Sie drei weitere Funktionen oder besser: Funktionalitäten:

- Sie benötigen eine Methode, die einen Thread in einen Wartezustand versetzt, wenn er sich innerhalb eines kritischen Bereichs befindet.
- Sie benötigen eine Methode, die einen Thread, der sich in einen Wartezustand versetzt hat, wieder aufwecken kann. Gibt es mehrere wartende Threads, sollte die Funktion in der Lage sein, nicht nur alle, sondern auch nur einen (beliebigen) Thread wieder zum Leben zu erwecken.

Diese Möglichkeiten bzw. Methoden gibt es. Sie heißen `Monitor.Wait`, `Monitor.PulseAll` und `Monitor.Pulse`. Für unser Vorhaben brauchen wir darüber hinaus eine Technik, die es uns ermöglicht, zwischen einem und mehreren laufenden Threads zu unterscheiden. Denn solange nur ein einzelner Thread läuft, darf der sich natürlich nicht in den Schlafmodus versetzen, da es keinen anderen gibt, der ihn wieder wach rütteln könnte. Doch diese Technik zu implementieren ist simpel: Eine einfache, statische Thread-Zählvariable vollführt diesen Trick. Wird ein neuer Thread gestartet, wird der Zähler zu Beginn der Thread-Prozedur hoch gezählt; beendet er seine Arbeit, zählt er ihn wieder

runter. Er versetzt sich nur dann in Schlaf, wenn noch mindestens zwei Threads laufen. Getreu dem Motto »Der Letzte macht das Licht aus« muss jeder Thread vor dem Verlassen seiner Arbeitsprozedur noch überprüfen, ob es weitere Threads gibt und den jeweils letzten prophylaktisch aus seinem Dornrösenschlaf erwecken.



**Abbildung 31.5:** Ein Thread, der auf den kritischen Abschnitt nicht zugreifen kann, wartet maximal eine Millisekunde, bevor er mit einer lapidaren Meldung den Wartevorgang abbricht

---

**BEGLEITDATEIEN:** Sie finden dieses Projekt im Verzeichnis .\VB 2005 - Entwicklerbuch\G - Smart-Client\Kap31\SimpleThread04 (Monitor Wait Pulse)\

---

Wenn Sie dieses Programm starten und mindestens zwei Mal auf die Schaltfläche zum Starten des Threads klicken, sehen Sie nach einer Weile ein Ergebnis, wie es in etwa dem in Abbildung 31.5 gezeigten entspricht.

Das entsprechende Listing des Arbeits-Threads sieht folgendermaßen aus:

```
'Die brauchen wir, um festzustellen, wie viele Threads unterwegs sind.  
Private myThreadCount As Integer  
  
'Dies ist der eigentliche Arbeits-Thread (Worker Thread), der das  
'Hochzählen und die Werteausgabe übernimmt.  
Private Sub UmfangreicheBerechnung()  
  
    Dim strTemp As String  
    Dim stepCount As Integer  
  
    'Neuer Thread: Zählen!  
    myThreadCount += 1
```

```

'Hier beginnt der kritische Abschnitt.
Monitor.Enter(Me)
For c As Integer = 0 To 50
    strTemp = Thread.CurrentThread.Name + ":: " + c.ToString
    'Der Thread wartet maximal eine Sekunde; bekommt er in dieser
    'Zeit keinen Zugriff auf den Code, steigt er aus.
    'Zugriff wurde gewährt - jetzt nimmt der Arbeits-Thread
    'erst seine eigentliche Arbeit auf.
    myThreadString = ""
    For z As Integer = 0 To strTemp.Length - 1
        myThreadString += strTemp.Substring(z, 1)
        Thread.Sleep(1)
    Next
    ADThreadSafeInfoBox.TSWriteLine(myThreadString)
    'Der Thread darf sich nur dann schlafen legen, wenn mindestens
    'ein weiterer Thread unterwegs ist, der ihn wieder wecken kann.
    If myThreadCount > 1 Then
        stepCount += 1
        If stepCount = 5 Then
            stepCount = 0
            'Ablösung naht!
            Monitor.Pulse(Me)
            ADThreadSafeInfoBox.TSWriteLine("Hab dich, du bist's")
            'Abgelöster geht schlafen.
            Monitor.Wait(Me)
        End If
    End If
    Next
    If myThreadCount > 1 Then
        'Alle anderen schlafen zu dieser Zeit.
        'Also mindestens einen wecken, bevor dieser Thread geht.
        'Passiert das nicht, schlafen die anderen bis zum nächsten Stromausfall...
        Monitor.Pulse(Me)
    End If
    'Hier ist der kritische Abschitt wieder vorbei.
    Monitor.Exit(Me)
    'Thread-Zähler vermindern.
    myThreadCount -= 1
End Sub

```

## Synchronisieren von beschränkten Ressourcen mit Mutex

Wenn Threads lediglich theoretische Aufgaben lösen müssen, ist es fast egal, wie viele Threads gleichzeitig laufen (natürlich sollten dabei Sie berücksichtigen, dass Sie grundsätzlich nur so viele Threads wie gerade nötig verwenden sollten, da das Umschalten zwischen den Threads selbst natürlich auch nicht wenig an Rechenleistung verschlingt). Doch wenn bestimmte Anwendungen Komponenten der nur begrenzt vorhandenen Hardware benötigen, dann müssen diese Komponenten zwischen den verschiedenen Threads ideal aufgeteilt werden.

An dieser Stelle kommt die Mutex-Klasse ins Spiel. Ihr Vorteil: Sie funktioniert zwar prinzipiell wie die Monitor-Klasse, doch sie ist instanzierbar. Das bedeutet: Sie können durch Mutex-Instanzen, deren Anzahl Sie von der Verfügbarkeit benötigter Hardwarekomponenten abhängig machen, einen Verteiler organisieren, der sich um die Zuteilung der jeweils nächsten freien Hardware-Komponente kümmert.

Das folgende Beispiel demonstriert den Umgang mit Mutex-Klassen. Damit dieses Beispiel die Mutex-Klasse auch hardwareunabhängig auf jedem Rechner demonstrieren kann, ist es ein wenig anders aufgebaut als die Beispiele, die Sie bisher kennen gelernt haben: Drei Ausgabefelder innerhalb des Formulars dienen zur Emulation von drei Hardwarekomponenten; Ziel des Programms ist es, die laufenden Threads, die inhaltlich nichts anderes machen als das vorherige Beispielprogramm, so auf die drei Komponenten aufzuteilen, dass sie optimal genutzt werden.

---

**BEGLEITDATEIEN:** Sie finden dieses Projekt im Verzeichnis .\VB 2005 - Entwicklerbuch\G - Smart-Client\Kap31\SimpleThread05 (MutexDemo).

---

Wenn Sie das Programm starten, sucht es sich nach dem Klick auf die Schaltfläche *Thread starten* die erste freie Hardware-Ressource (also ein Ausgabetextfeld). Ein weiterer Klick auf diese Schaltfläche startet einen weiteren Thread, der das nächste freie Ausgabefenster verwendet. Wenn Sie den vierten Thread starten, während alle vorherigen Threads noch laufen, wartet dieser auf das nächste freie Fenster. Sobald einer der Threads beendet ist, übernimmt er dessen Ausgabefenster. Die Arbeits-Thread-Routine ist so ausgelegt, dass eine einzelne Operation innerhalb des Threads unterschiedlich lange dauert. Nur so kann eine realistische Emulation einer simulierten Hardwarekomponente realisiert werden. Zu diesem Zweck gibt es ein im Konstruktor des Programms initialisiertes Random-Objekt, das den jeweils nächsten zufälligen Wert für die Sleep-Methode eines Arbeits-Threads generiert.

---

**HINWEIS:** Das Programm gibt die Bildschirmmeldungen nicht direkt mit einer bestimmten Anweisung aus – so wie Sie es von den bisherigen Beispielen gewohnt waren. Stattdessen verwendet es drei TextBox-Steuerelemente für die Textausgabe.

---

Bitte beachten Sie, dass die Veränderung der Eigenschaften eines Steuerelements ausschließlich aus dem Thread erfolgen darf, in dem das Steuerelement erstellt wurde. Aus diesem Grund werden Sie die Vorgehensweise zur Aktualisierung von Eigenschaften in diesem Beispiel ein wenig befremdlich finden. Ich werde im ► Abschnitt »Verwenden von Steuerelementen in Threads« ab Seite 953 auf dieses Problem genauer eingehen.

---



**Abbildung 31.6:** Drei Ausgabebereiche simulieren drei Hardware-Ressourcen. Über ein Array von *Mutex*-Objekten wird erkannt, welche als Nächstes zur Verfügung steht; der nächste anstehende Thread läuft dann im nächsten freien Fenster.

Das Codelisting:

```
'Speicher für Mutex-Objekte
Private myMutexes() As Mutex

'Speicher für TextBox-Controls
Private myTxtBoxes() As TextBox

'Zufallsgenerator für die künstlichen Wartezeiten im
'Arbeitsthread. Damit kann ein Thread unterschiedlich lange dauern.
Private myRandom As Random

'Delegat für das Aufrufen von Invoke, da Controls nicht
'thread-sicher sind.
Delegate Sub AddTBTextTSAcuallyDelegate(ByVal tb As TextBox, ByVal txt As String)

Public Sub New()
    MyBase.New()

    ' Dieser Aufruf ist für den Windows Form-Designer erforderlich.
    InitializeComponent()
    'Mutexes definieren.
    myMutexes = New Mutex() {New Mutex, New Mutex, New Mutex}
    'Textbox-Array zuweisen.
```

```

myTxtBoxes = New TextBox() {txtHardware1, txtHardware2, txtHardware3}
'Zufallsgenerator initialisieren.
myRandom = New Random(DateTime.Now.Millisecond)

End Sub

'Dies ist der eigentliche Arbeits-Thread (Worker Thread), der das
'Hochzählen und die Werteausgabe übernimmt.
Private Sub UmfangreicheBerechnung()

    Dim locMutexIndex As Integer
    Dim locTextBox As TextBox

    'Hier beginnt der kritische Abschnitt.
    'Warten, bis eine TextBox "frei" wird.
    locMutexIndex = Mutex.WaitAny(myMutexes)
    'Textbox, die dem freien Mutex entspricht, finden.
    locTextBox = myTxtBoxes(locMutexIndex)
    For c As Integer = 0 To 50
        SyncLock Me
            'Text in die TextBox des Threads ausgeben.
            AddTBTextTS(locTextBox, Thread.CurrentThread.Name + ":: " + c.ToString + vbNewLine)
        End SyncLock
        'Eine zufällige Weile lang warten.
        Thread.Sleep(myRandom.Next(50, 400))
    Next
    'Hier ist der kritische Abschnitt wieder vorbei.
    'Verwendete TextBox (Mutex) wieder freigeben.
    myMutexes(locMutexIndex).ReleaseMutex()
End Sub

```

Ein paar zusätzliche Erklärungen zu diesem Programm: Ein Mutex-Array wird als Klassen-Member zur Verwaltung der zur Verfügung stehenden Ressourcen im Konstruktor des Programms erstellt. Mit der statischen Funktion WaitAny, der ein Mutex-Array übergeben wird, wartet ein Thread solange, bis eines der Mutex-Objekte im Array frei wird. Wird es frei, liefert WaitAny den Index auf das jetzt freie Mutex-Objekt zurück, ändert dessen Zustand aber gleichzeitig in »blockiert«. Der zurück gelieferte Index wird anschließend verwendet, um die korrelierende TextBox zu finden, die diesem Mutex (nur über den Indexwert) quasi zugeordnet ist. Der Thread verwendet anschließend diese TextBox (wird über locTextBox referenziert), um die Ausgabe durchzuführen. WaitAny wartet übrigens nicht nur auf einen freien Mutex, sondern blockiert ihn auch wieder für den Thread, der ihn angefordert hat. Der Mutex bleibt solange im »Blockiert-Zustand«, bis der Thread den Mutex mit der ReleaseMutex-Methode wieder freigibt.

## Weitere Synchronisationsmechanismen

Neben den bereits beschriebenen Synchronisationsmechanismen kennt das .NET-Framework weitere Techniken, die in den folgenden Abschnitten kurz angerissen werden sollen:

## Synchronization- und MethodImpl-Attribut

Das Synchronization-Attribut erlaubt die Erklärung einer ganzen Klasse zum kritischen Abschnitt. Wenn Sie eine Klasse mit diesem Attribut versehen, kann nur ein einziger Thread zur gleichen Zeit auf die Klasseninstanz zugreifen. Die Anwendung dieses Attributes funktioniert prinzipiell folgendermaßen:

```
<System.Runtime.Remoting.Contexts.Synchronization()>_
Public Class KomplettSynchronisiert
    Sub MacheIrgendwas()
        'Hier die Anweisungen
        'des Arbeitsthreads.
    End Sub
    Sub MacheIrgendwasAnderes()
        'Hier die Anweisungen
        'des Arbeitsthreads.
    End Sub
End Class
```

Wenn die Synchronisation einer ganzen Klasse zu viel des Guten wäre, steht Ihnen mit MethodImpl und dessen Parameter MethodImplOptions.Synchronized ein weiteres Attribut zur Verfügung, das nur eine einzelne Prozedur einer Klasse zum kritischen Abschnitt erklärt. Seine Anwendung funktioniert wie folgt:

```
Public Class TeilweiseSynchronisiert
    <System.Runtime.CompilerServices.MethodImpl(Runtime.CompilerServices.MethodImplOptions.Synchronized)> _
    Sub MacheIrgendwasSynchronisiertes()
        'Hier die Anweisungen
        'des Arbeitsthreads.
    End Sub

    Sub MacheIrgendwasAnderes()
        'Hier die Anweisungen
        'des Arbeitsthreads.
    End Sub
End Class
```

## Die Interlocked-Klasse

Mit Hilfe der Interlocked-Klasse können Sie steuern, wie viele Threads einen kritischen Abschnitt betreten dürfen, bevor weitere Threads ausgesperrt werden. Prinzipiell funktioniert sie also wie die Monitor-Klasse und dessen Methode Enter, nur dass sie einen Parameter (eine Member-Variable der Klasse) angeben können, über die gesteuert wird, wie viele Threads den kritischen Abschnitt betreten dürfen.

```
Public Class InterlockedTest

    Dim myThreadZählerFürInterlocked As Integer

    Sub Arbeitsthread()
```

```

'Maximal 3 Threads kommen hier gleichzeitig hinein,
'dann ist Schluss!
If Interlocked.Increment(myThreadZählerFürInterlocked) <= 3 Then
    'Hier die Anweisungen
    'den Arbeitsthreads.
    Interlocked.Decrement(myThreadZählerFürInterlocked)
End If

End Sub
End Class

```

Die Klasse selbst bietet übrigens ausschließlich statische Funktionen an; sie muss also nicht instanziert werden (tatsächlich ist sie eine abstrakte Klasse (`Noninheritable`), und kann auch gar nicht instanziert werden).

## Die ReaderWriterLock-Klasse

Wenn mehrere Threads auf eine Datei zugreifen müssen, dann ist das so lange unkritisch, wie diese Threads aus der Datei lediglich lesen. Beim Schreiben sieht das anders aus: Sobald eine Datei geschrieben wird, darf kein anderer Thread zusätzlich in die Datei schreiben. Auch das Lesen aus einer Datei, in die gerade von einem anderen Thread geschrieben wird, könnte für die Ergebnisse einer Prozedur fatale Folgen haben.

Um dieses Problem zu lösen, gibt es die `ReaderWriterLock`-Klasse. Ihre wichtigsten Funktionen sind `AcquireReaderLock`, `ReleaseReaderLock`, `AcquireWriterLock` und `ReleaseWriterLock`.

Mehrere Threads, die sich über diese Klasse und deren Methoden synchronisieren wollen, müssen Zugriff auf dasselbe `ReaderWriterLock`-Objekt haben – eine Instanz dieser Klasse sollte also mindestens ein Klassen-Member sein oder als statische, öffentliche Eigenschaft programmweit verfügbar sein.<sup>5</sup>

Sowohl der Lesebereich als auch der Schreibbereich eines Threads werden nun als kritische Abschnitte definiert – etwa wie folgt:

```

Public Class ReaderWriteLockTest

    Dim myReaderWriterLock As New ReaderWriterLock

    Sub DateiLeseThread()
        'Hier kommt das Programm nur rein,
        'wenn nicht geschrieben wird, ein Schreibvorgang also
        'nicht zuvor durch ein myReaderWriterLock.AcquireWriterLock
        'eingeleitet wurde!
        '1000 Millisekunden wird auf den Lock gewartet.
        myReaderWriterLock.AcquireReaderLock(1000)
        'Lese-Thread nur ausführen, wenn Lock erteilt wurde.
        If myReaderWriterLock.IsReaderLockHeld Then

```

---

<sup>5</sup> Es könnte ja durchaus vorkommen, dass verschiedene Klassen und deren gleichzeitig laufende Arbeits-Threads einer Anwendung auf die gleiche Datei zugreifen müssen. In diesem Fall definieren Sie eine Klasse, die im statischen Konstruktor (`Shared New`) einen statischen Member vom Typ `ReaderWriterLock` instanziert und diesen über eine öffentliche statische Eigenschaft (`Public Shared Property ...`) allen anderen Klassen zur Verfügung stellt.

```

'Hier die Anweisungen
'des Arbeitsthreads
'der aus einer Datei liest.
myReaderWriterLock.ReleaseReaderLock()
Else
    'TimeOut, in die Datei konnte nicht rechtzeitig geschrieben werden.
End If
End Sub

Sub DateiSchreibThread()
    'Hier kommt ein neuer Thread nicht eher rein,
    'als bis ein anderer Lese-Thread zu Ende gelesen wurde
    'oder ein anderer Schreib-Thread den Schreibvorgang
    'komplettiert hat.
    '1000 Millisekunden wird auf den Lock gewartet.
    myReaderWriterLock.AcquireWriterLock(1000)
    'Schreib-Thread nur ausführen, wenn Lock erteilt wurde.
    If myReaderWriterLock.IsWriterLockHeld Then
        'Hier die Anweisungen
        'des Arbeitsthreads
        'der in die Datei schreibt.
        myReaderWriterLock.ReleaseWriterLock()
    Else
        'TimeOut, in die Datei konnte nicht rechtzeitig geschrieben werden.
    End If
End Sub
End Class

```

Schon die Kommentare im Beispielprogramm machen die Zusammenhänge der Operationen deutlich:

- Wenn ein Thread die Leseroutine betritt, erhält er einen Leseschutz. Dieser Leseschutz schützt diesen Thread davor, dass ein neuer Schreibvorgang beginnt, *bevor* das Lesen abgeschlossen ist (und der Thread das durch ReleaseReaderLock angezeigt hat).
- Betritt ein Thread die Schreibroutine, wartet sein AcquireWriterLock solange, bis alle ausstehenden Lesevorgänge abgearbeitet sind. Neue Lese-Anforderungen werden nun solange zurückgestellt, wie AcquireWriterLock auf seine Schutzzuteilung wartet und diesen Schutz nach Beenden des Schreibens wieder freigegeben hat.

### **Synchronisieren von voneinander abhängigen Threads mit ManualResetEvent und AutoResetEvent**

Eines vorweg: Lassen Sie sich von den Begriffen Event in den Namen dieser beiden Klassen nicht verwirren. Beide Begriffe bezeichnen instanzierbare Klassen und haben deshalb mit Ereignissen (*Events*), wie Sie sie bisher kennen gelernt haben, nicht das Geringste zu tun.

Sie verwenden beide Objekte, wenn bestimmte Threads voneinander abhängig sind. Sie haben beispielsweise einen Thread, der bestimmte Ergebnisse produziert, und weitere, die diese Ergebnisse anschließend weiterverarbeiten sollen. In diesem Fall verwenden Sie diese Objekte zur Synchronisation untereinander.

Beide Klassen unterscheiden sich lediglich in einem Punkt: Die AutoResetEvent-Klasse setzt ihren Zustand automatisch sofort zurück, sobald ein durch WaitOne geblockter Thread wieder gestartet wurde. ManualResetEvent macht eine Signaländerungsanzeige durch die Set-Eigenschaft erforderlich. Ein Beispiel für die AutoResetEvent-Klasse finden Sie übrigens im ► Abschnitt »Abbrechen und Beenden eines Threads« auf Seite 955.

## Verwenden von Steuerelementen in Threads

Wenn Sie Steuerelemente der Hauptanwendung von einem Thread aus verwenden wollen, müssen Sie folgendes beachten: Steuerelemente sind nicht thread-sicher, das heißt, sie dürfen nur aus dem Thread heraus verwendet werden, der sie erstellt hat. In der Regel ist das der Haupt-Thread der Anwendung, also der Thread, der die Benutzeroberfläche Ihrer Anwendung regelt. Aus diesem Grund nennt man diesen Thread auch den *UI-Thread* (*UI* als Abkürzung für *User Interface*, oder zu Deutsch: *Benutzeroberfläche*).

Damit ein anderer Thread dennoch das Steuerelement eines Formulars verwenden kann, muss er sich eines Tricks bedienen: Er muss dem Steuerelement mitteilen, dass es eine Prozedur aus seiner Klasse aufrufen soll. Diese Prozedur verändert dann die Eigenschaften des Steuerelements. Und da das Steuerelement diese Prozedur – wenn auch indirekt – selbst aufgerufen hat, wurde sie auch vom richtigen Thread (nämlich dem UI-Thread) aus aufgerufen.

Das Steuerelement selbst stellt dafür eine der wenigen thread-sicheren Prozeduren zur Verfügung, die es besitzt. Ihr Name: *Invoke*. Ihr wird als Parameter ein Delegat übergeben, der als Zeiger auf die eigentliche Prozedur dient, die aus dem Thread aufgerufen werden soll. Im *Mutex*-Beispiel ist diese Vorgehensweise schon zur Anwendung gekommen, und aus diesem Grund werden wir dieses Programm noch einmal unter diesem Aspekt unter die Lupe nehmen.

---

**BEGLEITDATEIEN:** Sie finden dieses Projekt im Verzeichnis .\VB 2005 - Entwicklerbuch\G - Smart-Client\Kap31\SimpleThread05 (MutexDemo).

---

Dieses Programm besteht aus drei TextBox-Steuerelementen, deren *Text*-Eigenschaft verwendet wird, um die Ausgaben des Programms anzuzeigen. Da die TextBox-Steuerelemente ausschließlich von der Thread-Prozedur verwendet werden (also in jedem Fall nicht vom UI-Thread), muss die Prozedur, die die Steuerelement-Eigenschaft setzt, zwingend über *Invoke* erfolgen.

```
'Delegat für das Aufrufen von Invoke, da Controls nicht thread-sicher sind.  
Delegate Sub AddTBTextTSActuallyDelegate(ByVal tb As TextBox, ByVal txt As String)  
  
'Dient zum Setzen einer Eigenschaft auf einer TextBox indirekt über Invoke.  
Private Sub AddTBTextTS(ByVal tb As TextBox, ByVal txt As String)  
    Dim locDel As New AddTBTextTSActuallyDelegate(AddressOf AddTBTextTSActually)  
  
    Me.Invoke(locDel, New Object() {tb, txt})  
  
End Sub
```

```

Private Sub AddTBTextTSActually(ByVal tb As TextBox, ByVal txt As String)
    tb.Text += txt
    tb.SelectionStart = tb.Text.Length - 1
    tb.ScrollToCaret()
End Sub

```

Bitte lassen Sie sich in diesem Beispiel nicht von der Tatsache irritieren, dass hier ein TextBox-Steuerelement selbst als Parameter mit übergeben wird. Diese Vorgehensweise ist für das Beispiel deshalb notwendig, da die zu verwendende TextBox vom nächsten freien Mutex-Objekt im Thread abhängig ist. Da das Steuerelement hier als Parameter selbst mit übergeben wird, ist es eigentlich egal, über welches Steuerelement im Formular die Invoke-Methode aufgerufen wird. In diesem Beispiel ist es das Formular selbst, dessen Invoke verwendet wird. Genauso gut könnte die entsprechende Zeile, die Invoke ausführt, auch folgendermaßen ausschauen:

```
tb.Invoke(locDel, New Object() {tb, txt})
```

Voraussetzung dafür ist allerdings, dass tb aus dem UI-Thread stammt.

Letzten Endes kommt es also nur darauf an, dass ein Steuerelement des UI-Threads den Aufruf des Delegaten durchführt. Um das sicherzustellen, sollten Sie, von Ausnahmen wie in diesem Fall einmal abgesehen, die Invoke-Methode des Steuerelements, auf das sich die Änderungen auch beziehen, für einen indirekten Delegatenauftrag verwenden.

## Managen von Threads

Thread-Objekte verfügen über einige Methoden, mit denen sie sich sowohl selbst steuern als auch von außen steuern lassen können. Zwei dieser Methoden haben Sie bereits kennen gelernt: die Start- und die Sleep-Methode. Die nächsten Abschnitte nehmen diese und weitere Methoden der Thread-Klasse ein wenig genauer unter die Lupe.

### Starten eines Threads mit Start

Sie starten einen Thread mit seiner Start-Methode. Voraussetzung dafür ist, dass Sie den Thread zuvor mit einem ThreadStart-Objekt instanziert haben, das bei seiner Instanzierung wiederum die Adresse der Prozedur erhalten hat, die den eigentlichen Thread darstellt.

### Vorübergehendes Aussetzen eines Threads mit Sleep – Statusänderungen im Framework bei Suspend und Resume

Wenn ein Thread für eine gewisse Zeit unterbrochen werden soll, verwenden Sie die Sleep-Methode des Thread-Objektes. Sleep übernimmt als Parameter entweder einen Integer-Wert, der die abzuwartende Zeitspanne in Millisekunden bestimmt oder einen TimeSpan-Wert, der ebenfalls die abzuwartende Dauer bestimmt, nach deren Ablauf der Thread wieder aktiv wird. Mit der Sleep-Methode kann sich ein Thread dadurch selbst »aussetzen« und wieder »aufwecken«.

Die Methoden Suspend und Resume sind im .NET 2.0-Framework übrigens als veraltet markiert, und sollten laut Microsoft nicht mehr verwendet werden. Stattdessen sollten Sie andere Synchronisierungstechniken verwenden.

Wie Sie einen Update Ihres Source-Codes vornehmen, zeigt der Vergleich der beiden Beispiele in den Verzeichnissen *ObsoleteThreadMethodsDemo* und *ThreadPoolThreads* im Verzeichnis der Beispielprojekte dieses Kapitels. Sie entsprechen beide dem Prinzip des Beispiels, das in ► Abschnitt »Verwenden des Thread-Pools« ab Seite 970 besprochen wird. Im ersten Verzeichnis finden Sie jedoch die veraltete Version, im zweiten die entsprechend angepasste.

## Abbrechen und Beenden eines Threads

`Abort` nennt sich die Methode, mit der Sie die Möglichkeit haben, einen Thread abzubrechen. Allerdings sollten Sie von dieser Funktion nur in Ausnahmefällen Gebrauch machen, denn: `Abort` nicht unmittelbar ausgeführt. Framework-intern wird eine Ausnahme vom Typ `ThreadAbortException` ausgelöst. Diese Ausnahme ist allerdings etwas Besonderes, da Sie sie nicht abfangen können. Falls der Thread jedoch in einem `Try/Catch/Finally`-Codeblock ausgeführt wird, garantiert das Framework, dass der `Finally`-Block bei `Abort` auf jeden Fall abgearbeitet wird. Ein Thread kann dabei mithilfe seiner `ThreadState`-Eigenschaft feststellen, dass er gerade abgebrochen wird, und das Abbrechen sogar mit `ResetAbort` verhindern.

Allerdings ist das Abbrechen eines Threads keine elegante Vorgehensweise – eher die Brechstangenmethode. Ein Thread sollte ausschließlich beendet werden, indem er seine Arbeitsprozedur verlässt. Auf der anderen Seite muss ein Thread auf jeden Fall dafür sorgen, dass er beendet wird, wenn das Hauptprogramm beendet wird, das ihn beherbergt.

In den vorangegangenen Beispielen ist das bislang nicht der Fall gewesen. Wenn Sie das Hauptfenster schließen, wird zwar das Hauptprogramm beendet – Threads, die zu dieser Zeit noch aktiv sind, laufen aber unbehelligt weiter. Das heißt, für das letzte Beispiel gilt »unbehelligt« nicht wirklich. Tatsächlich steigt das Programm mit einer Fehlermeldung aus, wenn Sie es schließen, während andere Threads noch munter am Werkeln sind. Das liegt daran, dass der Thread seine Ausgabe in eine `TextBox` vornimmt, die es zu diesem Zeitpunkt schon gar nicht mehr gibt.

Die erste Idee, die einem in den Sinn kommen könnte, um dieses Problem zu lösen, wäre ein klassenweites Flag, das dann gesetzt wird, wenn das Formular geschlossen wird. Dazu müsste man lediglich die `OnClosing`-Routine des Formulars überschreiben. Bevor ein Thread eine Ausgabe im Formular vornimmt, könnte er dieses Flag überprüfen. Wäre es gesetzt, beendete der Thread sich mit einem simplen `Exit Sub` auf unproblematische Art und Weise.

Allerdings ist diese Problemlösung – jedenfalls was Windows Forms-Anwendungen anbelangt – nicht konsequent zu Ende gedacht, denn: Wird dieses »Thread-Abbrechen-Flag« in dem Moment auf `True` gesetzt, in dem sich der Thread gerade zwischen der Flag-Abfrage und der Ausgabe des Textes ins Steuerelement befindet, hat die ganze Aktion überhaupt nichts gebracht. Auch in diesem Fall rauscht der Thread ungebremst in die Ausgaberroutine für die `TextBox`, die es auch in diesem Fall nicht mehr gibt. Abermals ist eine Ausnahme die Folge. Was also tun?

Eine Synchronisierungstechnik ist an dieser Stelle wieder indiziert. `OnClosing` muss so lange warten, bis ein Thread den kritischen Bereich des Schreibens in die `TextBox` beendet hat. Dazu muss der Thread diesen kritischen Bereich aber erst einmal definieren und dafür bietet sich in diesem Fall das `AutoRaiseEvent`-Objekt an (das `ManualRaiseEvent`-Objekt täte es übrigens genau so gut für unser Vorhaben).

---

**BEGLEITDATEIEN:** Sie finden dieses Projekt im Verzeichnis .\VB 2005 - Entwicklerbuch\G - Smart-Client\Kap31\SimpleThread06 (MutexDemo proper)\.

---

Wenn Sie dieses Beispiel starten, können Sie das Formular schließen, egal wie viele Threads noch unterwegs sind. Der veränderte Code für das Beispielprogramm sieht folgendermaßen aus:

```
'Member-Variable, damit die Threads durchnummeriert werden können.  
'Dient nur zur späteren Unterscheidung des laufenden Threads, wenn  
'er Ergebnisse im Ausgabefenster darstellt.  
Private myArbeitsThreadNr As Integer = 1  
  
'Speicher für Mutex-Objekte  
Private myMutexes() As Mutex  
  
'Speicher für TextBox-Controls  
Private myTxtBoxes() As TextBox  
  
'Zufallsgenerator für die künstlichen Wartezeiten im  
'Arbeitsthread. Damit kann ein Thread unterschiedlich lange dauern.  
Private myRandom As Random  
  
'Delegat für das Aufrufen von Invoke, da Controls nicht  
'Thread-Sicher sind.  
Delegate Sub AddTBTTextTSAcuallyDelegate(ByVal tb As TextBox, ByVal txt As String)  
  
'Flag, dass bestimmt, wann alle Threads zu gehen haben  
Private myAbortAllThreads As Boolean  
  
'Zusätzliches Synchronisierungsobjekt für "Alle Threads beenden!"  
Private myAutoResetEvent As AutoResetEvent  
  
Public Sub New()  
    MyBase.New()  
  
    ' Dieser Aufruf ist für den Windows Form-Designer erforderlich.  
    InitializeComponent()  
    'Mutexes definieren  
    myMutexes = New Mutex() {New Mutex, New Mutex, New Mutex}  
    'Textbox-Array zuweisen  
    myTxtBoxes = New TextBox() {txtHardware1, txtHardware2, txtHardware3}  
    'Zufallsgenerator initialisieren  
    myRandom = New Random(DateTime.Now.Millisecond)  
    'Zusätzliches Synchronisierungsobjekt für "Alle Threads beenden!" initialisieren  
    myAutoResetEvent = New AutoResetEvent(True)  
  
End Sub  
  
'Dies ist der eigentliche Arbeits-Thread (Worker Thread), der das  
'Hochzählen und die Werteausgabe übernimmt  
Private Sub UmfangreicheBerechnung()  
    Dim locMutexIndex As Integer  
    Dim locTextBox As TextBox
```

```

'Hier beginnt der 1. kritische Abschnitt der Mutexes
'Warten, bis eine TextBox "frei" wird
locMutexIndex = Mutex.WaitAny(myMutexes)
'Thread beenden, wenn das "Alles-Threads-Beenden-Flag" während
'des Wartens signalisiert wurde
If myAbortAllThreads Then
    'Verwendeten Textbox-Mutex wieder freigeben,
    'sonst knallt es, wenn die anderen Threads in diesem
    'Abschnitt beendet werden sollen!
    myMutexes(locMutexIndex).ReleaseMutex()
    Exit Sub
End If
'Textbox, die dem freien Mutex entspricht finden
locTextBox = myTxtBoxes(locMutexIndex)
'Hier beginnt der 2. kritische Abschnitt für OnClosing
myAutoResetEvent.Reset()
For c As Integer = 0 To 20
    SyncLock Me
        'Falls abgebrochen werden soll,
        If myAbortAllThreads Then
            'OnClosing benachrichtigen
            myAutoResetEvent.Set()
            'Verwendeten Textbox-Mutex wieder freigeben.
            'Grund: Siehe oben.
            myMutexes(locMutexIndex).ReleaseMutex()
            Exit Sub
        End If
        'Text in die TextBox des Threads ausgeben
        AddTBTTextS(locTextBox, Thread.CurrentThread.Name + ":: " + c.ToString + vbNewLine)
        Console.WriteLine(Thread.CurrentThread.Name + ":: " + c.ToString + vbNewLine)
    End SyncLock
    'Eine zufällige Weile lang warten
    Thread.Sleep(myRandom.Next(50, 400))
Next
'2. kritische Abschnitt beendet (OnClosing)
myAutoResetEvent.Set()
'Hier ist der 1. kritische Abschnitt wieder vorbei.
'Verwendete TextBox (Mutex) wieder freigeben
myMutexes(locMutexIndex).ReleaseMutex()
End Sub

Protected Overrides Sub OnClosing(ByVal e As System.ComponentModel.CancelEventArgs)
    'WICHTIG: Ein einfaches Warten mit WaitOne reicht in diesem
    'Fall nicht aus, da es sich um den Hauptthread handelt.
    'Dann aber würde WaitOne die Nachrichtenwarteschlange blockieren,
    'und ohne die funktioniert Invoke nicht. Deswegen wird hier ein TimeOut
    'angegeben, der Nachrichtenwarteschlange die Möglichkeit gegeben, einmal
    '"Luft zu holen", und dann weiter gewartet.
    myAbortAllThreads = True
    'DoEvents beim Warten in ein-Millisekunden-Abständen auslösen

```

```

Do While Not myAutoResetEvent.WaitOne(1, True)
    Application.DoEvents()
Loop
End Sub

'Dient zum Setzen einer Eigenschaft auf einer TextBox indirekt über Invoke
Private Sub AddTBTextTS(ByVal tb As TextBox, ByVal txt As String)
    Dim locDel As New AddTBTextTSAcuallyDelegate(AddressOf AddTBTextTSAcually)

    Me.Invoke(locDel, New Object() {tb, txt})

End Sub

Private Sub AddTBTextTSAcually(ByVal tb As TextBox, ByVal txt As String)
    tb.Text += txt
    tb.SelectionStart = tb.Text.Length - 1
    tb.ScrollToCaret()
End Sub

Private Sub btnThreadStarten_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnThreadStarten.Click
    'Dieses Objekt kapselt den eigentlichen Thread
    Dim locThread As Thread
    'Dieses Objekt benötigen Sie, um die Prozedur zu bestimmen,
    'die den Thread ausführt.
    Dim locThreadStart As ThreadStart

    'Thredausführende Prozedur bestimmen
    locThreadStart = New ThreadStart(AddressOf UmfangreicheBerechnung)
    'ThreadStart-Objekt dem Thread-Objekt übergeben
    locThread = New Thread(locThreadStart)
    'Thread-Namen bestimmen
    locThread.Name = "Arbeits-Thread: " + myArbeitsThreadNr.ToString
    'Thread starten
    locThread.Start()
    'Zähler, damit die Threads durch ihren Namen unterschieden werden können
    myArbeitsThreadNr += 1

End Sub

Private Sub btnBeenden_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnBeenden.Click
    'Einfach so geht's normalerweise nicht
    Me.Close()
End Sub

```

Eine kleine Anmerkung vielleicht noch an dieser Stelle zur überschriebenen `OnClosing`-Prozedur: Wie aus den Kommentaren im Codelisting schon ersichtlich, gilt es noch ein letztes Problem aus der Welt zu schaffen. Ein einfaches `WaitOne` würde den UI-Thread einfrieren. Dummerweise benötigt `Invoke` eines Steuerelements eine aktive Nachrichtenwarteschlange, um zu funktionieren. Aus diesem Grund müssen wir an dieser Stelle wieder einen Trick anwenden, damit sich das Programm beim Warten auf die letzte Textausgabe nicht aufhängt. Als Argument übergeben wir einen `Timeout`-Wert

von einer Millisekunde, nachdem `WaitOne` zunächst den Wartevorgang auf das Abschließen des kritischen Abschnittes eines Threads unterbricht. Ein anschließendes `DoEvents` erlaubt das Ausführen eines möglicherweise noch ausstehenden `Invoke` im Arbeits-Thread. `DoEvents` wird in einer Schleife abgehandelt, die so lange ausgeführt wird, bis `WaitOne` zurückmeldet, dass es nicht durch `TimeOut`, sondern tatsächlich durch die Signalisierung des Arbeits-Threads beendet wurde. In diesem Fall »weiß« `OnClosing`, dass der ausstehende Thread beendet wurde, die `TextBox` zur Ausgabe ergo nicht mehr gebraucht wird und das Formular zu diesem Zeitpunkt sicher geschlossen werden kann.

---

**HINWEIS:** Da die Verarbeitung dieser Warteschleife natürlich auch Zeit kostet, sollten Sie bei sehr zeitkritischen Operationen von dieser Methode absehen. Die Prozessorleistung, die für die Warteschleife benötigt wird, fehlt anderen Threads natürlich. Eine Zwischenlösung: Sie können auf Kosten der Reaktionszeit natürlich den `TimeOut`-Parameter für `WaitOne` erhöhen, um Leistung zu sparen. Denn während `WaitOne` »ausgeführt« wird, können andere Threads bedient werden.

---

## Datenaustausch zwischen Threads durch Kapseln von Threads in Klassen

Sie haben im Laufe der letzten Kapitel bereits feststellen können, dass Sie beim Einrichten eines Threads keine Parameter an die eigentliche Thread-Prozedur übergeben können. In der Praxis sind Threads, denen keine Daten zum Verarbeiten übergeben werden können, aber eher nutzlos. Doch so groß ist das Problem nicht, denn es gibt einen ganz einfachen Trick, mit dem Sie es meistern können: Sie kapseln die eigentliche Thread-Routine einfach in einer Klasse. Die notwendigen Parameter, die die Thread-Routine benötigt, können Sie dann einfach im Konstruktor der Klasse übergeben.

Da der Thread asynchron läuft, als nur sozusagen »angeschmissen« wird und dann alleine weiterarbeitet, muss die Thread-Klasse natürlich auch in der Lage sein, ein »ich habe fertig« an die ihn einbindende Instanz mitteilen zu können. Doch auch dieses Problem ist vergleichsweise einfach in den Griff zu bekommen: Wenn der Arbeits-Thread innerhalb der Klasse seine Aufgabe abgeschlossen hat, löst er einfach ein Ereignis aus. Die Ergebnisse, die durch ihn zustande gekommen sind, können anschließend durch Eigenschaften der Öffentlichkeit zugänglich gemacht werden.

Das folgende Codelisting zeigt, wie beispielsweise eine Klasse aussehen kann, die ein beliebiges Array in einem Thread sortieren kann.

```
Public Class SortArrayThreaded
    Implements IDisposable

    Public Event SortCompleted(ByVal sender As Object, ByVal e As SortCompletedEventArgs)
    Private myArrayToSort As ArrayList
    Private myAutoResetEvent As AutoResetEvent
    Private myTerminateThread As Boolean
    Private myThreadStart As ThreadStart
    Private myThread As Thread
    Private mySortingComplete As Boolean

    'Konstruktor übernimmt die zu sortierende Liste. Der ThreadName hat nur
    'dokumentarischen Charakter.
    Sub New(ByVal ArrayToSort As ArrayList, ByVal ThreadName As String)
```

```

myArrayToSort = ArrayToSort
myThread = New Thread(New ThreadStart(AddressOf SortArray))
myThread.Name = ThreadName
myAutoResetEvent = New AutoResetEvent(True)
End Sub

```

Der eigentliche Thread kann beginnen, wenn die die Klasse einbindende Instanz nach der Instanzierung die folgende Methode aufruft. Damit der Thread mit anderen Aufgaben synchronisiert werden kann, stellt er ein AutoResetEvent-Objekt zur Verfügung, dessen Signalisierung an dieser Stelle zurückgesetzt wird. Ein anderer Thread, der darauf wartet, dass die Sortierung beendet wird, kann nun nicht nur durch das Ereignis, sondern auch über die WaitOne-Methode (oder, wenn mehrere Threads zu synchronisieren sind, auch über WaitAny oder WaitAll) über das Ende der Sortierung informiert werden.

```

'Thread starten und signalisieren, dass blockiert.
Public Sub StartSortingAsynchron()
    myAutoResetEvent.Reset()
    myThread.Start()
End Sub

Private Sub SortArray()

    Dim locSortCompletionStatus As SortCompletionStatus

    If Not myArrayToSort Is Nothing Then
        locSortCompletionStatus = Me.ShellSort()
    End If
    myAutoResetEvent.Set()
    'Dieses Ereignis könnte man einbinden, falls der Sort-Thread über das
    'Sortierende benachrichten soll.
    RaiseEvent SortCompleted(Me, New SortCompletedEventArgs(locSortCompletionStatus))
End Sub

```

Kleine Anmerkung am Rande: Die folgende Prozedur stellt den eigentlichen Arbeits-Thread der Klasse dar. Natürlich gibt es eine Reihe von Sortiermöglichkeiten von Elementen, die im Framework eingebaut sind. Aber behalten Sie im Hinterkopf, dass es hier in erster Linie um die Demonstration der grundsätzlichen Threading-Möglichkeiten geht.

```

'Sortiert eine ArrayList, die IComparable-Member enthält.
'Null-Werte oder inkompatible Typen werden nicht geprüft!
Private Function ShellSort() As SortCompletionStatus

    Dim locOutCount, locInCount As Integer
    Dim locDelta As Integer
    Dim locElement As IComparable

    locDelta = 1

    'Größten Wert der Distanzfolge ermitteln.
    Do
        locDelta = 3 * locDelta + 1
    Loop Until locDelta > myArrayToSort.Count

```

```

Do
    'War eins zu groß, also wieder teilen.
    locDelta \= 3

    'Shellsorts Kernalgorithmus
    For locOutCount = locDelta To myArrayToSort.Count - 1
        locElement = CType(myArrayToSort(locOutCount), IComparable)
        locInCount = locOutCount
        Do While CType(myArrayToSort(locInCount - locDelta), IComparable).CompareTo(locElement) = 1
            If myTerminateThread Then
                Return SortCompletionStatus.Aborted
            End If
            myArrayToSort(locInCount) = myArrayToSort(locInCount - locDelta)
            locInCount = locInCount - locDelta
            If (locInCount <= locDelta) Then Exit Do
        Loop
        myArrayToSort(locInCount) = locElement
    Next
Loop Until locDelta = 0
Return SortCompletionStatus.Completed

End Function

```

---

**WICHTIG:** Damit ein laufender Thread auf elegante Art und Weise beendet werden kann (und nicht mit brachialer Gewalt durch Abort), gibt es eine Member-Variable namens `myTerminateThread`, deren Setzen auf True bewirkt, dass der Sortalgorithmus seine Arbeit abbricht, die Prozedur verlässt und damit den Arbeits-Thread sauber zu Ende führt. Um ein Programm nicht durch laufende Sortier-Threads zu blocken, wenn es für die Klasseninstanz Zeit wird, vom Garbage Collector entsorgt zu werden, implementiert die Klasse `IDisposable` und damit `Dispose`, das diesen Member setzt. Der Garbage Collector ist damit spätestens derjenige, der den Thread beendet. Eigentlich ist das aber nur das Netz unter dem Trapez. Sie sollten in jedem Fall selbst durch geschickte Programmierung darauf achten, dass kein noch laufender Thread existiert, wenn Sie das umgebende Programm beenden. Rufen Sie im Zweifelsfall lieber selbst die `Dispose`-Methode Ihrer Thread-Klasse auf, damit der Thread in jedem Fall ordnungsgemäß beendet wird.

```

'Dispose beendet einen vielleicht noch laufenden Sortier-Thread.
Public Sub Dispose() Implements System.IDisposable.Dispose
    myTerminateThread = True
End Sub

'Stellt die benötigten Eigenschaften zur Verfügung.
Public ReadOnly Property AutoResetEvent() As AutoResetEvent
    Get
        Return myAutoResetEvent
    End Get
End Property

Public ReadOnly Property ArrayList() As ArrayList
    Get
        Return myArrayList
    End Get
End Property
End Class

```

```
'Die beiden möglichen Ergebnisse, wenn der Sortier-Thread beendet wurde.  
'Falls er durch Dispose abgebrochen wurde, liefert er Aborted zurück.  
Public Enum SortCompletionStatus  
    Completed  
    Aborted  
End Enum
```

Damit das Ereignis, das ausgelöst wird, wenn der Thread die Sortierung beendet hat, auch ordnungsgemäß über die durchgeföhrten Vorgänge informieren kann, gibt es zusätzlich zur eigentlichen Klasse noch ein paar Hilfselemente für diesen Zweck:

```
'Dient nur der Ereignisübergabe, wenn Sie das SortCompleted-Ereignis  
'nach Abbruch oder Abschluss des Sortierens auslösen wollen.  
Public Class SortCompletedEventArgs  
    Inherits EventArgs  
  
    Private mySortCompletionStatus As SortCompletionStatus  
    Sub New(ByVal scs As SortCompletionStatus)  
        mySortCompletionStatus = scs  
    End Sub  
  
    Public Property SortCompletionStatus() As SortCompletionStatus  
        Get  
            Return mySortCompletionStatus  
        End Get  
        Set(ByVal Value As SortCompletionStatus)  
            mySortCompletionStatus = Value  
        End Set  
    End Property  
  
End Class
```

Soviel zum eigentlichen Arbeitstier dieses Beispiels. Nun ist es an der Zeit herauszufinden, ob es auch wirklich das leistet, was wir von ihm erwarten.

## Der Einsatz von Thread-Klassen in der Praxis

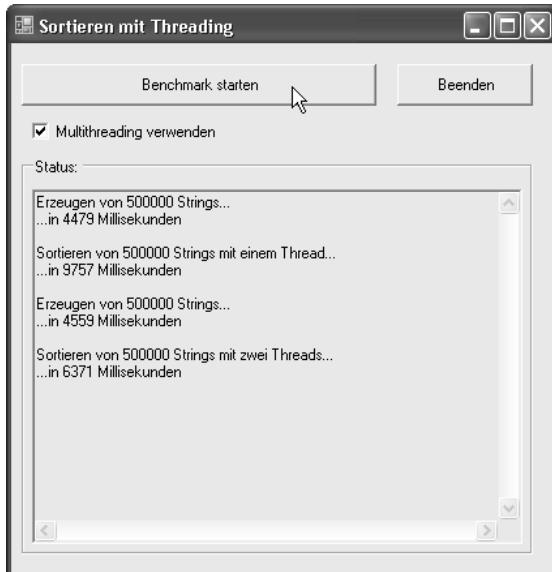
Sie finden in den Begleitdateien ein Beispiel, dass die im vorherigen Abschnitt vorgestellte Klasse demonstriert. Und es macht nicht nur das. Es zeigt auch die Geschwindigkeitsverhältnisse von Programmen, wenn Sie mit einem oder mehreren Threads bestimmte Probleme lösen.

---

**BEGLEITDATEIEN:** Sie finden dieses Projekt im Verzeichnis .\VB 2005 - Entwicklerbuch\G - Smart-Client\Kap31\ThreadsInPraxis (MultiThreadingSorting)\.

---

Wenn Sie das Programm starten, zeigt es Ihnen einen Dialog, wie Sie ihn in etwa auch in Abbildung 31.7 sehen können. Bevor Sie den Test starten, entscheiden Sie sich durch Anklicken des entsprechenden Feldes, ob Sie für die Sortierung der Daten Single- oder Multithreading verwenden wollen. Mit einem weiteren Klick auf *Benchmark* starten Sie die Threading-Operationen.



**Abbildung 31.7:** Solche Unterschiede erreichen Sie nur auf einem Multiprozessor- oder MultiCore-System. Wichtig: Beim Einsatz von Hyperthreading auf einem Prozessor kann das Sortieren unter Umständen sogar länger dauern, als beim ausgeschalteten Hyperthreading.

Das folgende Codelisting zeigt, wie das Programm funktioniert. Längere Kommentare im Listing finden Sie im Folgenden der leichteren Lesbarkeit wegen in Normalschrift gesetzt und dort, wo es sinnvoll erscheint, etwas ausführlicher beschrieben. Sie sind im Listing aber ebenfalls vorhanden.

Einige Worte zur generellen Funktionsweise vorweg: Das Programm arbeitet mit maximal vier Threads, aber mindestens drei Threads, wenn die Sortierung vorgenommen wird. Der UI-Thread – also der Thread, der das Formular verwaltet – startet automatisch mit dem Programm. Wenn der Anwender den Benchmark startet, beginnt ein zweiter Thread, der die zu sortierenden Elemente erzeugt und dann entweder einen weiteren Thread oder zwei weitere Threads startet, um diese Elemente zu sortieren. Mit dieser Vorgehensweise wird erreicht, dass der UI-Thread ungestört weiter operieren kann, ohne durch den eigentlichen Arbeits-Thread angehalten zu werden.

```
'Dieser Namensbereich enthält die benötigten Threading-Objekte.
Imports System.Threading
Imports System.IO

Public Class frmMain
    Inherits System.Windows.Forms.Form

    #Region " Vom Windows Form Designer generierter Code "
        'Aus Platzgründen ausgelassen
    #End Region

    Private Const cAmountOfElements As Integer = 75000
    Private Const cCharsPerString As Integer = 100
    Private Const cShowResults As Boolean = False
```

Die folgenden Delegaten werden benötigt, um bestimmte Eigenschaften der Steuerelemente des Formulars thread-sicher verändern zu können. Das gilt für das Setzen der Text-Eigenschaft des TextBox-Steuerelements, das für die Darstellung der Kommentare zuständig ist, auf der einen und für das Wiedereinschalten der Schaltflächen auf der anderen Seite.

```

'Delegaten für das Aufrufen von Invoke, da Controls nicht
'thread-sicher sind.
'Für die Textbox zur Ausgabe
Private Delegate Sub AddTBTextTSActuallyDelegate(ByVal txt As String)
'Für das Einschalten der Buttons, wenn der Vorgang beendet ist.
Private Delegate Sub TSEnableControlsDelegate()

'Flag, das bestimmt, wann alle Threads zu gehen haben.
Private myAbortAllThreads As Boolean

'Ausgabe-Textbox. Statisch deswegen, damit jeder Thread
'zu jeder Zeit darauf zugreifen kann.
Private Shared myAusgabeTextBox As TextBox

'Synchronisation für die Ausgabe-Textbox
Private Shared myAutoResetEvent As AutoResetEvent

'Merkt sich thread-safe, ob die Sortierung mit einem oder
'zwei Threads durchgeführt werden soll.
Private myUseTwoThread As Boolean

'Ereignis, um dem UI-Thread mitteilen zu können,
'dass der Hauptarbeits-Thread abgeschlossen ist.
Private Event MainThreadFinished()

Public Sub New()
    MyBase.New()

    ' Dieser Aufruf ist für den Windows Form-Designer erforderlich.
    InitializeComponent()

    'Synchronisation für die Ausgabe-Textbox
    myAutoResetEvent = New AutoResetEvent(True)

End Sub

```

Wichtig bei der Verwendung der folgenden Funktion ist die Synchronisation mit allen Programmprozeduren, die sie verwenden, denn: Sollte das Formular genau in dem Moment geschlossen werden, in dem ein Thread genau diese Routine erreicht hat, dann würde Invoke auf ein Steuerelement zugreifen, das zu dieser Zeit nicht mehr existierte. Eine Ausnahme wäre die Folge. Aus diesem Grund wird das Schließen des Formulars mit der Ausgabe in der TextBox über ein AutoResetEvent-Objekt synchronisiert. OnClosing fängt das Schließen-Ereignis ab und wartet, bis eine mögliche, gerade laufende Ausgabe in der TextBox abgeschlossen ist.

```

'Dient zum Setzen einer Eigenschaft auf der TextBox indirekt über Invoke.
Public Shared Sub AddTBText(ByVal txt As String)
    'Formular nicht mehr oder noch nicht da...
    If AusgabeTextBox Is Nothing Then
        '...und tschö!
        Exit Sub
    End If
    'Threads synchronisieren: Hier beginnt kritischer Abschnitt.
    myAutoResetEvent.Reset()

```

```

Dim locDel As New AddTBTextTSActuallyDelegate(AddressOf AddTBTextTSActually)
AusgabeTextBox.Invoke(locDel, New Object() {txt})
myAutoResetEvent.Set()
End Sub

'Diese Routine wird indirekt über Invoke aufgerufen, und ist damit UI-Thread.
Private Shared Sub AddTBTextTSActually(ByVal txt As String)
    AusgabeTextBox.Text += txt
    AusgabeTextBox.SelectionStart = AusgabeTextBox.Text.Length - 1
    AusgabeTextBox.ScrollToCaret()
End Sub

'Liefert die TextBox als Eigenschaft.
Private Shared ReadOnly Property AusgabeTextBox() As TextBox
    Get
        Return myAusgabeTextBox
    End Get
End Property

```

Die folgende Routine kümmert sich um die notwendigen Schritte, wenn der Anwender die Schaltfläche *Benchmark starten* angeklickt hat. Sie startet dann den Arbeits-Thread (den Haupt-Thread), der zunächst alle zu sortierenden Elemente erzeugt und sie dann anschließend durch die Sortier-Threads ordnen lässt.

```

Private Sub btnThreadStarten_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnBenchmarkStarten.Click
    'Dieses Objekt kapselt den eigentlichen Thread.
    Dim locThread As Thread
    'Dieses Objekt benötigen Sie, um die Prozedur zu bestimmen,
    'die den Hauptthread ausführt.
    Dim locThreadStart As ThreadStart

    'Ausgabetextbox bestimmen.
    myAusgabeTextBox = Me.txtAusgabe

```

Damit der Haupt-Thread kein zweites Mal gestartet werden kann, schaltet die Routine die Schaltflächen an dieser Stelle aus. Wenn der komplette Testparcours abgeschlossen ist, löst der Haupt-Thread ein Ereignis aus, das vom Formular eingebunden wird. Die Behandlungsprozedur dieses Ereignisses schaltet die Schaltflächen dann indirekt über *Invoke* wieder ein. Diese Vorgehensweise dient nur der Demonstration: Genauso gut könnte der Haupt-Thread selbst die Schaltflächen wieder einschalten, wenn alle Operationen abgeschlossen sind.

```

btnBenchmarkStarten.Enabled = False
btnBeenden.Enabled = False

'Festhalten, ob die Sortierung in einem oder zwei Threads erfolgen soll.
myUseTwoThread = Me.chkMultithreading.Checked

'Thread ausführende Prozedur bestimmen.
locThreadStart = New ThreadStart(AddressOf StartMainThread)
'ThreadStart-Objekt dem Thread-Objekt übergeben
locThread = New Thread(locThreadStart)
'Thread-Namen bestimmen.
locThread.Name = "MainThread"

```

```

'Haupt-Thread starten.
locThread.Start()

'Der UI-Thread (die Nachrichtenwarteschlange) läuft jetzt leer nebenbei.
'Auf diese Weise kann der komplette Testparcours ruhig 100% Prozessorleistung
'verschlingen; da er in einem Extra-Thread läuft, bleibt das Programm
'dennoch bedienbar.

End Sub

```

Dies ist die eigentliche Haupt-Thread-Routine (aber nicht der UI-Thread!), die das Array mit den zu sortierenden Elementen generiert und dann selbst wiederum entweder ein oder zwei Arbeits-Threads aufruft, die die eigentliche Sortierung durchführen.

```

Private Sub StartMainThread()

    Dim locStrings(cAmountOfElements - 1) As String
    Dim locGauge As New HighSpeedTimeGauge
    Dim locAutoResetEvents(1) As AutoResetEvent

    Dim locRandom As New Random(DateTime.Now.Millisecond)
    Dim locChars(cCharsPerString) As Char

    'Damit die Controls nach Abschluss des Sortierens auf dem Formular
    'wieder eingeschaltet werden können, muss der Haupt-Thread das Ende
    'der Sortierung mitbekommen. Deswegen: Ereignis
    AddHandler MainThreadFinished, AddressOf MainThreadFinishedHandler

    'String-Array erzeugen; hatten wir schon zig Mal...
    AddTBText("Erzeugen von " + cAmountOfElements.ToString + " Strings..." + vbNewLine)
    locGauge.Start()
    For locOutCount As Integer = 0 To cAmountOfElements - 1
        For locInCount As Integer = 0 To cCharsPerString - 1
            If myAbortAllThreads Then
                RemoveHandler MainThreadFinished, AddressOf MainThreadFinishedHandler
                Exit Sub
            End If
            Dim locIntTemp As Integer = Convert.ToInt32(locRandom.NextDouble * 52)
            If locIntTemp > 26 Then
                locIntTemp += 97 - 26
            Else
                locIntTemp += 65
            End If
            locChars(locInCount) = Convert.ToChar(locIntTemp)
        Next
        locStrings(locOutCount) = New String(locChars)
    Next
    locGauge.Stop()
    AddTBText("...in " + locGauge.DurationInMilliSeconds.ToString + " Millisekunden" + vbNewLine)
    AddTBText(vbNewLine)
    locGauge.Reset()

```

Die Member-Variable `myUseTwoThread` wird durch den Ereignis-Handler des CheckBox-Steuerelements gesetzt. Sie bestimmt, ob der Haupt-Thread das Sortieren mit zwei oder nur einem Thread starten soll.

```

If Not myUseTwoThread Then
    'Messung starten - hier wird mit nur einem Thread sortiert.
    locGauge.Start()
    Dim locFirstSortThread As New SortArrayThreaded(New ArrayList(locStrings), "1. SortThread")

    AddTBText("Sortieren von " + cAmountOfElements.ToString() + " Strings mit einem Thread..." + _
              vbCrLf)
    locAutoResetEvents(0) = locFirstSortThread.AutoResetEvent
    locFirstSortThread.StartSortingAsynchron()

```

Die folgende Do-While-Schleife wartet, bis der Sortier-Thread abgeschlossen ist. Der Sortier-Thread, der durch die SortArrayThreaded-Klasse gekapselt wird, stellt eine AutoResetEvents-Eigenschaft bereit, die signalisiert wird, wenn der Sortierungsvorgang abgeschlossen ist.

---

**TIPP:** Wenn Sie eigene Klassen implementieren, die Aufgaben kapseln, welche in Threads ausgeführt werden, sollten Sie ebenfalls dafür sorgen, dass der Abschluss einer Aufgabe nicht nur durch ein Ereignis, sondern auch mit einem Synchronisierungs-Objekt wie beispielsweise der Mutex- oder der AutoResetEvent-Klasse signalisiert wird.

---

```

Do While Not locAutoResetEvents(0).WaitOne(1, True)
    'Mit Timeout-Wert warten, damit ein Eingreifen (Abbrechen) im Sortier-Thread möglich
    'bleibt, wenn das komplette Programm beendet werden soll.
    If myAbortAllThreads = True Then
        'Die Dispose-Methode der SortArrayThread-Klasse bricht einen vielleicht laufenden
        'Sortiert-Thread ab.
        locFirstSortThread.Dispose()
        RemoveHandler MainThreadFinished, AddressOf MainThreadFinishedHandler
        Exit Sub
    End If
Loop
'Messung beenden.
locGauge.Stop()
AddTBText("...in " + locGauge.DurationInMilliseconds.ToString() + " Millisekunden" + vbCrLf)
AddTBText(vbCrLf)
'ArrayList zurück-casten.
locStrings = CType(locFirstSortThread.ArrayList.ToArray(GetType(String)), String())
Else
    'Es soll mit zwei Threads sortiert werden.
    locGauge.Start()
    'Zwei ArrayLists erstellen, die jeweils den oberen und den unteren Teil der
    'zu sortierenden Gesamtliste beinhalten.
    Dim locFirstAL As New ArrayList(locStrings.Length \ 2 + 1)
    Dim locSecondAL As New ArrayList(locStrings.Length \ 2 + 1)
    Dim locMitte As Integer = locStrings.Length \ 2

    'Elemente über beide ArrayLists verteilen
    For c As Integer = 0 To locMitte - 1
        locFirstAL.Add(locStrings(c))
        locSecondAL.Add(locStrings(c + locMitte))
    Next

```

```

'Zwei SortArrayThreaded-Instanzen erzeugen und ihnen die Teillisten übergeben.
Dim locFirstSortThread As New SortArrayThreaded(locFirstAL, "1. SortThread")
Dim locSecondSortThread As New SortArrayThreaded(locSecondAL, "2. SortThread")

AddTBText("Sortieren von " + cAmountOfElements.ToString + " Strings mit zwei Threads..." + _
vbNewLine)
'Beide AutoResetEvent-Objekte der beiden Sortier-Threads in ein Array packen,
'damit auf den Abschluss beider Threads gewartet werden kann.
locAutoResetEvents(0) = locFirstSortThread.AutoResetEvent
locAutoResetEvents(1) = locSecondSortThread.AutoResetEvent
'Los geht's!
locFirstSortThread.StartSortingAsynchron()
locSecondSortThread.StartSortingAsynchron()
'Warten, bis beide Sortier-Threads beendet sind; dabei die Erkennung eines
'möglichen Programmabbruchs offen halten.
Do While Not AutoResetEvent.WaitAll(locAutoResetEvents, 1, True)
    If myAbortAllThreads = True Then
        locFirstSortThread.Dispose()
        locSecondSortThread.Dispose()
        RemoveHandler MainThreadFinished, AddressOf MainThreadFinishedHandler
        Exit Sub
    End If
Loop

'Beide sortierten String-Arrays wieder zusammenführen.
Dim locIndexOnSecond As Integer
Dim locTempArray As New ArrayList(cAmountOfElements)
For locIndexOnFirst As Integer = 0 To locFirstAL.Count - 1
    Do
        If locIndexOnSecond < locSecondAL.Count Then
            If CType(locFirstAL(locIndexOnFirst), IComparable).CompareTo( _
                CType(locSecondAL(locIndexOnSecond), IComparable)) < 0 Then
                locTempArray.Add(locFirstAL(locIndexOnFirst))
                Exit Do
            Else
                locTempArray.Add(locSecondAL(locIndexOnSecond))
                locIndexOnSecond += 1
            End If
        Else
            locTempArray.Add(locFirstAL(locIndexOnFirst))
            Exit Do
        End If
    Loop
    'Möglichen Abbruch auch an dieser Stelle berücksichtigen.
    If myAbortAllThreads = True Then
        locFirstSortThread.Dispose()
        locSecondSortThread.Dispose()
        RemoveHandler MainThreadFinished, AddressOf MainThreadFinishedHandler
        Exit Sub
    End If
Next

```

```

'Falls es Reste aus dem zweiten Array gibt, diese auch noch kopieren.
If locIndexOnSecond < (locSecondAL.Count - 1) Then
    For c As Integer = locIndexOnSecond To locSecondAL.Count - 1
        locTempArray.Add(locSecondAL(c))
    Next
End If
locStrings = CType(locTempArray.ToArray(GetType(String)), String())

locGauge.Stop()
AddTBText("...in " + locGauge.DurationInMilliseconds.ToString() + " Millisekunden" + vbNewLine)
AddTBText(vbNewLine)
End If

'Falls das entsprechende Flag gesetzt ist, die Ergebnisse in die Textbox schreiben.
If cShowResults Then
    For Each s As String In locStrings
        If myAbortAllThreads Then
            RemoveHandler MainThreadFinished, AddressOf MainThreadFinishedHandler
            Exit Sub
        End If
        AddTBText(s + vbNewLine)
        AddTBText(vbNewLine)
    Next
End If

'Ereignis auslösen, das die Schaltflächen wieder einschaltet.
RaiseEvent MainThreadFinished()
'Ereignishandler wieder entfernen.
RemoveHandler MainThreadFinished, AddressOf MainThreadFinishedHandler

End Sub

```

---

**WICHTIG:** Wenn ein Thread einen Ereignishandler über RaiseEvent aufruft, gehört dieser Ereignishandler immer zum Ereignis auslösenden Thread, ganz egal, welcher Klasse der Ereignis-Handler zugeordnet ist. Deshalb gilt auch für eine Routine, die durch ein Thread-Ereignis aufgerufen wurde: Steuerelemente, die hier manipuliert werden, können nur über Invoke manipuliert werden, damit sie innerhalb des UI-Threads verändert werden.

Diese Tatsache macht die Thread-Programmierung von Windows-Benutzeroberflächen vergleichsweise aufwändig. Abhilfe schafft hier die BackgroundWorker-Komponente, die im ► Abschnitt »Threads durch den Background-Worker initialisieren« ab Seite 978 besprochen wird.

---

```

Private Sub MainThreadFinishedHandler()
    Console.WriteLine("MainThread beendet: " + Thread.CurrentThread.Name)
    Me.Invoke(New TSEnableControlsDelegate(AddressOf TSEnableControlsActually))
End Sub

Private Sub TSEnableControlsActually()
    btnBeenden.Enabled = True
    btnBenchmarkStarten.Enabled = True
End Sub

```

```

Private Sub btnBeenden_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnBeenden.Click
    'Nur schließen. OnClosing synchronisiert.
    Me.Close()
End Sub
Protected Overrides Sub OnClosing(ByVal e As System.ComponentModel.CancelEventArgs)
    'Falls eine TextBox-Ausgabe läuft, warten, bis diese abgeschlossen ist.
    'Sonst erfolgt die Ausgabe womöglich gerade zu der Zeit, wenn der Anwender
    'das Formular geschlossen hat. Dann wird die TextBox allerdings entladen,
    'ihr Handle zerstört, und die Ausgabe löst eine Ausnahme aus, weil sie das
    'zerstörte Handle der TextBox verwenden will.
    Do While Not myAutoResetEvent.WaitOne(1, True)
        Application.DoEvents()
    Loop
    'Ausgabe sicher --> jetzt erst abbrechen.
    myAbortAllThreads = True
End Sub
End Class

```

**TIPP:** Denkbar für die Lösung dieses Beispiels ist auch noch eine andere Vorgehensweise, die obendrein noch effektiver ist: Der Haupt-Thread könnte auf die AutoResetEvent-Objekte zum Warten auf das Beenden des Sortievorgangs komplett verzichten und sich stattdessen selbst mit Suspend aussetzen, nachdem er das Sortieren angeworfen hat. Die Sortierroutinen könnten ihrerseits durch das Auslösen des »Fertig!«-Ereignisses einen Ereignis-Handler aufrufen, der den Haupt-Thread wieder in Gang und schließlich zum Abschluss bringt. Das Beispiel des folgenden Abschnittes, dessen Hauptaufgabe eigentlich die Demonstration des Thread-Pools ist, verdeutlicht diese Vorgehensweise nebenbei.

---

## Verwenden des Thread-Pools

In den bisherigen Beispielen haben Sie Threads als Instrument kennen gelernt, die nur dann einmalig hervorgeholt werden, wenn man sie braucht. In Anwendungen, die unter Praxisbedingungen arbeiten, sieht das oft anders aus: Sicherlich lässt sich voraussagen, welche Threads in Anwendungen häufiger und welche weniger häufig benötigt werden; doch in jedem Fall macht es die Praxis notwendig – im Gegensatz zu den bislang gezeigten Beispielen – dass Thread-Objekte ständig erzeugt, verwendet, ausgesetzt und wieder zerstört werden; unter Umständen passiert das Hunderte Male in einer komplexen Applikation.

Das Erstellen eines Threads gehört für das Betriebssystem mit zu den aufwändigeren Dingen, die es zu erfüllen hat. Schon das Erstellen eines Threads kostet, gerade wenn es für ständig sich wiederholende, kleinere Aufgaben zigfach wiederholt werden muss, wertvolle Ressourcen. Aus diesem Grund bedient man sich eines Tricks, wenn innerhalb einer Anwendung viele kleine Threads eingesetzt werden sollen. Threads werden nicht ständig neu, sondern innerhalb eines so genannten Thread-Pools definiert. Wenn eine Anwendung einen Thread aus dem Thread-Pool anfordert, dann wird dieser beim ersten Mal zwar auch neu erstellt, doch anschließend nicht sofort wieder gelöscht, wenn die Anwendung ihn nicht mehr benötigt.

Stattdessen bleibt seine Grunddefinition im Pool erhalten, und das Thread-Objekt muss nicht komplett neu erstellt werden, wenn die Anwendung es für eine andere Aufgabe benötigt. Sie muss lediglich die Adresse der neuen Arbeits-Thread-Prozedur angeben; ansonsten kann das Thread-Objekt des Thread-Pools weiterverwendet werden.

Das Anlegen eines Threads aus dem Thread-Pool gestaltet sich eigentlich noch simpler als das Anlegen eines Threads mit der herkömmlichen Thread-Klasse. Sie übergeben der statischen Methode `QueueUserWorkItem` lediglich die Adresse der Prozedur, die als Thread ausgeführt werden soll. Im Gegensatz zum normalen Thread können Sie der Thread-Routine sogar eine Variable vom Typ `Object` als Parameter mitgeben (optional, Sie müssen also nicht).

---

**HINWEIS:** Threads, die über den Thread-Pool erstellt werden, laufen als so genannte Hintergrund-Threads. Diese Threads haben eine besondere Eigenschaft: Wenn der Haupt- bzw. UI-Thread der Applikation endet, werden auch die Hintergrund-Threads automatisch beendet.

---

Zusammenfassend gesagt, haben also Thread-Pool-Threads folgende Vorteile:

- Sie benötigen keinen zusätzlichen Programmieraufwand, um sie vorzeitig beenden zu können, da sie als Hintergrund-Thread eingerichtet werden.
- Sie benötigen kein zusätzliches `Delegate`-Objekt (Startobjekt), um gestartet zu werden.
- Sie werden in einem Rutsch definiert und gestartet.
- Ihre benötigten Ressourcen lassen sich schneller reservieren, da intern die Thread-Instanz nicht gelöscht sondern nur »ruhiggestellt« wird und damit ohne Aufwand für andere Zwecke wieder verwendet werden kann.
- Der Arbeits-Thread-Routine kann einen Parameter vom Typ `Object` empfangen.

Allerdings gibt es auch Nachteile: Maximal stehen pro verfügbaren Prozessor 25 Threads im Thread-Pool zur Verfügung. Wenn Sie einen neuen Thread darüber hinaus anfordern, wartet die dafür zuständige Methode `QueueUserWorkItem` solange, bis ein zurzeit noch reserviertes Thread-Objekt des Pools wieder verfügbar ist.

---

**BEGLEITDATEIEN:** Das folgende Beispiel demonstriert das Sortieren mit Threads aus dem Thread-Pool. Sie finden das Projekt dieses Beispiels im Verzeichnis `.\VB 2005 - Entwicklerbuch\G - SmartClient\Kap31\ThreadPoolThreads`. Da es weitestgehend dem vorherigen Beispiel entspricht, zeigt das folgende Codelisting nur die geänderten Stellen.

---

Betrachten wir zunächst das Codelisting des Sortier-Threads, der nunmehr auf einem Threadpool-Thread beruht (die eigentliche Sortierroutine ist in dieser Version aus Platzgründen ausgelassen).

Public Class SortArrayThreaded

```
Public Event SortCompleted(ByVal sender As Object, ByVal e As SortCompletedEventArgs)
Private myArrayToSort As ArrayList
Private myTerminateThread As Boolean
Private mySortingComplete As Boolean
Private myThreadName As String
```

```

'Konstruktor übernimmt die zu sortierende Liste. Der ThreadName hat nur
'dokumentarischen Charakter.
Sub New(ByVal ArrayToSort As ArrayList, ByVal ThreadName As String)
    myArrayToSort = ArrayToSort
    myThreadName = ThreadName
End Sub

'Thread starten
Public Sub StartSortingAsynchron()
    ThreadPool.QueueUserWorkItem(AddressOf SortArray)
End Sub

Private Sub SortArray(ByVal state As Object)
    .
    .'Wie gehbat.
    .
End Function

Public ReadOnly Property ArrayList() As ArrayList
    Get
        Return myArrayToSort
    End Get
End Property
End Class

```

Damit eine den Thread in Gang setzende Instanz weiß, welcher Thread beendet wurde, wenn mehrere Sortierungen gestartet werden, sind die Ereignisparameter in dieser Version leicht erweitert worden und geben nunmehr auch den Thread-Namen zurück:

```

'Dient nur der Ereignisübergabe, wenn Sie das SortCompleted-Ereignis
'nach Abbruch oder Abschluss des Sortierens auslösen wollen.
Public Class SortCompletedEventArgs
    Inherits EventArgs

    Private mySortCompletionStatus As SortCompletionStatus
    Private myThreadName As String

    Sub New(ByVal scs As SortCompletionStatus, ByVal ThreadName As String)
        mySortCompletionStatus = scs
    End Sub

    Public Property ThreadName() As String
        Get
            Return myThreadName
        End Get
        Set(ByVal Value As String)
            myThreadName = Value
        End Set
    End Property

```

```

Public Property SortCompletionStatus() As SortCompletionStatus
    Get
        Return mySortCompletionStatus
    End Get
    Set(ByVal Value As SortCompletionStatus)
        mySortCompletionStatus = Value
    End Set
End Property
End Class

```

Die wesentlichen Unterschiede im eigentlichen Arbeits-Thread beschränken sich auf das Warten des Sortierungsabschlusses. Während im vorherigen Beispiel eine Warteschleife einiges an Rechenzeit gekostet hat, wird der Arbeits-Thread hier komplett ausgesetzt. Erst die Ereignisbehandlungsroutine des SortCompleted-Ereignisses setzt den Arbeits-Thread wieder in Gang, der seine Aufgabe dann zu Ende führen kann:

```

'Dies ist die eigentliche Haupt-Thread-Routine (aber nicht der UI-Thread!), die das Testarray
'generiert und dann selbst wiederum entweder ein oder zwei Arbeits-Threads aufruft,
'die die eigentliche Sortierung durchführen.
Private Sub StartMainThread()

    Dim locStrings(cAmountOfElements - 1) As String
    Dim locGauge As New HighSpeedTimeGauge

    Dim locRandom As New Random(DateTime.Now.Millisecond)
    Dim locChars(cCharsPerString) As Char

    'Damit die Controls nach Abschluss des Sortierens auf dem Formular
    'wieder eingeschaltet werden können, muss der Haupt-Thread das Ende
    'der Sortierung mitbekommen. Deswegen: Ereignis
    AddHandler MainThreadFinished, AddressOf MainThreadFinishedHandler

    'String-Array erzeugen; hatten wir schon zig Mal...
    AddTBText("Erzeugen von " + cAmountOfElements.ToString + " Strings..." + vbNewLine)
    locGauge.Start()
    For locOutCount As Integer = 0 To cAmountOfElements - 1
        For locInCount As Integer = 0 To cCharsPerString - 1
            Dim locIntTemp As Integer = Convert.ToInt32(locRandom.NextDouble * 52)
            If locIntTemp > 26 Then
                locIntTemp += 97 - 26
            Else
                locIntTemp += 65
            End If
            locChars(locInCount) = Convert.ToChar(locIntTemp)
        Next
        locStrings(locOutCount) = New String(locChars)
    Next
    locGauge.Stop()
    AddTBText("...in " + locGauge.DurationInMilliSeconds.ToString + " Millisekunden" + vbNewLine)
    AddTBText(vbNewLine)
    locGauge.Reset()

```

```

If Not myUseTwoThread Then
    'Messung starten - hier wird mit nur einem Thread sortiert.
    locGauge.Start()
    Dim locFirstSortThread As New SortArrayThreaded(New ArrayList(locStrings), "1. SortThread")

    AddTBText("Sortieren von " + cAmountOfElements.ToString() + " Strings mit einem Thread..." _
              + vbNewLine)

    'Handler hinzufügen, der das Ende des Sortierens mitbekommt und dann wiederum
    'mySortCompletion hochzählt, damit die Warteschleife des Hauptthreads beendet werden kann
    AddHandler locFirstSortThread.SortCompleted, AddressOf SortCompletionHandler

    'los geht's
    locFirstSortThread.StartSortingAsynchron()

    'Warten, bis der Sortier-Thread abgeschlossen ist.
    'mySortCompletion wird durch das SortCompletion-Ereignis hochgezählt
    'wenn es 1 erreicht hat, schmeißt der Ereignishandler diesen Thread wieder an.
    mySyncUIResetEvent.WaitOne()

    'Handler wieder entfernen
    RemoveHandler locFirstSortThread.SortCompleted, AddressOf SortCompletionHandler

    'Messung beenden
    locGauge.Stop()
    AddTBText("...in " + locGauge.DurationInMilliseconds.ToString() + " Millisekunden" + vbNewLine)
    AddTBText(vbNewLine)
    'Array-List zurück-casten.
    locStrings = CType(locFirstSortThread.ArrayList.ToArray(GetType(String)), String())
Else
    'Es soll mit zwei Threads sortiert werden
    .
    .
    .
End Sub

```

Der Thread hält hier, nachdem er die Ereignisbehandlungsroutine eingerichtet und den Sortievorgang gestartet hat, komplett an (siehe fett markierte Zeilen im Listing). Durch die folgende Ereignisroutine wird er wieder zum Leben erweckt, wenn der Sortievorgang abgeschlossen wurde:

```

Private Sub SortCompletionHandler(ByVal sender As Object, ByVal e As SortCompletedEventArgs)
    mySortCompletion += 1
    If myUseTwoThread Then
        If mySortCompletion = 2 Then
            mySyncUIResetEvent.Set()
        End If
    Else
        If mySortCompletion = 1 Then
            mySyncUIResetEvent.Set()
        End If
    End If
End Sub

```

---

**HINWEIS:** Diese Version des Beispiels verzichtet der Einfachheit halber auf die Möglichkeit, den Thread vorzeitig abzubrechen.

---

## Thread-sichere Formulare in Klassen kapseln

Steuerelemente können von Threads aus nur durch deren Invoke thread-sicher bearbeitet werden – die vergangenen Abschnitte haben das in aller Ausführlichkeit gezeigt. Wenn Sie Ihre Formulare thread-sicher gestalten wollen, dann ist es das Beste, wenn Sie nach Möglichkeit die dazugehörigen Threads in den Formular-Klassen kapseln – etwa, wie es der ► Abschnitt »Datenaustausch zwischen Threads durch Kapseln von Threads in Klassen« ab Seite 959 gezeigt hat. Auch dabei gilt natürlich, dass Sie die Steuerelemente eines solchen Formulars aus seinen Arbeits-Threads heraus nur über Invoke verarbeiten dürfen.

Für einige Anwendungen kann es jedoch sinnvoll sein, dass ein Thread selbst ein Formular erstellt und es unter seine Verwaltung stellt. Denken Sie an die Beispiele, die Sie schon kennen gelernt haben: Einige der ersten Testprogramme haben auf eine ADThreadSafeInfoBox-Klasse zurückgegriffen, die die thread-sichere Ausgabe von Text in einem Formular erlaubte, obwohl an keiner Stelle die Instanzierung einer auf *Form* basierten Klasse erfolgte.

Wenn ein Thread völlig unabhängig vom dem ihn einbindenden Programm ein Formular erstellen will, hat er nur eine Chance: Er muss für das Formular (und alle weiteren, die dieses Formular aufruft), eine eigene Warteschlange implementieren.

Wenn Sie also eine komplette Klasse schaffen wollen, die einen Arbeits-Thread beinhaltet, der selbst auf ein Formular zugreifen muss, dann sind folgende Schritte nötig:

- Die den Thread startende Prozedur muss zunächst einen neuen Thread erstellen, der zum zusätzlichen UI-Thread wird.
- Dieser neue Thread instanziert dann das Hauptformular des neuen UI-Threads und bindet das Ereignis Application.ThreadExit ein.
- Mit Application.Run kann er nun im neu geschaffenen Thread eine Nachrichtenwarteschlange erstellen und die Instanz des Formulars an diese binden. Der Thread ist dann automatisch beendet, wenn der Anwender (oder eine andere Instanz) das gebundene Formular auf irgendeine Weise schließt.
- Für den Fall, dass die umgebende Applikation zuvor geschlossen wurde, löst das Application-Objekt das ThreadExit-Ereignis aus. Die Klasse muss in der Ereignisbehandlungsroutine dieses Ereignisses ein paar Aufräumarbeiten durchführen: Durch das eigentliche Programmende ist der zweite UI-Thread nämlich noch nicht ebenfalls automatisch abgeschlossen. Ein explizites Application.ExitThread wird an dieser Stelle noch notwendig, um den zweiten UI-Thread spätestens jetzt zu beenden.

Das folgende Beispiel geht sogar noch einen Schritt weiter, um absolute Unabhängigkeit zu erlangen: Es erstellt den zweiten UI-Thread im statischen Konstruktor der Klasse. Wenn eine andere Instanz die statische Funktion TSWrite (oder TSWriteLine) das erste Mal aufruft, um eine Ausgabe in das Statusfenster durchzuführen, sorgt der statische Konstruktor dafür, dass der neue UI-Thread erstellt wird. Die Ausgabe erfolgt dann anschließend in das nun vorhandene Fenster (durch ein einfaches TextBox-Steuerelement simuliert). Damit der Konstruktor dem neuen UI-Thread ein wenig Zeit gibt,

in Gang zu kommen und das Formular zu instanzieren, und damit verhindert wird, dass TSWrite[Line] eine Ausgabe in ein Steuerelement durchführt, das noch nicht existiert, erfolgt die notwendige Synchronisierung mit einem ManualResetEvent-Objekt.

```
Imports System.Threading

Public Class ADThreadSafeInfoBox
    Private Shared myText As String
    Private Shared myInstance As ADThreadSafeInfoBox
    Private Shared myThread As Thread
    Private Shared myManualResetEvent As ManualResetEvent

    Private Delegate Sub TSWriteActallyDelegate()
    Private Delegate Sub ThisInstanceCloseDelegate()

    Private Shared myThisInstanceCloseDelegate As ThisInstanceCloseDelegate

    Private Shared Sub ThisInstanceCloseActually()
        myInstance.Dispose()
    End Sub

    'Statischer Konstruktor erstellt neuen UI-Thread
    Shared Sub New()
        myText = ""
        myManualResetEvent = New ManualResetEvent(False)
        CreateInstanceThroughThread()
    End Sub

    'Teil des Kontruktors; könnte theoretisch auch von
    'anderswo in Gang gesetzt werden. Diese Routine erstellt
    'den neuen UI-Thread und startet ihn...
    Private Shared Sub CreateInstanceThroughThread()
        myThread = New Thread(New ThreadStart(AddressOf CreateInstanceThroughThreadActually))
        myThread.Name = "2. UI-Thread"
        myThread.Start()
    End Sub

    '...und der neue UI-Thread erstellt jetzt das Formular
    'und bindet es an eine neue Nachrichtenwarteschlange.
    Private Shared Sub CreateInstanceThroughThreadActually()
        'Instanz des Formulars erstellen
        myInstance = New ADThreadSafeInfoBox
        'Wir müssen auf jeden Fall wissen, wann die Hauptapplikation (der Haupt-UI-Thread) geht.
        myThisInstanceCloseDelegate = New ThisInstanceCloseDelegate(AddressOf ThisInstanceCloseActually)

        AddHandler Application.ThreadExit, AddressOf ThreadExitHandler
        'Und wir müssen wissen, ab wann das Formular wirklich existiert;
        'vorher dürfen keine Ausgaben ins Formular erfolgen
        AddHandler myInstance.HandleCreated, AddressOf HandleCreatedHandler
        'und ab wann nicht mehr. Für dieses Beispiel ist dieses Ereignis nicht soooo wichtig.
        AddHandler myInstance.HandleDestroyed, AddressOf HandleDestroyedHandler
    End Sub

```

```

'Hier wird die Warteschlange gestartet
Application.Run(myInstance)
End Sub

'Dieser Ereignis-Handler wird aufgerufen, wenn das Hauptprogramm beendet wird.
'Der zweite UI-Thread wird damit beendet.
Private Shared Sub ThreadExitHandler(ByVal sender As Object, ByVal e As EventArgs)
    Console.WriteLine("ThreadExit")
    'Keine TextBox mehr vorhanden:
    ' Synchronisationsvoraussetzung für TSWrite schaffen
    myManualResetEvent.Reset()
    Try
        myInstance.Invoke(myThisInstanceCloseDelegate)
    Catch ex As Exception
    End Try
End Sub

'TSWrite signalisieren, dass die Ausgabe in die TextBox jetzt sicher ist,
'da das Formular-Handle erstellt wurde.
Private Shared Sub HandleCreatedHandler(ByVal sender As Object, ByVal e As EventArgs)
    Console.WriteLine("HandleCreated")
    myManualResetEvent.Set()
End Sub

'Vielleicht später mal wichtig; hier nur zur Demo.
Private Shared Sub HandleDestroyedHandler(ByVal sender As Object, ByVal e As EventArgs)
    'Nur für Testzwecke
    Console.WriteLine("HandleDestroyed")
End Sub

'Nutzt TSWrite; siehe dort.
Public Shared Sub TSWriteLine(ByVal Message As String)
    SyncLock (GetType(ADThreadSafeInfoBox))
        Message += vbNewLine
        TSWrite(Message)
    End SyncLock
End Sub

'Ausgabe ohne neue Zeile
Public Shared Sub TSWrite(ByVal Message As String)
    SyncLock (GetType(ADThreadSafeInfoBox))
        'Synchronisierung: Wenn nach 50 Millisekunden
        'keine TextBox vorhanden ist --> Befehl ignorieren
        If Not myManualResetEvent.WaitOne(50, True) Then
            Exit Sub
        End If
        myText += Message
        Try
            myInstance.Invoke(New TSWriteActuallyDelegate(AddressOf TSWriteActually))
        Catch ex As Exception
        End Try
    End SyncLock
End Sub

```

```

'Thread-sichere Ausgabe in die TextBox mit Invoke
Private Shared Sub TSWriteActually()
    myInstance.txtOutput.Text = myText
    myInstance.txtOutput.SelectionStart = myText.Length - 1
    myInstance.txtOutput.ScrollToCaret()
End Sub
End Class

```

## Threads durch den Background-Worker initiieren

Neu im .NET-Framework 2.0 ist die so genannte `BackgroundWorker`-Komponente, die die meiner Meinung nach einfachste Methode darstellt, eine rechenintensive Methode in einem anderen Thread laufen zu lassen.

Falls Sie dieses Kapitel aufmerksam studiert haben, werden Sie feststellen, dass der Einsatz dieser Komponente zwar nicht so viel Flexibilität bietet, wie die Programmierung von Threads über das `Thread`-Objekt; aber durch eine besondere intern angewandte Technik hat sie einen unschlagbaren Vorteil: Sie kann Statusmeldungen über Ereignisse zur Verfügung stellen, die im Ursprungs-Thread (in der Regel also dem UI-Thread) laufen; der Aufwand, sich `Invoke` bedienen zu müssen, um beispielsweise ein `Label`-Steuerelement zu aktualisieren, fällt also weg.

Die Vorgehensweise, um mithilfe der `BackgroundWorker`-Komponente eine Prozedur als neuen Thread laufen zu lassen, ist außergewöhnlich aber simple: Ihr Thread läuft als Ereignisbehandlungsprozedur.

- Sie triggern die `BackgroundWorker`-Komponente mit der Methode `RunWorkerAsync`. Die wiederum sorgt dafür, dass das `DoWork`-Ereignis ausgelöst wird, und ihre implementierte Behandlungsroutine, die dieses Ereignis behandelt, *ist* die Prozedur, die auf einem anderen Thread läuft.
- Möchten Sie, dass Fortschrittsinformationen aus ihrer Thread-Routine heraus gesendet werden, sorgen Sie vor Beginn des Threads, dass die `WorkerReportsProgress`-Eigenschaft auf `True` gesetzt ist, denn nur dann unterstützt die `BackgroundWorker`-Komponente das Berichten von Fortschrittsstatus. Auch die Kommunikation des Fortschritts funktioniert über Ereignisse. Wenn die entsprechenden Vorbereitungen getroffen wurden, lösen Sie zu gegebener Zeit innerhalb Ihres Threads (also dem `DoWork`-Ereignisbehandler) das `ProgressChanged`-Ereignis aus, das Sie mit der `ReportProgress`-Methode initiieren.

---

**WICHTIG:** Dieses Ereignis läuft dann, anders als zu erwarten wäre, auf dem Thread, der die `BackgroundWorker`-Komponente kapselt (in der Regel also dem UI-Thread), und nicht auf dem Thread, der das `ProgressChanged`-Ereignis ausgelöst hat. UI-Komponenten können deswegen direkt in diesem Ereignis angesprochen werden!

---

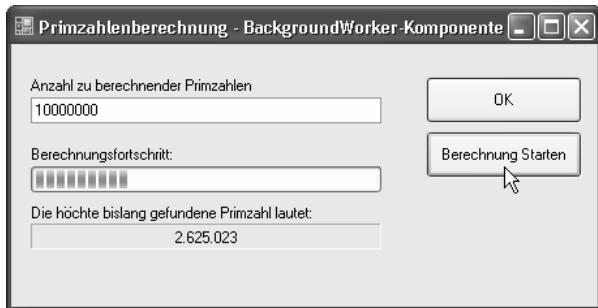
- Wenn der `DoWork`-Thread beendet wurde, löst die `BackgroundWorker`-Komponente automatisch das `RunWorkerCompleted`-Ereignis aus. Eine Prozedur, die dieses Ereignis behandelt, läuft auch wieder auf dem ursprünglichen Thread (also in der Regel dem UI-Thread) und nicht auf dem Thread, der durch `DoWork` initiiert wurde.

---

**BEGLEITDATEIEN:** Das Projekt für das folgende Beispiel finden Sie im Verzeichnis `.\VB 2005 - Entwicklerbuch\G - SmartClient\Kap31\BackgroundWorkerDemo\`.

---

Das Beispiel dient zur Berechnung von Primzahlen in einem eigenen Thread. Wenn Sie es starten, sehen Sie einen Dialog, wie Sie ihn auch in Abbildung 31.8 sehen können.



**Abbildung 31.8:** Mit dem *BackgroundWorker* können Sie auf einfachste Weise neue Threads starten, und bekommen Fortschritts-Informationen auf dem ursprünglichen UI-Thread!

Der entsprechende Code des Programms sieht folgendermaßen aus:

```
Public Class frmMain

    Private myPrimzahlen As List(Of Long)

    Private Sub btnBerechnungStarten_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles btnBerechnungStarten.Click

        'Festlegen, dass der BackgroundWorker über den Fortschritt berichten können soll.
        PrimzahlenBackgroundWorker.WorkerReportsProgress = True

        'Fortschrittsanzeige konfigurieren.
        pbBerechnungsfortschritt.Minimum = 0
        pbBerechnungsfortschritt.Maximum = 1000
        pbBerechnungsfortschritt.Value = 0
        lblGefundenText.Text = "Die höchste bislang gefundene Primzahl lautet:"

        'Ereignis auslösen lassen, damit der Thread
        'in der DoWork-Ereignisbehandlungsroutine gestartet werden kann.
        'Die Obergrenze wird aus dem Textfeld als Argument übergeben.
        Me.PrimzahlenBackgroundWorker.RunWorkerAsync(txtAnzahlPrimzahlen.Text)
    End Sub

    Private Sub PrimzahlenBackgroundWorker_DoWork(ByVal sender As Object, _
        ByVal e As System.ComponentModel.DoWorkEventArgs) _
        Handles PrimzahlenBackgroundWorker.DoWork

        'Das Argument (die Obergrenze) aus den Ereignisparametern wieder herauslesen.
        Dim locMax As Integer = CInt(e.Argument)

        'Ein paar Variablen für die Fortschrittsanzzeige einrichten.
        '(OK: Es sind Promille).
        Dim locProgressFaktor As Double = 1000 / locMax
        Dim locProgressAlt As Integer = 0
        Dim locProgressAktuell As Integer
```

```

'Fertig, wenn Obergrenze kleiner 3 ist.
If (locMax < 3) Then Return

'Neues Array definieren, dass die Primzahlen enthält.
'(Wir verwenden als Algorithmus das Sieb des Eratosthenes.)
myPrimzahlen = New List(Of Long)

For z As Integer = 2 To locMax
    If IstPrimzahl(myPrimzahlen, z) Then
        myPrimzahlen.Add(z)

        'Progressinformation senden. Das darf, da der Thread durch SendMessage
        '"gewechselt" wird, nicht zu häufig passieren, sonst hängt der
        'UI-Thread, der die BackgroundWorker-Komponente anfangs initiiert hat.
        locProgressAktuell = CInt(z * locProgressFaktor)
        If locProgressAktuell > locProgressAlt Then
            locProgressAlt = locProgressAktuell
            'Das ProgressChange-Ereignis auslösen, um über den
            'Fortschritt zu informieren.
            PrimzahlenBackgroundWorker.ReportProgress(locProgressAktuell)
        End If
    End If
Next
End Sub

Private Function IstPrimzahl(ByVal Primzahlen As List(Of Long), ByVal Number As Long) As Boolean
    For Each locTestZahl As Long In Primzahlen
        If (Number Mod locTestZahl = 0) Then Return False
        If (locTestZahl >= Math.Sqrt(Number)) Then Exit For
    Next
    Return True
End Function

'Wird aufgerufen, wenn eine BackgroundWorker-Komponente mit
'ReportProgress über den Fortschritt informiert. Dieser Ereignisbehandler
'läuft dennoch auf dem UI-Thread und nicht dem DoWork-Thread. Er kann daher
'ohne Probleme Steuerelemente direkt manipulieren.
Private Sub PrimzahlenBackgroundWorker_ProgressChanged(ByVal sender As Object, _
    ByVal e As System.ComponentModel.ProgressChangedEventArgs) _
    Handles PrimzahlenBackgroundWorker.ProgressChanged
    pbBerechnungsfortschritt.Value = e.ProgressPercentage
    lblHöchstePrimzahl.Text = myPrimzahlen(myPrimzahlen.Count - 1).ToString("#,##0")
End Sub

'Wird aufgerufen, wenn der DoWork-Thread beendet wurde.
'Auch dieser Ereignisbehandler läuft auf dem UI-Thread.
Private Sub PrimzahlenBackgroundWorker_RunWorkerCompleted(ByVal sender As Object, _
    ByVal e As System.ComponentModel.RunWorkerCompletedEventArgs) _
    Handles PrimzahlenBackgroundWorker.RunWorkerCompleted
    lblGefundenText.Text = "Die höchste gefundene Primzahl im Bereich lautet:"
    lblHöchstePrimzahl.Text = myPrimzahlen(myPrimzahlen.Count - 1).ToString("#,##0")
End Sub
End Class

```

---

**HINWEIS:** Achten Sie darauf, das ProgressChanged-Ereignis nicht zu häufig auszulösen, da Sie den UI-Thread unter Umständen durch das Aktualisieren von Steuerelementen überlasten könnten. Der DoWork-Thread läuft dabei zwar regelmäßig weiter, der UI-Thread reagiert dann aber quasi nicht mehr.

---

## Threads durch asynchrone Aufrufe von Delegaten initiieren

Delegaten haben Sie in ► Kapitel 15 kennen gelernt, und für genauere Informationen zu diesem Thema lesen Sie bitte dort nach. Kurz zusammengefasst: Sie dienen zum indirekten Aufruf von Methoden über in bestimmten Objektvariablen abgelegten Adressen dieser Methoden.

Mit der Methode BeginInvoke haben Sie die Möglichkeit, eine Prozedur, die durch einen Delegaten dargestellt wird, asynchron, also auf einem eigenen Thread laufen zu lassen. Sie sollten EndInvoke aufrufen, wenn die Arbeit erledigt ist. Mit BeginInvoke haben Sie die Möglichkeit, einen zweiten Delegaten anzugeben, der aufgerufen wird, wenn die Arbeit abgeschlossen wurde.

Das folgende Beispiel, das ebenfalls Primzahlen berechnet, demonstriert den Einsatz des asynchronen Ausführens von Prozeduren über Delegaten.

---

**BEGLEITDATEIEN:** Das Projekt für das folgende Beispiel finden Sie im Verzeichnis `.\VB 2005 - Entwicklerbuch\G-SmartClient\Kap31\AsyncDelegates\AsyncDelegates`.

---

```
Imports System.Collections.Generic
Imports System.ComponentModel

Public Class Form1

    Private Delegate Sub PrimzahlenErstellenDelegate(ByVal Max As Integer)
    Private Delegate Sub ErgebnisSetzenDelegate(ByVal ErgebnisText As String)

    Private myPrimzahlen As List(Of Long)
    Private myPZDelegate As PrimzahlenErstellenDelegate
    Private myErgebnisSetzenDelegate As ErgebnisSetzenDelegate
    Private myAsyncOperation As AsyncOperation

    Sub New()

        ' Dieser Aufruf ist für den Windows Form-Designer erforderlich.
        InitializeComponent()

        ' Fügen Sie Initialisierungen nach dem InitializeComponent()-Aufruf hinzu.
        myErgebnisSetzenDelegate = New ErgebnisSetzenDelegate(AddressOf ErgebnisSetzen)
    End Sub

    Private Sub btnBerechnungStarten_Click(ByVal sender As System.Object,
                                         ByVal e As System.EventArgs) Handles btnBerechnungStarten.Click
        'Delegate definieren.
        myPZDelegate = New PrimzahlenErstellenDelegate(AddressOf PrimzahlenErstellen)
    End Sub

    Private Sub PrimzahlenErstellen(ByRef result As List(Of Long), ByVal max As Integer)
        Dim i As Integer
        For i = 2 To max
            If IsPrime(i) Then
                result.Add(i)
            End If
        Next
    End Sub

    Private Function IsPrime(ByVal number As Integer) As Boolean
        Dim i As Integer
        For i = 2 To number - 1
            If number Mod i = 0 Then
                Return False
            End If
        Next
        Return True
    End Function

    Private Sub ErgebnisSetzen(ByVal text As String)
        txtErgebnis.Text = text
    End Sub
End Class
```

```

'Delegate ansynchron aufrufen.
myPZDelegate.BeginInvoke(Integer.Parse(txtAnzahlPrimzahlen.Text), _
AddressOf BerechnungAbgeschlossenCallback, Nothing)
End Sub

Private Sub PrimzahlenErstellen(ByVal Max As Integer)

If (Max < 3) Then Return
myPrimzahlen = New List(Of Long)

For z As Integer = 2 To Max
If IstPrimzahl(myPrimzahlen, z) Then
    myPrimzahlen.Add(z)
    'Direkte Manipulation des Steuerelements ist nicht möglich,
    'da dieser Thread nicht dem UI-Thread entspricht.
    lblHöchstePrimzahl.Invoke(myErgebnisSetzenDelegate, _
        myPrimzahlen(myPrimzahlen.Count - 1).ToString("#,#0"))
End If
Next
End Sub

Private Function IstPrimzahl(ByVal Primzahlen As List(Of Long), ByVal Number As Long) As Boolean

For Each locTestZahl As Long In Primzahlen
    If (Number Mod locTestZahl = 0) Then Return False
    If (locTestZahl >= Math.Sqrt(Number)) Then Exit For
Next
Return True
End Function

'Delegatroutine für das Aktualisieren der Benutzeroberfläche.
'Notwendig, weil die Aktualisierung auf dem Thread des Delegaten ausgeführt wird.
Private Sub ErgebnisSetzen(ByVal ErgebnisText As String)
    lblHöchstePrimzahl.Text = ErgebnisText
End Sub

'Der Rückroutine des Delegaten, der aufgerufen wird, wenn seine Aufgabe beendet ist.
Private Sub BerechnungAbgeschlossenCallback(ByVal ar As System.IAsyncResult)
    lblHöchstePrimzahl.Invoke(myErgebnisSetzenDelegate, _
        myPrimzahlen(myPrimzahlen.Count - 1).ToString("#,#0"))
    myPZDelegate.EndInvoke(ar)
End Sub

Private Sub btnOK_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnOK.Click
    Me.Close()
End Sub
End Class

```

# 32 SQL Server 2005 und ADO.NET

---

- 
- 984 SQL Server 2005 Express**
  - 992 Grundsätzliches zu Datenbanken**
  - 998 Programmieren mit ADO.NET**
  - 1009 Ändern und Ergänzen von Daten in Datentabellen**
  - 1021 DataSets und typisierte DataSets**
  - 1037 Und so geht es weiter**
- 

Wenn Anwendungen entwickelt werden, dann müssen diese in 80 % aller Fälle mit Daten hantieren. In der Regel greift dann nicht nur *ein* Computer auf die Daten zu, sondern *mehrere* Computer müssen zur gleichen Zeit mit einer Datenquelle arbeiten. Zwar ist das Framework durchaus in der Lage, auch komplexere Datenstrukturen über Rechnergrenzen hinaus zu verwalten, doch der Einsatz von Datenbanksystemen ist für solche Lösungen angezeigt. Doch auch für Einzelplatzlösungen ist der Einsatz von Datenbankanwendungen eine sinnvolle Angelegenheit. Dank SQL (*Structured Query Language*, etwa: strukturierte Abfragesprache) können Daten auf einfache Weise sortiert und gefiltert werden, und da Daten sowieso nach einer Arbeitssitzung gespeichert werden müssen, bietet sich der Einsatz einer Datenbank auch für Insellösungen mehr als an.

Im Framework gibt es ein mächtiges Werkzeug zum Abfragen und Verwalten von Daten in Datenbanken namens ADO.NET. ADO ist die Abkürzung von *ActiveX Data Objects* (etwa: *ActiveX-Datenobjekte*), und es stellt eine Klassenbibliothek zur Verfügung, mit deren Hilfe die verschiedensten Datenbanksysteme gesteuert werden können – sei es Microsofts SQL Server,<sup>1</sup> Oracle oder Access, um nur einige wenige zu nennen.

Durch eine weitestgehend vorhandene Standardisierung von SQL auf der einen und einem konsequenten Einsatz von Klassen und Klassenvererbung im Framework auf der anderen Seite, unterscheiden sich die Vorgehensweisen bei der Programmierung mit den verschiedenen Datenanbietern obendrein nur marginal.

---

<sup>1</sup> Sprich: »Siehkwl sörwer«.

---

**HINWEIS:** Einige der in diesem Kapitel vorgestellten Funktionen können Sie leider nicht mit der Visual Basic Express Edition durchführen, weil ihr die entsprechende Funktionalität fehlt (wie beispielsweise der Server-Explorer). Das bedeutet natürlich nicht, dass Sie keine Datenbankprogrammierung mit Visual Basic Express durchführen können! Visual Basic Express leistet sprachtechnisch das Gleiche, wie jede andere Version von Visual Basic bzw. Visual Studio, und dazu gehört natürlich auch ADO.NET.

---

## SQL Server 2005 Express

SQL Server Express ist der Nachfolger der Microsoft SQL Server Desktop Engine, vielen eher bekannt unter dem Namen MSDE. Beide basieren auf ihren größeren Brüdern, SQL Server 2005 bzw. SQL Server 2000, ihnen fehlt jedoch einiges in Sachen Funktionalität für Administrationsaufgaben. Darüber hinaus erfuhren sie ebenfalls ein paar Einschnitte in ihrer Leistungsfähigkeit, aber dafür stellt Microsoft beide Systeme unendgeldlich zur Verfügung, und sie lassen sich damit als professionelle Datenbankplattformen in Ihren eigenen Anwendungen einsetzen.

SQL Server Express erfuhr die folgenden Einschränkungen:

- Datenbanken sind auf maximal 4 GByte beschränkt.
- Maximal ein Prozessor eines Systems wird verwendet.
- Maximal ein Gigabyte Hauptspeicher eines Systems kommt zur Anwendung.
- Maximal fünf so genannter Pipes werden gleichzeitig verarbeitet. Damit lässt sich, natürlich je nach wirklicher Auslastung, eine die Datenbank durchschnittlich in Anspruch nehmende Büroanwendung durchaus mit 20, 30 Clients betreiben, bevor Leistungseinschnitte bemerkbar werden.

---

**HINWEIS:** Denken Sie daran, dass SQL Server Express sich auch auf Computern installieren lässt, die über mehr Hauptspeicher oder mehrere Prozessoren verfügen. Diese werden dann lediglich nicht genutzt.

Denken Sie auch daran, dass Sie sich bei Microsoft explizit registrieren lassen müssen, bevor Sie eine eigene Anwendung auf Basis von SQL Server Express vertreiben. Das kostet zwar nichts (Stand der Drucklegung dieses Buches), Sie müssen es aber halt machen. Der IntelliLink *G3201* bringt Sie direkt zu dieser Registrierungsseite (Sie benötigen dafür ein Microsoft Passport Konto).

---

Von diesen Einschränkungen abgesehen, steht Ihnen mit SQL Server Express eine Datenbankplattform zur Verfügung, die dem großen Bruder – was die relationale Datenbank angeht – in beinahe nichts nachsteht. Umfangreiche, höchst performante Datenbanken lassen sich mit dieser Software realisieren und die volle Programmierfähigkeit mit gespeicherten Prozeduren (*Stored Procedures*), Funktionen und auch die Ausführungsfähigkeit von verwaltetem Code steht Ihnen mit SQL Server Express zur Verfügung.

Sie können SQL-Express direkt von der Microsoft Website herunterladen. Folgen Sie dazu dem IntelliLink *G3201*.

## Einsatz von SQL Server Express in eigenen Anwendungen

SQL Server Express wird ab Visual Studio 2005 Standard Edition direkt mitinstalliert. Solange, wie Sie sich quasi »zu Hause«, in Ihrer eigenen Entwicklungsumgebung befinden, werden Ihnen daher keine Probleme beim Aufbau, der Programmierung oder der Anwendung einer Datenbank begegnen.

Anders sieht das schon aus, wenn Sie SQL Server Express später in seiner Produktivumgebung einsetzen wollen. Dort begegnen Ihnen dann unter Umständen Herausforderungen, mit denen Sie sich nicht erst zum Zeitpunkt der Installation Ihrer Anwendung beschäftigen sollten, denn:

Anwendungen, die auf SQL Server Express basieren, lassen sich solange problemlos installieren, wie SQL Server Express auf dem gleichen Rechner wie Ihre Anwendung selbst läuft. Etwas aufwändiger wird es, wenn

- Ihre Anwendung von mehreren Rechnern aus auf eine SQL Server Express-Instanz zugreifen soll.
- Ihnen kein Server-Betriebssystem, das als Active Directory eingerichtet ist, als Plattform für SQL Server Express zur Verfügung steht, oder Sie SQL Server Express nicht zumindest auf einem Computer (Windows 2000 SP4, XP Professional, eine Windows Vista-Version mit Domänenanbindungsmöglichkeit) installieren können, der Mitglied einer Domäne ist.

Der Grund: Standardmäßig wird SQL Server Express (wie übrigens auch seine großen Brüder) so installiert, dass es die integrierte Sicherheit des Betriebssystems für Anmeldungen an einer SQL Server-Instanz oder einer Datenbank nutzt. Solange wie ein Benutzer (oder eine Anwendung) sich auf dem gleichen System an der SQL Server-Instanz anmeldet, auf dem SQL Server Express selbst auch installiert wurde, ist das kein Problem. Problematisch wird bei dieser Anmeldemethode die Anmeldung von Rechnern des Netzwerkes, wenn kein Domänencontroller zur Verfügung steht, der eine vertraute Verbindung zwischen dem Client-Computer und dem Computer, der die SQL Server-Instanz beherbergt, bestätigen könnte.<sup>2</sup>

Für solche Fälle ist die so genannte *Gemischter Modus*-Authentifizierung angezeigt. Bei dieser Art der Authentifizierung können Sie sowohl die integrierte Sicherheit von Windows als auch die Authentifizierungsmethode verwenden, die vom SQL Server-System selbst zur Verfügung gestellt wird. Der Nachteil: Letzterer besteht in geringeren Sicherheitsstandards. Um eine Verbindung auf diese Weise zu einem SQL Server aufzubauen, müssen Kennwort und Benutzername beim Verbindungsaufbau angegeben werden. Die Wahrscheinlichkeit, dass Anmelddaten dadurch kompromittiert werden könnten, ist daher ungleich höher, wenn sie nicht auf geschickte Weise in Ihrer Software verschlüsselt werden.

Ein weiterer Nachteil ist, dass Sie in Netzwerken, in denen Active Directory nicht zur Verfügung steht, keine automatisierte Installation (die so genannte »stille Installation« von SQL Server Express) durchführen sollten, da diese standardmäßig *Integrated Security* als Zugangsart einrichtet. Um sich spätere Konfigurationsarbeit zu sparen, ergibt es mehr Sinn, schon während der Installation ein Administratorkennwort für die Anmeldung an der Instanz ohne integrierte Sicherheit vergeben und auch die Anmeldemethode direkt auf *Gemischter Modus* einstellen. Da entsprechende Administrationswerkzeuge fehlen, ist die spätere Umstellung nur mit entsprechenden Systemaufrufen durch

---

<sup>2</sup> Jedenfalls wenn Sie als Administrator am Rechner angemeldet sind – was die meisten Entwickler ja leider häufig sind. Als »normaler« Benutzer muss man auch erst einmal gezielt Zugriff vom SQL Server erteilt bekommen... CREATE LOGIN bzw. CREATE USER FROM LOGIN lauten hier die Stichworte, die genauer zu beschreiben den Rahmen an dieser Stelle allerdings sprengen würden.

gespeicherte Prozeduren zu bewerkstelligen und damit für ungeübte Daten-Banker nicht unbedingt so ohne weiteres machbar.<sup>3</sup>

## Installation von SQL Server 2005 Express unter Windows XP im »gemischten Modus«

Die folgende Schritt-für-Schritt-Anleitung zeigt, wie Sie SQL Server Express unter Windows XP mit vorhandenem Service Pack 2 (SP2) installieren, wenn Ihnen kein Domänencontroller zur Verfügung steht. Sie haben die Möglichkeit, eine »stille Installation« durchführen zu lassen, oder aber die Konfiguration mithilfe eines Assistenten zu durchlaufen. Die folgenden Abschnitte beschreiben die assistentengestützte Installation. Sie setzen die finale deutsche Version von *SQL Server 2005 Express Edition* voraus.

- Sie starten die manuelle Installation mit einem Doppelklick auf *sqlexpr.exe*. Wenn Sie die Installation von SQL Server Express gestartet haben, sehen Sie anschließend folgenden Dialog:



**Abbildung 32.1:** Bestätigung des Endanwenderbenutzervertrags

- Klicken Sie auf *Ich stimme den Bedingungen des Lizenzvertrags zu*, um die Lizenzvereinbarung zu akzeptieren und klicken Sie anschließend auf *Weiter*, um zum nächsten Schritt zu gelangen.
- Klicken Sie auf *Installieren*, um mit der Installation zu beginnen. Der Installationsassistent richtet nun die erforderlichen Komponenten ein, die für die weitere Installation erforderlich sind.
- Klicken Sie anschließend auf *Weiter*. Wenn die System-Konfigurationsüberprüfung abgeschlossen wurde, sehen Sie die eigentlichen Installationsassistenten von SQL Server Express, etwa wie in der folgenden Grafik zu sehen.

<sup>3</sup> Im Internet sind aber auch »Antwortdateien« für die stille Installation verfügbar, die SQL Server direkt im gemischten Modus einrichten.



**Abbildung 32.2:** Die Willkommensmeldung zur SQL Server 2005 Express-Installation

- Klicken Sie auf *Weiter*, um zur erweiterten Systemkonfigurierungsüberprüfung zu gelangen. Auf Maschinen mit weniger als 512 MByte-Hauptspeicher sehen Sie möglicherweise eine *Mindestanforderung an die Hardware*-Warnung. Für kleinere Datenbankanwendungen reicht ein Hauptspeicher von 256 MByte (Windows 2000) bzw. 386 MByte (Windows XP, Windows 2003) aber aus, so dass Sie in diesem Fall die Warnung ignorieren können.<sup>4</sup> Klicken Sie auf *Weiter*.

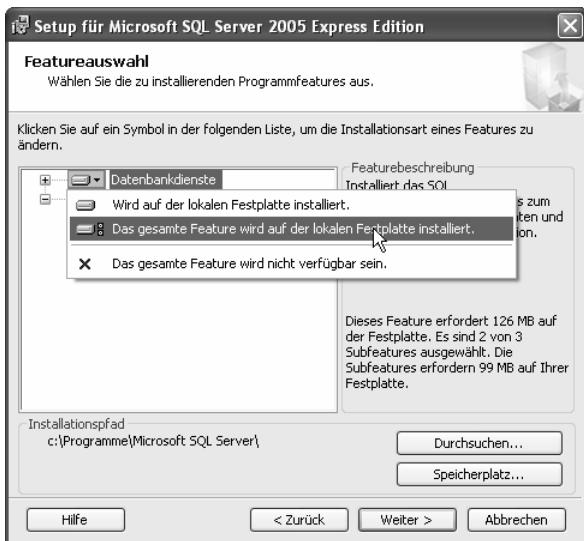


**Abbildung 32.3:** Entfernen Sie in diesem Dialog das Häkchen für den Zugriff auf erweiterte Konfigurationsoptionen

---

<sup>4</sup> Das gilt nur, wenn keine anderen speicherintensiven Anwendungen auf diesem Rechner aktiv sind.

- Im nächsten Dialog, den Sie in Abbildung 32.3 sehen, geben Sie die gewünschten Daten ein. Wichtig: Achten Sie dabei darauf, das Häkchen *Erweiterte Konfigurationsoptionen ausblenden* zu entfernen, damit Sie während der weiteren Installation Zugriff auf die notwendigen erweiterten Installationsoptionen nehmen können.



**Abbildung 32.4:** Bestimmen Sie in diesem Dialog den Installationsumfang

- Klicken Sie anschließend auf *Weiter*.
- Ändern Sie im folgenden Dialog den Installationsumfang, etwa wie in Abbildung 32.4 gezeigt. Verfahren Sie genauso für die Installation der Client-Komponenten.

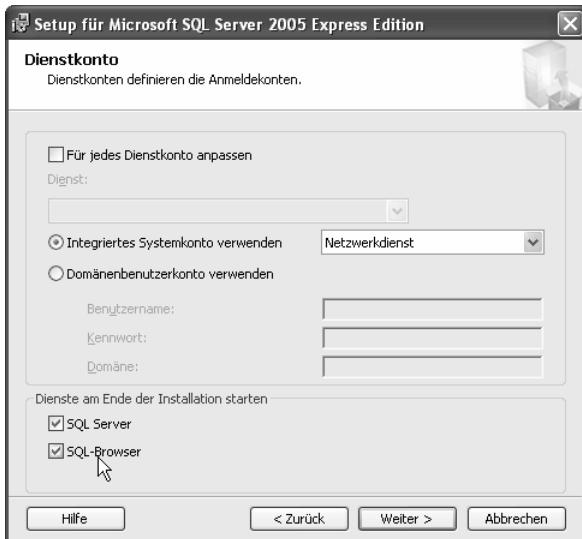


**Abbildung 32.5:** In diesem Dialog bestimmen Sie, unter welchem Namen die SQL Server-Instanz später zu erreichen ist

- Klicken Sie anschließend auf *Weiter*.
- Bestimmen Sie im nächsten Dialog den Namen der SQL Server-Instanz (Abbildung 32.5), die Ihre Datenbanken später verwalten soll. Sie können die Voreinstellung *Benannte Instanz: SQLExpress* belassen wie sie ist, wenn es nicht bereits eine SQL Server Instanz unter diesem Namen auf demselben Rechner gibt.

**TIPP:** Falls Sie sich mit einer »alten« Anwendung, die beispielsweise ODBC verwendet, gegen den SQL Server verbinden wollen, kann die Benutzung einer benannten Instanz Probleme bereiten. Benutzen Sie dann die Standardinstanz. Falls Sie aber ausschließlich mit .NET und ADO.NET arbeiten, bieten die benannten Instanzen eher mehr Möglichkeiten. So kann man alle Datenbanken zum Verkauf in der Instanz »Verkauf« unterbringen usw. Solche Instanzen arbeiten dann wie unabhängige SQL Server und können einzeln gestartet und gestoppt werden, sie verfügen über eine eigene Benutzerverwaltung, etc.

- Klicken Sie auf *Weiter*.



**Abbildung 32.6:** Bestimmen Sie die Aktivierung des SQL-Browser-Dienstes, damit diese SQL-Instanz später von anderen Computern des Netzwerkes aus per Instanznamen gefunden werden kann

- Damit Sie die SQL Server-Instanz später beispielsweise mit dem Verbindungsdialog des Server-Explorers von Visual Studio finden können, schalten Sie den SQL Browser-Dienst auf diesem Computer direkt mit ein, indem Sie das entsprechende Häkchen setzen.

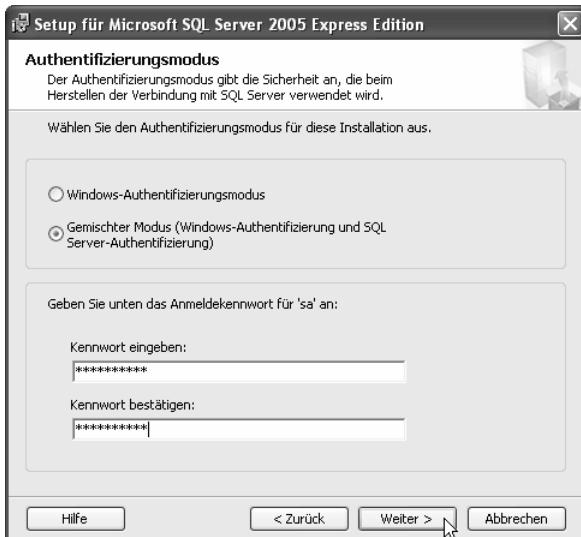
**TIPP:** Für den professionellen Einsatz von SQL Server sollten Sie Folgendes beherzigen: Die Vorgabe »Netzwerkdienst« ist das Konto mit den geringsten Rechten. Damit wird es böswilligen Angreifern erheblich erschwert, über den SQL Server Zugriff auf ihr System zu erlangen. Der Netzwerkdienst ist daher dem ebenfalls verfügbaren Konto »Locales System« auf jeden Fall vorzuziehen. Für die weitergehenden Möglichkeiten sollten Sie jedoch ein eigenes so genanntes »Dienstkontos« einrichten und hier im Setup bestimmen. Man könnte auf die Idee kommen, dass auch ein späterer Wechsel auf ein Dienstkontos z.B. über die Systemsteuerung und dem Eintrag »Dienste« möglich wäre. Dies ist auf den ersten Blick auch richtig; das Dienstkontos für den SQL Server benötigt aber für bestimmte Aufgaben sehr »spezielle« Rechte, wie zum Beispiel für das Ausführen von Festplattenwartungen (nur dann etwa funktioniert das neue Feature der schnellen Dateiinitialisierung).

---

Alle benötigten Rechte werden aber hier im Setup automatisch richtig vergeben: Sie sparen sich also viel Zeit und Mühe, wenn Sie hier direkt das gewünschte Konto als Dienstkonto eingeben. Falls Sie erwägen, solche Möglichkeiten wie Replikation (also den automatisierten Datenabgleich zwischen mehreren SQL Servern im Netz) zu benutzen, sollten Sie hier sogar ein Domänenkonto angeben.

---

- Klicken Sie anschließend auf *Weiter*.



**Abbildung 32.7:** Verwenden Sie *Gemischter Modus*, wenn Sie die SQL Server-Instanz in einem Nicht-Domänen-Netzwerk von anderen Computern des Netzwerks erreichen wollen

- Im nächsten Dialog (Abbildung 32.7) bestimmen Sie den Authentifizierungsmodus, mit dem sich Clients an der Instanz und an von ihr verwalteten Datenbanken anmelden können. Wenn Ihre Anwendung ausschließlich auf der gleichen Maschine läuft, die auch die SQL Server-Instanz enthält oder SQL Server auf einer Maschine läuft, die Teil einer Active Directory Domäne ist, wählen Sie die sichere Authentifizierungsmethode *Windows-Authentifizierungsmodus*. Ist keine Domäne vorhanden, müssen Sie zur Anmeldung an einer SQL Server Instanz Anmeldeinformationen übergeben; das funktioniert nur, wenn die Instanz zuvor in den *gemischten Modus* geschaltet wurde. In diesem Fall bestimmen Sie ebenfalls ein Systemadministrator-Kennwort (»sa«), mit dem Sie sich später an der Instanz anmelden können.

---

**HINWEIS:** Ein sehr bekannter Virus (SQL Slammer) benutzte beim SQL Server 2000 die menschlich verständliche Schwäche, dass Benutzer damals zumeist KEIN Kennwort für das sa-Konto angaben; schon deshalb warnt das Setup hier deutlich. Vergeben Sie daher auf jeden Fall ein sicheres Kennwort (mind. 6 Zeichen, Groß- und Kleinschreibung, Sonderzeichen und Zahlen, kein Wort der deutschen Sprache, also beispielsweise im Stil von »P@a\$\$w0rd«)

---

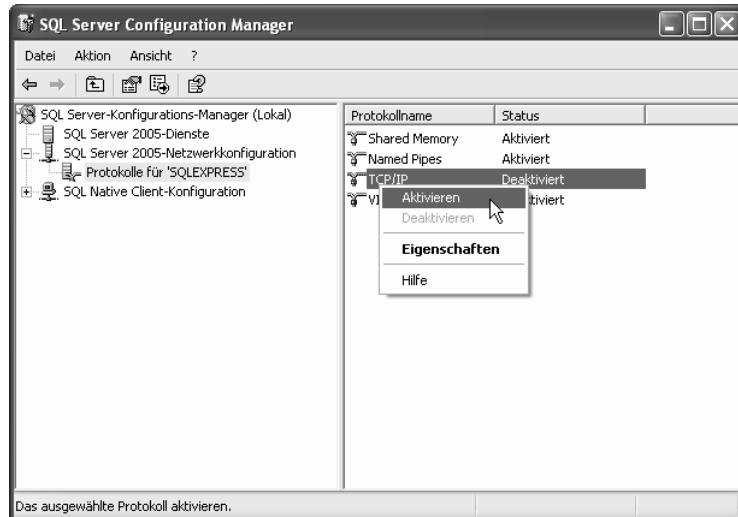
- Klicken Sie anschließend auf *Weiter*.

Die Einstellungen der nächsten drei Dialoge können Sie so belassen, wie sie vorgegeben sind. Klicken Sie dreimal auf *Weiter*, und anschließend auf *Installieren*, um den eigentlichen Installationsvorgang zu starten und die Einrichtung abzuschließen.

## Konfiguration der Netzwerkeinstellungen und Einrichten der Firewall unter Windows XP SP2

Damit eine SQL Server-Instanz im Netzwerk erreichbar ist, müssen Sie die Netzwerkprotokolle konfigurieren und ggf. einschalten. Sollten Sie SQL Server Express auf einem Windows XP-System mit Service Pack 2 betreiben und dieser Rechner nicht Teilnehmer einer Active Directory Domäne sein, müssen Sie obendrein die Firewall konfigurieren:

- Um die Netzwerkprotokolle einzurichten, starten Sie den SQL Configuration Manager aus dem Startmenü (*Start/Alle Programme/Microsoft SQL Server 2005/Konfigurationstools/SQL Server Configuration Manager*).
- Öffnen Sie in der linken Baumstruktur den Zweig *SQL Server 2005-Netzwerkkonfiguration*, und klicken Sie auf *Protokolle für 'SQLEXPRESS'*.



**Abbildung 32.8:** Aktivieren Sie in diesem Dialog die entsprechenden Protokolle, wenn Sie von anderen Rechnern des Netzwerkes aus auf diese SQL Server-Instanz zugreifen wollen

- Öffnen Sie in der rechten Liste das Kontextmenü über dem Eintrag *Named Pipes*. Wählen Sie aus dem Kontextmenü den Eintrag *Aktivieren*.
- Öffnen Sie das Kontextmenü in der rechten Liste über dem Eintrag *TCP/IP*. Wählen Sie aus dem Kontextmenü den Eintrag *Aktivieren*.

### Firewall-Einstellungen unter Windows XP SP2

Falls sich SQL Server auf einem Windows XP-SP2-Rechner befindet, der nicht zu einer Active Directory-Domäne gehört, schalten Sie die entsprechenden Ports für den SQL Server- bzw. SQL Browser-Dienst frei. Dazu wählen Sie aus der Systemsteuerung *Windows Firewall*.



**Abbildung 32.9:** Öffnen Sie TCP-Port 1433 sowie UDP-Port 1434, um »von Außen« den SQL Server erreichen zu können. Öffnen Sie ebenfalls den Port für die Datei- und Druckerfreigabe, um über Named Pipes auf SQL Server zugreifen zu können (Port 445, bereits in der Ausnahmenliste vorhanden).

- Aktivieren Sie in der Ausnahmenliste die Datei- und Druckerfreigabe, da nur so die Ansteuerung über Named Pipes funktionieren wird (Port 445). Dieser Port ist standardmäßig in der Ausnahmenliste vorhanden.
- Klicken Sie auf der Registerkarte *Ausnahmen* auf *Port*, und geben Sie den TCP Port 1433 frei. Dieser Port wird standardmäßig für die erste SQL Server-Instanz vergeben.
- Öffnen Sie zusätzlich den UDP-Port 1434.

## Grundsätzliches zu Datenbanken

Bevor Sie beginnen, mit Datenbanken zu programmieren, sollten Ihnen die Grundlagen zu diesem Thema bekannt sein. Bitte haben Sie dabei Verständnis dafür, dass ich an dieser Stelle aus Platzgründen nicht sehr ausführlich auf das Thema eingehen kann. Was die verschiedenen Datenbanksysteme anbelangt, so haben sie eines auf jeden Fall gemeinsam: Sie organisieren die zu verwaltenden Daten in Zeilen und Spalten verschiedener Tabellen. Die Spalten enthalten dabei verschiedene Datenfelder, wie etwa Name, Vorname, Personal-Nr. oder Postleitzahl und Ort bei einer Adressentabelle. Die einzelnen Zeilen entsprechen den eigentlichen Datensätzen. Am deutlichsten wird dieser Zusammenhang am konkreten Beispiel.

## Aufbau der Beispieldatenbank

Die folgenden Beispiele verwenden eine Beispieldatenbank, die verschiedene Daten eines fiktiven Unternehmens abbildet. Dazu gehören:

- Berater, die im Unternehmen arbeiten, und mit der Bearbeitung bestimmter Projekte wie beispielsweise Softwareentwicklung oder dem Schreiben von Dokumentationen oder Büchern betraut sind.
- Projekte, die die Berater derzeit bearbeiten.
- Eine Zeittabelle, die darüber Auskunft gibt, welcher Mitarbeiter an welchem Projekt zu welcher Zeit gearbeitet hat.

Diese Daten wurden zufällig erzeugt. Dazu diente ein Programm, das nicht nur in der Lage ist, diese Daten zu generieren, sondern das Ihnen auch einen Einblick in die Struktur der Daten geben kann.

---

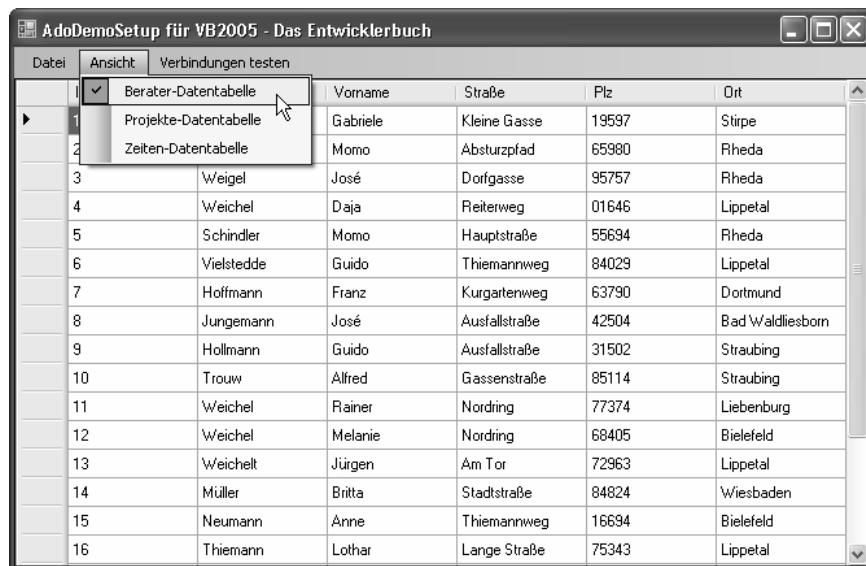
**BEGLEITDATEIEN:** Sie finden die Projektmappe für dieses Beispiel im Verzeichnis .\VB 2005 - Entwicklerbuch\H - Database\AdoDemoSetup\AdoDemoSetup.sln. Voraussetzung dafür ist, dass Sie von dem Computer, auf dem Sie dieses Beispiel ausprobieren möchten, einen Zugriff auf eine SQL Server 2005-Instanz haben.

---

Nach dem ersten Start fragt Sie das Programm nach einer SQL Server 2005-Instanz. Verwenden Sie SQL Server 2005 Express Edition auf Ihrem Entwicklungsrechner (was mit Visual Studio ab der Standard Edition automatisch mit installiert wird), dann klicken Sie einfach auf das entsprechende Kontrollkästchen, um die Standardinstanz zu verwenden, und beenden Sie den Dialog mit *OK*.

Die Demo-Datenbank wird anschließend automatisch erstellt, und Sie gelangen in einen Dialog, mit dem Sie die Möglichkeit haben, die Datenbank »mit Fleisch«, also mit fiktiven Daten zu füllen.

Anschließend (und bei jedem Neustart nach der ersten Einrichtung) bietet sich Ihnen ein Bild, wie Sie es auch in der folgenden Abbildung sehen können. Über das *Ansicht*-Menü haben Sie hier die Möglichkeit zu bestimmen, welche der drei Datentabellen, aus denen die Datenbank besteht, Sie in dem DataGridView-Steuerelement darstellen lassen möchten.



The screenshot shows a Windows application window titled "AdoDemoSetup für VB2005 - Das Entwicklerbuch". The menu bar includes "Datei", "Ansicht", and "Verbindungen testen". The "Ansicht" tab is selected. A dropdown menu is open under "Ansicht", showing three options: "Berater-Datentabelle" (selected, indicated by a checked checkbox), "Projekte-Datentabelle", and "Zeiten-Datentabelle". The main area contains a DataGridView with 16 rows of data. The columns are labeled "Vorname", "Straße", "Plz", and "Ort". The data is as follows:

	Vorname	Straße	Plz	Ort
1	Gabriele	Kleine Gasse	19597	Stirpe
2	Momo	Absturzpfad	65990	Rheda
3	Weigel	José	95757	Rheda
4	Weichel	Daja	01646	Lippetal
5	Schindler	Momo	55694	Rheda
6	Vielstedde	Guido	84029	Lippetal
7	Hoffmann	Franz	63790	Dortmund
8	Jungemann	José	42504	Bad Waldliesborn
9	Hollmann	Guido	31502	Straubing
10	Trouw	Alfred	85114	Straubing
11	Weichel	Rainer	77374	Liebenburg
12	Weichel	Melanie	68405	Bielefeld
13	Weichelt	Jürgen	72963	Lippetal
14	Müller	Britta	84824	Wiesbaden
15	Neumann	Anne	16694	Bielefeld
16	Thiemann	Lothar	75343	Lippetal

**Abbildung 32.10:** Dieses Programm ist nicht nur in der Lage die Demodatenbank zu generieren, sondern stellt die Daten der einzelnen Tabellen auch dar

Anhand dieser Abbildung können Sie genau sehen, wie die Datentabelle zeilen- und spaltenweise organisiert ist. Die einzelnen Datenzeilen werden übrigens auch *Datensätze* genannt (eine einzige Datenzeile ist ein *Datensatz*).

---

**TIPP:** Falls Sie im Laufe der Zeit zu viel mit der Datenbank herumgespielt und sie damit sehr in Unordnung gebracht haben, wählen Sie aus dem Menü *Datei* den Menübefehl *Vorhandene Daten löschen und neu erstellen*.

---

# Klärung grundsätzlicher Begriffe

Für die verschiedenen Elemente, mit denen Sie bei der Programmierung von Datenbanken arbeiten, gibt es spezielle Begriffe, die Sie sich einprägen sollten:

## Verbindungen zur Datenbank über Connection-Objekte

Bevor Sie auch nur ein einziges Datum aus einer Datenbank herauskitzeln oder der Datenbank-Engine mitteilen können, dass sie selbst irgendwelche Operationen durchführen soll, müssen Sie eine *Verbindung* zur Datenbank herstellen. In ADO.NET verwenden Sie dafür das *Connection*-Objekt, um eine Verbindung zu einem *Daten-Provider* (etwa: *Datenanbieter*) aufzubauen (wie beispielsweise Microsoft Access oder SQL Server 2005). Es gibt die verschiedensten Datenbankprovider, und bei Daten-Provider abhängigen Objekten stehen die eigentlichen Provider bei der Namensgebung der Klassen Pate. So verwenden wir für alle folgenden Beispiele ausschließlich SQL Server als Datenprovider – die *Connection*-Klasse für diesen Provider nennt sich deshalb *SqlConnection*. Griffen Sie über *OleDb* beispielsweise auf eine Access-Datenbank zu, würden Sie dazu die *OleDbConnection*-Klasse verwenden.

## Befehle an die Datenbank mit dem Command-Objekt übermitteln

Nachdem Sie eine Verbindung zur Datenbank hergestellt haben, können Sie über *Command*-Objekte Befehle an die Datenbank senden. Diese Befehle dienen entweder dazu, Daten bestimmter Tabellen (die auch über SQL-Befehle miteinander verknüpft werden können) abzufragen oder andere Befehle – zum Beispiel zum Löschen von Datensätzen – an die Datenbank abzusetzen. Im Falle von SQL Server als Daten-Provider nennt sich das entsprechende *Command*-Objekt *SqlCommand*.

---

**TIPP:** *SqlCommand*-Objekte verwenden Sie ebenfalls, um gespeicherte Prozeduren auf einer SQL Server-Datenbank zu initiieren.

---

## Ermitteln von Resultsets durch SQL-Abfragen

Wenn über *Command*-Objekte bestimmte Befehle an die Datenbank übermittelt werden, die Daten zurückliefern, dann spricht man vom Zurückerhalten eines *Resultsets* – eines Ergebnissatzes (an Daten). Resultsets können beispielsweise mit einem *DataReader*-Objekt (*SqlDataReader* im Falle von SQL Server) direkt gelesen oder in ein *DataSet*-bzw. *DataTable*-Objekt übertragen werden.

## Prinzipielle Vorgehensweise beim Abfragen und Modifizieren von Daten

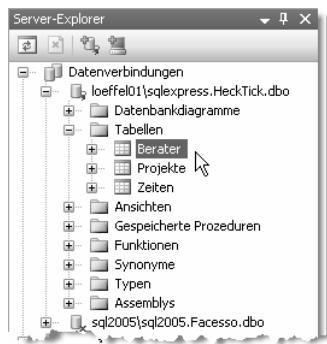
ADO.NET kennt zwei grundsätzliche Arbeitsmodi: den unverbundenen und den verbundenen. Beim verbundenen Modus trudeln die Daten nacheinander ein. Sie können Daten dabei nicht verändern, sondern nur lesen. Sie können die einzelnen Datensätze bei dieser Vorgehensweise nur der Abfrage entsprechend nacheinander einlesen; ein Sprung zurück oder ein Überspringen von Datensätzen ist dabei nicht möglich.

Der unverbundene Modus ist die Arbeitsweise, mit der Sie Daten nicht nur stringent auslesen, sondern frei editieren können. Bevor Sie im unverbundenen Modus arbeiten, wird natürlich eine Verbindung zur Datenbank hergestellt. Nach einer erfolgten Abfrage verwenden Sie beispielsweise ein `DataTable`-Objekt, um die abgefragten Daten komplett dort hinein zu laden. Danach stehen die Daten völlig getrennt von der eigentlichen Datenbank im Speicher Ihres Computers und können dort verarbeitet werden. Ab jetzt gibt es keine Verbindung mehr zur Datenbank. Erst wenn Sie die Verarbeitung abgeschlossen haben, schicken Sie sie – erst jetzt sind Sie wieder mit der Datenbank verbunden – zur Datenbank zurück.

## Einsehen von Daten mit dem Server-Explorer

Das beste Werkzeug zum Einsehen von Daten oder Tabellenstrukturen in einer Datenbank sind die Werkzeuge selbst, die die Datenbanken für diesen Zweck anbieten. Doch nicht immer stehen diese Tools auf dem Rechner zur Verfügung, auf dem man entwickelt. Für diese Fälle stellt Visual Studio (ab der Standard-Edition) den *Server-Explorer* zur Verfügung, der in Visual Studio 2005 die datenbankeigenen Tools eigentlich überflüssig macht.

Den Server-Explorer finden Sie unterhalb der Toolbox, wenn Sie die Standardeinstellung nicht geändert haben. Falls er nicht von vorne herein angezeigt wird, schalten Sie ihn einfach ein, indem Sie aus dem Menü *Ansicht* den Eintrag *Server-Explorer* auswählen.



**Abbildung 32.11:** Mit dem Server-Explorer können Sie auf einfache Weise Einsicht in Daten und Tabellenstrukturen nehmen

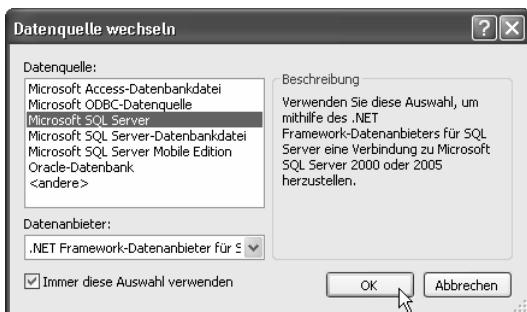
Um eine Verbindung wie in unserem Beispiel zur lokalen SQL Server-Express-Demodatenbank aufzubauen, verfahren Sie folgendermaßen:

- Im Server-Explorer klicken Sie mit der rechten Maustaste auf *Datenverbindungen*, und aus dem Kontextmenü, das jetzt erscheint, wählen Sie den Eintrag *Verbindung hinzufügen*.



**Abbildung 32.12:** So bauen Sie eine Verbindung zu einer Datenbank auf der lokalen SQL Server Express-Instanz auf und testen die Verbindung

- Im Dialog, der jetzt erscheint (siehe Abbildung 32.12), überprüfen Sie die Datenquelle, die *Microsoft SQL Server (SqlClient)* lautet muss. Falls das nicht der Fall ist, klicken Sie auf Ändern.



**Abbildung 32.13:** Wählen Sie aus diesem Dialog die Datenquelle aus, die Sie verwenden möchten

- Wählen Sie aus der Liste *Microsoft SQL Server* aus. Wählen Sie *.NET Framework-Datenanbieter für SQL Server* aus der Aufklappliste *Datenanbieter*, und bestätigen Sie den Dialog mit *OK*.
- Geben Sie – für den Fall, dass Sie auf die lokale SQL Server-Express-Instanz zugreifen wollen – als Servernamen *.\SQLEXPRESS* ein. Wählen Sie in diesem Fall *Windows-Authentifizierung verwenden*, um ihr Windows-Benutzerkonto auch für die Anmeldung am SQL Server zu verwenden.
- Im Bereich *Mit Datenbank verbinden*, klappen Sie die Aufklappliste der Datenbanknamen aus, und wählen damit die zu bearbeitende Datenbank – in unserem Beispiel die Datenbank *HeckTick*.

## Anekdoten aus der Praxis

Ein lieber Kollege von mir – Jürgen *Heckhuis*, und achten Sie auf seinen Nachnamen – seines Zeichens begnadeter Netzwerktechniker, kam eines Tages zu mir, und bat mich, ich möge ihm doch ein Beispiel für Visual Basic 2005 zum Üben geben, er möchte diese Programmiersprache nämlich erlernen. Gemeinsam bestimmten wir ein Zeiterfassungs- und Auswertungsprogramm als bestes Beispiel für viele der zu verwendenden Techniken in Visual Basic 2005 aus. Nachdem er sich einige Zeit – so glaubte ich jedenfalls – mit den ersten Startproblemen beschäftigt hatte, tönte es »Heureka!« aus seinem Büro. Ich eilte sofort dort hin und fragte ihn, ob er die erste Version bereits fertig habe. »Nein, nein«, entgegnete er, »ich hab noch gar nichts gemacht. Aber ich habe einen genialen Namen für das Programm: *HeckTick!!!*«

Dieser Kalauer blieb so sehr hängen, dass ich ihn für dieses Beispiel verwenden musste – nur für den Fall, dass Sie sich wundern ...

- Abschließend können Sie die Verbindung zur Datenbank mit der gleichnamigen Schaltfläche testen. Ansonsten klicken Sie auf OK, um den Vorgang abzuschließen.

Die Verbindung zur Datenbank wurde nun zur Liste der Datenverbindungen im Server-Explorer hinzugefügt.

IDBerater	Nachname	Vorname	Straße	Plz	Ort
1	Weichel	Daja	Aue	90562	Wuppertal
2	Sonntag	José	Nordring	25019	Wuppertal
3	Meier	Ulwe	Stadtstraße	36878	Liebenburg
4	Weichel	Axel	Crash Ave	93063	Lippeatal
5	Plenge	Barbara	Stadtstraße	12073	Lippeatal
6	Heckhuis	Momo	Kleine Gasse	81377	Lippeatal
7	Westermann	Melanie	Am Brunnen	52420	Bielefeld
8	Engisch	Barbara	Dorfstraße	44852	Unterschleißheim
9	Plenge	Anne	Crash Ave	49132	Straubing
10	Ademmer	Barbara	Kleine Gasse	06572	Wiesbaden
11	Tiemann	Daja	Reiterweg	48938	Soest
12	Braun	Alfred	Parkstraße	01878	Wuppertal
13	Hörstmann	Thomas	Crash Ave	36519	Wiesbaden
14	Vielstedde	Anne	Main Road	78164	Liebenburg
15	Braun	Hans	Alter Postweg	23034	Stippe
16	Rode	Lothar	Am Tor	31792	Braunschweig
17	Weigel	Guido	Gassenstraße	05897	Unterschleißheim
18	Weigel	José	Windpockenallee	33201	Unterschleißheim
19	Westermann	Katrin	Dorfplatz	43673	Lippeatal
20	Thiemann	José	Windpockenallee	76122	München
*	NULL	NULL	NULL	NULL	NULL

Abbildung 32.14: Ein Doppelklick auf eine Tabelle macht Daten und Struktur sichtbar

Sie können anschließend den nun vorhandenen Zweig unter Datenverbindungen öffnen. Die Verbindung zur gerade ausgewählten SQL Server-Datenbank ist jetzt dort zu sehen. Öffnen Sie auch die nächsten Zweige, werden die Tabellen sichtbar. Ein Doppelklick auf eine Tabelle zeigt Ihnen die Tabellenstruktur und die Daten an, etwa wie in Abbildung 32.14 zu sehen.

Sie können die Daten in dieser Ansicht ändern und auch neue Datensätze hinzufügen. Über die Kontextmenüs der einzelnen Elemente der Baumstruktur des Server-Explorers haben Sie ferner die Mög-

lichkeit, weitere Funktionen abzurufen, wie beispielsweise das Erstellen neuer Tabellen, das Verändern von Tabellenstrukturen, das Erstellen von gespeicherten Prozeduren und vieles mehr.

Wenn Sie die Datenvorschau nicht mehr benötigen, schließen Sie das Fenster einfach.

## Programmieren mit ADO.NET

Soviel zur theoretischen Vorbereitung. In diesem Abschnitt erfahren Sie nun Grundlegendes darüber, wie Sie die verschiedenen Objekte, die ADO.NET zur Verfügung stellt, nutzen können, um in eigenen Programmen Datenabfragen durchzuführen und Daten in Tabellen zu verändern und zu ergänzen.

### Verbindungen zur Datenbank mit der Connection-Klasse herstellen und Befehle mit der Command-Klasse ausführen

Um eine Verbindung zu einer Datenbank herzustellen, benötigen Sie ein *Connection*-Objekt. Um anschließend einen Befehl an die Datenbank (oder SQL Server-Instanz) zu senden, benötigen Sie ein *Command*-Objekt, das diese Befehle kapselt. Im Falle von SQL Server nennen sich diese beiden Klassen *SqlConnection* und *SqlCommand*. Die grundsätzliche Vorgehensweise, um mit der Datenbank in Kommunikation zu treten, lautet folgendermaßen (die relevanten Teile sind fett hervorgehoben):

```
'Wichtig: In diesem Namespace befinden sich die SQL Server ADO-Klassen
Imports System.Data.SqlClient
```

```
Module mdlMain
```

```
Sub Main()
    Dim locConnection As SqlConnection
    Dim locCommand As SqlCommand
    Dim locDataReader As SqlDataReader

    'Connection-Objekt holen, damit die Datenbankverbindung hergestellt werden kann
    locConnection = New SqlConnection _
        ("Data Source=.\SQLEXPRESS;Initial Catalog=Hecktick;Integrated Security=True")

    'Würden Sie den "gemischten Modus" verwenden, wäre Folgendes die richtige Wahl.
    'Aber aufgepasst: Das Passwort steht dann im Quellcode und kann leicht gefunden werden!
    'Eine Verschlüsselung des Passworts wenigstens mit einfachen Mitteln wäre dann angezeigt.
    'locConnection = New SqlConnection _
        ("Data Source=.\SQLEXPRESS;Initial Catalog=Hecktick;User ID=sa; Password=Irgendwas")

    'Wichtig: Verbindung muss geöffnet sein
    locConnection.Open()

    'Command-Objekt erstellen, mit der ein Befehl an die Datenbank abgesetzt
    'und ein ResultSet eingeholt werden kann
    locCommand = New SqlCommand("SQLBefehle", locConnection)

    'Mit Using sorgen wir dafür, dass die Verbindung wieder geschlossen wird,
    'wenn locConnection aus dem Using-Scope herausläuft.
    Using locConnection
```

```

'Command-Objekt einweisen, dass es den SQL-Befehl ausführt
locDataReader = locCommand.ExecuteReader()
End Using
End Sub
End Module

```

In diesem Beispiel würde das verwendete SqlCommand-Objekt natürlich zu einer Ausnahme führen, da es keinen gültigen Befehl enthält. Dieser kleine Codeabschnitt soll aber auch lediglich demonstrieren, wie Sie grundsätzlich verfahren, um eine Verbindung zur Datenbank herzustellen und entsprechende Befehle abzusetzen oder Daten durch Select-Abfragen anzufordern.

Der nächste Abschnitt zeigt das am konkreten Beispiel.

## Datenverbindungen herstellen und Resultsets mit der DataReader-Klasse auslesen

Das nachfolgend beschriebene Projekt macht nichts weiter, als mit dem SqlConnection-Objekt eine Verbindung zur Demo-Datenbank aufzubauen. Anschließend verwendet es ein SqlCommand-Objekt, um eine SELECT-Anweisung an die Datenbank abzusetzen und das von der Datenbank zurück gelieferte Resultset mit dem SqlDataReader auszulesen. Die Beispielanwendung ist der Einfachheit halber als Konsolenanwendung konzipiert.

---

**BEGLEITDATEIEN:** Sie finden dieses Beispiel im Verzeichnis .\VB 2005 - Entwicklerbuch\G - SmartClient\Kap32\DataReader\.

---

'Wichtig: In diesem Namespace befinden sich die SQL Server ADO-Klassen  
Imports System.Data.SqlClient

Module mdlMain

```

Sub Main()
    Dim locConnection As SqlConnection
    Dim locCommand As SqlCommand
    Dim locDataReader As SqlDataReader

    'Connection-Objekt holen, damit die Datenbankverbindung hergestellt werden kann
    locConnection = New _
        SqlConnection("Data Source=.\SQLEXPRESS;Initial Catalog=Hecktick;Integrated Security=True")

    'Würden Sie den "gemischten Modus" verwenden, wäre Folgendes die richtige Wahl.
    'Aber aufgepasst: Das Passwort steht dann im Quellcode und kann leicht gefunden werden!
    'Eine Verschlüsselung des Passworts wenigstens mit einfachen Mitteln wäre dann angezeigt.
    'locConnection =
    'New SqlConnection("Data Source=.\SQLEXPRESS;Initial Catalog=Hecktick;User ID=sa; Password=Irgendwas")

    'Command-Objekt erstellen, mit der ein Befehl an die Datenbank abgesetzt
    'und ein ResultSet eingeholt werden kann
    locCommand = New SqlCommand("SELECT TOP 15 * FROM Berater ORDER BY [Nachname]", locConnection)

    'Wichtig für den DataReader: Verbindung muss geöffnet sein
    locConnection.Open()

```

```

'Damit sorgen wir dafür, dass die Verbindung wieder geschlossen wird,
'wenn locConnection aus dem Using-Scope herausläuft.
Using locConnection
    Console.WriteLine("Inhalt der Tabelle Mitarbeiter (erste 15. Datensätze, nach Nachnamen sortiert)")

    'Command-Objekt einweisen, dass er ein Reader-Objekt mit Datenzugriff erstellt
    locDataReader = locCommand.ExecuteReader()

    'Feststellen, ob überhaupt Daten vorhanden sind:
    If locDataReader.HasRows Then
        'Read holt den jeweils nächsten Datensatz und wird False, wenn es keinen weiteren gibt
        Do While locDataReader.Read
            'Alle Datenspalten durchlaufen
            For c As Integer = 0 To locDataReader.FieldCount - 1
                'GetValue holt das eigentliche Datum als Object
                Console.WriteLine(locDataReader.GetValue(c).ToString + vbTab)
            Next
            'Datensatz komplett dargestellt, Zeilenwechsel
            Console.WriteLine()
        Loop
        'Auf Tastatureingabe warten
        Console.ReadLine()
    End If
End Using
End Sub

End Module

```

Um an die Daten heranzukommen, muss natürlich als erstes eine Verbindung zur SQL Server-Instanz hergestellt werden – dabei verwendet das Beispiel das Muster aus dem vorherigen Abschnitt.

Durch die ExecuteReader-Methode des SqlCommand-Objektes wird der im SqlCommand-Objekt gekapselte Befehl einerseits an den Server geschickt, andererseits erstellt diese Methode ein *DataReader*-Objekt vom Typ SqlDataReader, mit dem Sie anschließend Zugriff auf die Daten nehmen können, so die SQL-Datenbank tatsächlich Daten zurückgeliefert hat (was übrigens, wie im Beispiel zu sehen, mit der Eigenschaft HasRows des SqlDataReader-Objekts festgestellt werden kann).

---

**HINWEIS:** Ein *DataReader*-Objekt kann nicht direkt instanziert werden; Sie müssen es von einem *Command*-Objekt erstellen lassen, um es – wie im Beispiel zu sehen – anschließend verwenden zu können.

---

Die Read-Funktion des *DataReader*-Objektes sorgt dann dafür, dass der jeweils nächste Datensatz aus der Datenbank gelesen wird. Wichtig: Die Daten werden dabei tatsächlich zeilenweise von der Datenbank gelesen – mit jedem Aufruf der Read-Methode werden also die nächsten Daten von der Datenbank übertragen und können anschließend mit den GetXXX-Funktionen des *DataReader* abgerufen werden. Die Read-Methode liefert als Funktionsergebnis False zurück, wenn keine weiteren Datensätze mehr im Resultset zur Verfügung stehen. Beim Öffnen des *DataReader* steht der interne Zeiger also VOR dem ersten Datensatz, die erste Read-Anweisung verschiebt das Lesen auf die erste Zeile. Daher ist auch eine Prüfung, ob überhaupt Daten zurückgegeben wurden sehr einfach. Sollte Ihrer Abfrage gar keine Daten entsprechen, würde die erste Read-Anweisung schon False zurückgeben. Diese Vorgehensweise macht übrigens grundsätzlich dann Sinn, wenn Sie Daten lediglich auslesen müssen und dabei möglichst wenig Datenverkehr produzieren wollen.

Wenn Sie dieses Programm starten, zeigt es in etwa das folgende Ergebnis auf dem Bildschirm an:

Inhalt	Tabelle	Berater	(ersten 15 Datensätze, nach Nachnamen sortiert)	
10	Ademmer	Barbara	Kleine Gasse	06572 Wiesbaden
15	Braun	Hans	Alter Postweg	23034 Stirpe
12	Braun	Alfred	Parkstraße	01878 Wuppertal
8	Englisch	Barbara	Dorfstraße	44852 Unterschleißheim
6	Heckhuis	Momo	Kleine Gasse	81377 Lippetal
13	Hörstmann	Thomas	Crash Ave	36519 Wiesbaden
3	Meier	Uwe	Stadtstraße	36878 Liebenburg
9	Plenge	Anne	Crash Ave	48132 Straubing
5	Plenge	Barbara	Stadtstraße	12073 Lippetal
16	Rode	Lothar	Am Tor	31792 Braunschweig
2	Sonntag	José	Nordring	25019 Wuppertal
20	Thiemann	José	Windpockenallee	76122 München
11	Tiemann	Daja	Reiterweg	48938 Soest
14	Vielstedde	Anne	Main Road	78164 Liebenburg
4	Weichel	Axel	Crash Ave	93063 Lippetal

---

**TIPP:** In den Beispielen dieses Kapitels wird das `SqlConnection`-Objekt mit `Using` benutzt. Daher wird automatisch `Dispose` für das `SqlConnection`-Objekt aufgerufen, und diese Vorgehensweise für die meisten Datenbankanwendungen auch okay. In einem Datenbankprojekt, das wirklich performant sein muss, bietet es sich unter Umständen an, die Verbindung selber zu pflegen: Sie muss geöffnet werden, bevor Sie mit einem `DataReader` Daten lesen können und sollte auch nach dem Beenden des Lesevorgangs wieder geschlossen werden (aber das Verbindungsobjekt wird dabei nicht mit `Dispose` zum Entsorgen freigegeben).

Die `ExecuteReader` Methode unterstützt eine Überladung, bei der Sie beispielsweise angeben können, dass mit dem Schließen des `DataReader` auch die benutzte Verbindung geschlossen wird. Das Öffnen und Schließen einer Verbindung ist vergleichsweise »teuer«, was die Benutzung von Ressourcen auch auf dem SQL Server angeht. ADO.NET »poolt« daher standardmäßig Verbindungen, d.h. wenn Sie im Programm die Verbindung mit `Close` schließen, wird diese nicht endgültig geschlossen, sondern in den Pool zurückgestellt und beim nächsten `Open` wieder verwendet – was übrigens nur dann gilt, wenn die Verbindungszeichenfolge exakt (!) der vorherigen entsprochen hat. Dennoch wirkt es sich auf die Performance positiv aus, das Objekt nicht mit `Dispose` (oder durch die Verwendung mit `Using`) zu entsorgen, obgleich dieser Performancegewinn bei »normalen« Datenbankanwendungen nur marginal ist.

---

## Unverbundene Daten mit dem `DataTable`-Objekt verwalten

Wenn Sie Daten mit dem `DataReader` aus einer Datenbanktabelle lesen, hat das eigentlich nur einen Vorteil: Sie erhalten die Daten paketweise, verursachen damit wenig Datenverkehr und belasten den Hauptspeicher nur gering. Der große Nachteil: Daten sind mit dem `DataReader` äußerst umständlich zu handhaben, das haben Sie in dem kleinen Beispiel schon feststellen dürfen.

Mit dem `DataTable`-Objekt haben Sie es da schon viel einfacher. Es ist der erste Schritt zum Arbeiten mit nicht verbundenen Daten. Sie stellen sich das `DataTable`-Objekt am besten als eine Auflistung mit Elementen vor, deren Datenstruktur von den gespeicherten Daten der Datenbanktabelle abhängt (oder besser: des Resultsets, denn durch `SELECT`-Abfragen können auch Kombinationen mehrerer Tabellen oder nicht alle Felder einer Tabelle zurückgegeben werden). Die einzelnen Elemente einer `DataTable` können, wie bei anderen Auflistungen auch, durch die `Items`-Eigenschaft ihrer `Row`-Auflistung angesprochen werden, die die eigentlichen Daten als Auflistung aus so genannten `DataRow`-

Objekten erhält. Ein einzelnes Item ist also grundsätzlich vom Typ `DataRow`. Ein `DataRow`-Objekt erlaubt dann schließlich den Zugriff auf die eigentlichen Daten.

## Die DataAdapter-Klasse

Um ein `DataTable`-Objekt mit Inhalt zu füllen, benötigen Sie eine besondere Schnittstelle zur Datenbank, den so genannten *DataAdapter*.<sup>5</sup> Ein einzelnes *Command*-Objekt ist hier nicht mehr ausreichend, da es nur einen *DataReader* erzeugen kann, um die Schemainformationen einer Tabelle zu ermitteln und die Daten aufgrund der gegebenen Abfrage einzulesen. Doch Daten müssen gegebenenfalls auch aktualisiert oder neue Daten in die Datenbank eingefügt werden können. Und dafür sind mehrere Kommandos erforderlich, die der *DataAdapter* alle unter einem Hut vereint – wie das genau funktioniert, dazu später mehr.

Betrachten wir die Vorgehensweise, die ADO.NET verwendet, wenn Daten einer SQL Server-Tabelle mithilfe von `SqlDataAdapter` und `DataTable` abgerufen werden sollen:

- Das `SqlDataAdapter`-Objekt öffnet die Verbindung zur Datenbank im Bedarfsfall.
- Es weist das `SqlCommand`-Objekt an, das es aus seiner `SelectCommand`-Eigenschaft ermittelt, die Schemainformationen des Resultsets einzuholen, das zuvor beispielsweise durch eine `SELECT`-Anweisung abgefragt wurde.
- Es baut aus diesen Schemainformationen die Grundstruktur eines neuen `DataTable`-Objekt auf.
- Es verwendet anschließend ein `SqlDataReader`-Objekt, das durch das `SqlCommand`-Objekt angelegt wurde, um das `DataTable`-Objekt mit Daten zu füllen.
- Es schließt die Verbindung zur Datenbank.

Das Ergebnis: Die Daten stehen anschließend – losgelöst von der eigentlichen Datenbank – im Speicher des Computers, und sie sind durch die `Rows`-Eigenschaft (die die `DataRow`-Auflistung zur Verfügung stellt) des `DataTable`-Objektes abrufbar.

Diese Vorgehensweise wird durch das folgende Beispiel demonstriert, das eine Erweiterung des vorherigen Beispielprojektes darstellt. Mit diesem Programm können Sie also nicht nur die Struktur der Datenbank oder ihrer Tabellen einsehen, sondern auch Zugriff auf die gespeicherten Daten selbst nehmen.

Da Sie im Programm die `SELECT`-Anweisung frei verändern können, bietet es sich darüber hinaus auch zum Herumexperimentieren mit verschiedenen SQL-Abfragen an.

---

**BEGLEITDATEIEN:** Sie finden dieses Beispiel im Verzeichnis `.\VB 2005 - Entwicklerbuch\G - SmartClient\Kap32\DataTableDemo\`.

---

<sup>5</sup> Und wenn Sie jetzt genau hinschauen, stellen Sie fest, dass `DataTable` und `DataAdapter` auf unterschiedliche Weise hier im Text formatiert sind. `DataTable` ist datenbankunabhängig, deswegen ist es keine Klassenkategorie sondern eine konkrete Klasse und in Listingschrift formatiert. `DataAdapter` hingegen ist wie `Connection` und `DataReader` Daten-Provider abhängig – im Falle von SQL Server heißt die entsprechende Klasse `SqlDataAdapter` und ist deswegen nicht in Listingschrift sondern kursiv formatiert.

Wenn Sie dieses Beispielprojekt laden und starten, erlaubt es Ihnen, eine Abfrage zu formulieren, und das Ergebnis dieser Abfrage in Tabellenform im darunter liegenden DataGridView-Steuerelement sichtbar zu machen (siehe Abbildung 32.15).

Ein Blick auf das Listing offenbart, wie einfach der Einsatz mit dem DataTable-Objekt im Grunde genommen ist:

```
Imports System.Data.SqlClient

Public Class Form1

    Private Sub btnAusführen_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
        Handles btnAusführen.Click
        Dim locConnection As SqlConnection

        'Connection-Objekt holen, damit die Datenbankverbindung hergestellt werden kann
        locConnection = New SqlConnection _
            ("Data Source=.\SQLEXPRESS;Initial Catalog=Hecktick;Integrated Security=True")

        'Würden Sie den "gemischten Modus" verwenden, wäre Folgendes die richtige Wahl.
        'Aber aufgepasst: Das Passwort steht dann im Quellcode und kann leicht gefunden werden!
        'Eine Verschlüsselung des Passworts wenigstens mit einfachen Mitteln wäre dann angezeigt.
        'locConnection = New SqlConnection _
        '    ("Data Source=.\SQLEXPRESS;Initial Catalog=Hecktick;User ID=sa; Password=Irgendwas")

        Dim locAdapter As New SqlDataAdapter(txtSelectString.Text, locConnection)
        Dim locTable As New DataTable
        Try
            locAdapter.Fill(locTable)
        Catch ex As Exception
            MessageBox.Show(ex.Message, "Fehler bei der Befehlausführung:", _
                MessageBoxButtons.OK, MessageBoxIcon.Exclamation)
        End Try

        'Zur einfachen Darstellung der Daten reicht es auch ohne Hilfe einer
        'BindungSource aus, die DataTable der DataSource-Eigenschaft der DataGridView
        'zuzuweisen. In diesem Fall wird intern ein CurrencyManager-Objekt
        'für die Datenbindung anstelle der BindungSource eingerichtet.
        dgvData.DataSource = locTable
    End Sub
End Class
```

Die entscheidenden Zeilen in diesem Listing sind die fett hervorgehobenen. Beim Erstellen eines neuen SqlDataAdapter-Objektes übergeben Sie seinem Konstruktor sowohl die SELECT-Zeichenfolge als auch die Verbindung, die das Objekt verwenden soll, um einerseits den SELECT-Befehl abzusetzen und andererseits die Daten für die spätere Aufbereitung im DataTable-Objekt zu ermitteln. Sie erstellen anschließend ein neues DataTable-Objekt, und weisen den *DataAdapter* mit seiner *Fill*-Methode an, das DataTable-Objekt mit den abgefragten Daten zu füllen.

**Abbildung 32.15:** Mit diesem Beispiel, das den Einsatz der `DataTable`-Klasse demonstriert, können Sie beliebige Abfragen an die Beispieldatenbank stellen und die Resultsets in Tabellenform sichtbar machen

### Zugriff auf die einzelnen Datenfeldinhalte über das `DataRow`-Objekt

Um an die einzelnen Inhalte einer Datenzeile programmtechnisch zu gelangen (und diese nicht nur durch Zuweisung an die `DataSource`-Eigenschaft in einem `DataGridView`-Steuerelement darstellen zu lassen), bedienen Sie sich wie schon erwähnt der `DataRow`-Objekte einer `DataTable`, die Sie durch die `Rows`-Eigenschaft ermitteln können.

**Abbildung 32.16:** Per Doppelklick auf eine Tabellenzelle können Sie sich die Details eines Datensatzes in einem Nachrichtenfeld anzeigen lassen

Im Beispielprojekt haben Sie die Möglichkeit, sich durch Doppelklick auf eine Tabellenzelle die entsprechende Zeile en détail in einem Meldungsfeld ausgeben zu lassen.

Der entsprechende Code, der das erledigt, sieht folgendermaßen aus:

```
'Wird ausgelöst, wenn der Anwender auf eine Zelle doppelt klickt.
Private Sub dgvData_CellMouseDoubleClick(ByVal sender As System.Object,
                                         ByVal e As System.Windows.Forms.DataGridViewCellEventArgs) Handles dgvData.CellMouseDoubleClick

    'Die Datentabelle existiert noch, wir haben sie schließlich der
    'DataSource der DataGridView zugeordnet.
    Dim locDataTable As DataTable = DirectCast(dgvData.DataSource, DataTable)

    'Jetzt ermitteln wir die entsprechende DataRow der DataTable
    'Die folgende Methode funktioniert auch, wenn die Tabelle sortiert wurde,
    'da jede Zeile der DataGridView das "Original"-Objekt enthält,
    'aus der sie entstanden ist. In diesem Fall ist das ein DataRowView-Objekt,
    'mit dem die ursprüngliche Zeile der DataTable abrufbar ist.
    Dim loc0 As Object = dgvData.Rows(e.RowIndex)
    Dim locDataRowView As DataRowView = TryCast(dgvData.Rows(e.RowIndex).DataBoundItem, DataRowView)

    'War kein DataRowView-Objekt, dann beenden.
    If locDataRowView Is Nothing Then
        Return
    End If

    'Daraus können wir jetzt die eigentliche DataRow ableiten
    Dim locDataRow As DataRow = locDataRowView.Row

    'Und die Daten, die sich in der DataRow befinden,
    'geben wir anschließend in einer MessageBox aus:
    'Das StringBuilder-Objekt verwenden wir zum Zusammenbauen des Strings.
    Dim locSb As New System.Text.StringBuilder

    'Durch die einzelnen Spalten der Tabelle iterieren:
    For Each locSpalte As DataColumn In locDataRow.Table.Columns
        'String zusammenbauen. Erst den Spaltennamen,
        locSb.Append(locSpalte.ColumnName)
        'dann den Spaltentyp in Klammern,
        locSb.Append("(" & locSpalte.DataType.ToString & ")")
        'dann den eigentlichen Inhalt des Datenfeldes
        locSb.Append(locDataRow(locSpalte.ColumnName).ToString)
        'und schließlich einen Zeilenumbruch.
        locSb.Append(vbNewLine)
    Next

    'Alles in einer MessageBox ausgeben.
    MessageBox.Show(locSb.ToString, "Datenzeile enthält:", _
                    MessageBoxButtons.OK, MessageBoxIcon.Information)
End Sub
```

## Einige SELECT-Beispiele:

Sie können, wie zuvor schon angedeutet, die SELECT-Zeichenfolge des vorherigen Beispielprojektes durch Eingabe in der entsprechenden TextBox völlig frei gestalten, um beliebige Abfragen zu erstellen.

Viele Entwickler verstehen anfangs vergleichsweise schnell das Prinzip solcher Abfragen, doch Details machen Probleme. Zwar können ein paar Beispiele, die Sie an dieser Stelle finden, natürlich nicht eine vernünftige Einführung in die T-SQL-Sprache von SQL Server ersetzen, aber diese Beispiele helfen Ihnen auf jeden Fall über die ersten Hürden bei der Ausformulierung von Abfragen hinweg und ersparen Ihnen, so denke ich, einige Stunden Suchen im Internet.

Die folgende Bildreihe gibt Ihnen ein paar Einblicke in die Abfragen und ihre möglichen Ergebnisse (die Demodatenbank bei den Beispielen immer vorausgesetzt).

### **SELECT-Beispiel 1 – Bereichsabfrage zwischen zwei Datumswerten**

**Aufgabe:** Sie möchten alle Zeiten der Zeitentabelle ermitteln, die an einem bestimmten Buchungstagsdatum aufgetreten sind (im Beispiel am 15.3.2006):

#### **SELECT-Command-String:**

```
SELECT * FROM Zeiten WHERE StartZeit > CONVERT(datetime,'15.03.2006 00:00:00',104)
                           AND StartZeit < CONVERT(datetime,'15.03.2006 23:59:59',104)
```

**Anmerkung:** Da die Buchungsdaten unserer Beispieldatenbank auch den Datumsplatz enthalten, selektieren Sie die Startzeit eines Datums *zwischen* 0:00:00 Uhr und 23:59:59; nur mit der reinen Angabe des Datums würden Sie natürlich keinen einzigen Datensatz zurückerhalten, es sei denn die Buchung begäne exakt um 0:00 Uhr.

Und ebenfalls wichtig zu wissen: verwenden Sie die CONVERT-Funktion von T-SQL (wie im Beispiel zu sehen), um eine Datumskonstante, die als Zeichenkette vorliegt, in den eigentlichen Datumswert umzuwandeln. Geben Sie der CONVERT-Funktion als dritten Parameter dabei den Stilwert 104 an, der bestimmt, dass es sich um ein deutsches Datumsformat handelt. Auf diese Weise haben Sie keine Probleme mit kulturabhängigen Abfragen wie dieser auf Plattformen anderer als der deutschen Kultur. Ließen Sie eine implizite Konvertierung vornehmen, indem Sie die Zeichenkette ohne zu Hilfenahme der CONVERT-Funktion verwendeten, würde die Abfrage auf einem englischsprachigen System fehlschlagen, da hier einerseits Monat und Tag vertauscht sind und andererseits Zeiten nicht im 24-Stundenformat sondern mit AM/PM-Designatoren angegeben werden. Mit der Angabe von 104 zwingen Sie SQL Server quasi das deutsche Datumsformat für solche Abfrage auf.

## Aussehen des Resultsets:

IDZeiten	IDProjekt	IDBerater	Startzeit	Endzeit	ArbBeschreibung
823	7	41	15.03.2006 08:24	15.03.2006 11:55	Dokumentation p
824	8	41	15.03.2006 13:38	15.03.2006 15:14	Dokumentation ü
825	8	41	15.03.2006 15:14	15.03.2006 16:44	Dokumentation p
826	5	42	15.03.2006 08:09	15.03.2006 10:02	Dokumentation p
827	2	42	15.03.2006 10:02	15.03.2006 11:46	Programmieraufw
828	11	42	15.03.2006 12:06	15.03.2006 15:39	Programmieraufw
829	3	43	15.03.2006 08:33	15.03.2006 10:10	Dokumentation p
830	3	43	15.03.2006 10:10	15.03.2006 11:50	Dokumentation ü
831	4	43	15.03.2006 12:12	15.03.2006 15:32	Dokumentation Re

Abbildung 32.17: Resultset, das sich aus der Abfrage des entsprechenden Zeitbereichs ergibt

## SELECT-Beispiel 2 – Gezieltes Auswählen von Feldern einer Tabelle

**Aufgabe:** Sie möchten nicht alle Felder einer Tabelle in ein Resultset aufnehmen, sondern nur gezielte. Im folgenden Beispiel ermitteln Sie nur die Berater-ID der Mitarbeiter, die am 17.3.2006 vor 13:00 Uhr gearbeitet haben, die Projekt-ID der Projekte, an denen sie gearbeitet haben sowie die Anfangszeit.

**Anmerkung:** Wenn Sie bei SELECT-Abfragen die Feldnamen einzeln angeben (nicht über das Joker-Zeichen »\*«), werden nur die angegebenen Felder als Spalten im Resultset angelegt. Natürlich können Sie dennoch alle anderen Felder für Selektierungen (mit WHERE) und Sortierungen (mit ORDER BY) verwenden.

### SELECT-Command-String:

```
SELECT [IDBerater], [IDProjekt], [StartZeit] FROM [Zeiten]
    WHERE StartZeit > CONVERT(datetime,'17.03.2006 00:00:00',104)
        AND StartZeit <= CONVERT(datetime,'17.03.2006 13:00:00',104)
```

## Aussehen des Resultsets:

IDBerater	IDProjekt	StartZeit
41	11	17.03.2006 07:13
41	9	17.03.2006 10:45
43	8	17.03.2006 07:31
43	5	17.03.2006 11:49
44	8	17.03.2006 08:39
45	4	17.03.2006 08:59
45	4	17.03.2006 10:42
46	3	17.03.2006 08:06
46	4	17.03.2006 12:19
47	9	17.03.2006 09:21

**Abbildung 32.18:** Resultset, das sich aus der Abfrage des entsprechenden Zeitbereichs ergibt, wobei nur die angegebenen Spalten in das Resultset einfließen

## SELECT-Beispiel 3 – Verknüpfung mehrere Tabellen mit Join-Abfragen

**Aufgabe:** Sie möchten eine Liste mit Personalnummer, Vornamen und Nachnamen der Mitarbeiter ermitteln, die in der Woche vom 15.3. bis zum 20.3.2004 vor 8 Uhr morgens zu arbeiten begonnen haben. Die Anfangszeit soll ebenfalls mit in der Liste enthalten sein. Geordnet werden soll die Liste nach Mitarbeiternachnamen.

**Anmerkung:** Mit *Join-Abfragen*, die mehrere Tabellen in der *SELECT*-Abfrage beinhalten, können Sie Resultsets erzeugen, die Daten aus verschiedenen Tabellen kombinieren.

### SELECT-Command-String:

```

SELECT      Zeiten.Startzeit, Zeiten.ArbBeschreibung, Berater.Nachname, Berater.Vorname,
            Zeiten.IDBerater

FROM        Zeiten INNER JOIN
            Berater ON Zeiten.IDBerater = Berater.IDBerater
WHERE       (Zeiten.Startzeit > CONVERT(datetime, '17.03.2006 00:00:00', 104))
            AND (Zeiten.Startzeit <= CONVERT(datetime, '17.03.2006 13:00:00', 104))

ORDER BY    Berater.Nachname

```

## Aussehen des Resultsets:

IDBerater	Startzeit	Arbeitsbeschreibung	Nachname	Vorname
45	17.03.2006 08:59	Programmieraufw...	Braun	Axel
45	17.03.2006 10:42	Dokumentation p...	Braun	Axel
56	17.03.2006 07:30	Dokumentation p...	Braun	Katrin
56	17.03.2006 12:41	Programmieraufw...	Braun	Katrin
48	17.03.2006 07:05	Programmieraufw...	Englisch	Katrin
48	17.03.2006 11:35	Dokumentation ü...	Englisch	Katrin
50	17.03.2006 07:12	Programmieraufw...	Heckhuis	Britta
50	17.03.2006 11:53	Dokumentation ü...	Heckhuis	Britta
51	17.03.2006 08:34	Organisation: Re...	Hoffmann	Katrin
51	17.03.2006 11:56	Organisation: Re...	Hoffmann	Katrin

**Abbildung 32.19:** Resultset, das sich aus der Abfrage des entsprechenden Zeitbereichs ergibt, wobei nur die angegebenen Spalten in das Resultset einfließen; zwei Tabellen fließen in die Abfrage ein, die mit einer *JOIN*-Abfrage verknüpft sind

---

**HINWEIS:** Feldnamen in eckigen Klammern zu platzieren ist bei SELECT-Anweisungen nicht unbedingt notwendig. Auf diese Weise verhindern Sie aber, dass der SQL-Parser der jeweiligen Datenbank-Engine Feldnamen von SQL-Befehlen nicht unterscheiden kann oder dass die Namen aufgrund von Sonderzeichen wie Leerzeichen in »My Customer« nicht zu erkennen sind. Um ganz auf der sichereren Seite zu sein, sollten Sie nicht nur Feldnamen in eckigen Klammern einschließen, sondern auch nach Möglichkeit auf Feldnamen verzichten, die SQL-Anweisungen darstellen. Gerade der Feldname *Name* ist bei vielen fehlschlagenden SQL-Anweisungen Grund für stundenlange Fehlersuche – deswegen heißt das Feld in der Beispieldatenbank auch *Nachname*.

---

## Ändern und Ergänzen von Daten in Datentabellen

Ich möchte Ihnen nicht die Illusionen rauben, aber wenn Sie in Ihrer längeren Programmierkarriere schon mit Vorläufern von ADO.NET (ADO, DAO, RDO) gearbeitet haben, dann wird Ihnen die folgende Behauptung vielleicht sehr merkwürdig und unglaublich vorkommen:

»Es gibt nur jeweils eine Möglichkeit, Daten in einer Datenbanktabelle von außen zu verändern oder eine Tabelle um Daten zu ergänzen, nämlich mit der INSERT-, DELETE- und der UPDATE-SQL-Anweisung.«<sup>6</sup>

---

<sup>6</sup> Bei SQL Server 2005 bzw. diesem im Zusammenspiel mit ADO.NET 2.0 ist das nicht mehr ganz korrekt, da es auch die Möglichkeit gibt, große Massen von Daten mit so genannten Bulk-Kopierfunktionen in eine SQL Server-Datenbank zu »pumpen«. Doch dieses Thema würde in diesem Umfeld zu weit führen.

»Moment«, werden Sie vielleicht jetzt sagen, »UPDATE schön und gut, aber wieso SQL? Es gab doch schon zu ADO und DAO Recordset-Objekte, mit denen das Ergänzen und Verändern von Daten einfach mit Edit- und Update der entsprechenden Recordset-Objekte funktionierte oder nicht?«.

Sie haben Recht. Und auch nicht. Denn was vielen Programmierern immer verborgen blieb, war die Weise, wie Daten in einer Datenbanktabelle tatsächlich ergänzt oder verändert wurden, und das folgende Beispiel gibt Ihnen einen kleinen Vorgeschmack auf die wirkliche Realität.<sup>7</sup>

---

**BEGLEITDATEIEN:** Sie finden dieses Beispiel im Verzeichnis .\VB 2005 - Entwicklerbuch\G - SmartClient\Kap32\Insert Update manuell\.

---

Dieses Beispielprogramm ist eine Konsolenanwendung, also – dank wenig Overhead – relativ leicht zu durchschauen. Um nachvollziehen zu können, was beim Ablauf des Programms passiert, werden Sie eine Verbindung vom Server-Explorer zur Beispieldatenbank benötigen. Wie Sie eine Verbindung herstellen können, erfahren Sie im ► Abschnitt »Einsehen von Daten mit dem Server-Explorer« ab Seite 995.

Wenn Sie das Programm starten, sehen Sie zunächst die folgende Bildschirmausgabe (hier zum besseren Verständnis ein wenig leserlicher formatiert)

```
INSERT INTO [Projekte] ([Projektname],[Projektbeschreibung],[Startzeitpunkt],  
[Endzeitpunkt], [Ausführungsort])  
VALUES ('Orcas','(VS 2007) Schreiben des Entwicklerbuchs','01.02.2007','31.12.2007','Büro')  
  
INSERT INTO [Projekte] ([Projektname],[Projektbeschreibung],[Startzeitpunkt],  
[Endzeitpunkt], [Ausführungsort])  
VALUES ('Katmai','(SQL Server 2005) Schreiben des Crash-Kurs','01.03.2007','31.01.2008','Büro')
```

Betrachten Sie die hinzugefügten Daten nun im Server-Explorer  
Drücken Sie anschließend Return

Wechseln Sie nun, **ohne** das Programm zunächst zu beenden, zum Server-Explorer. Falls der Server-Explorer im Laufzeitmodus nicht angezeigt werden sollte, aktivieren Sie ihn, indem Sie aus dem Menü *Ansicht* den Menüpunkt *Server-Explorer* auswählen.

Wenn Sie die Verbindung zur Demodatenbank hergestellt haben, öffnen Sie alle Zweige und doppelklicken Sie auf die Tabelle *Projekte*. Sie sehen die neuen Datensätze nun in der Tabelle, etwa wie in Abbildung 32.20 zu sehen.

---

<sup>7</sup> So viel zu roter und blauer Pille ...

Projekte: Abfra...press.Hecktick] mdlMain.vb						
IDProjekte	Projektname	Projektbeschreibung	Startzeitpunkt	Endzeitpunkt	Ausführungsort	
1	Visual Basic 200...	Erstellung eines 1.100 Seiten Buches für Mic...	01.02.2005 00:00:00	15.03.2006 00:00:00	Office	
2	Adresso.NET	Erstellung einer Client-/Server-fähigen Adre...	24.12.2005 00:00:00	30.06.2006 00:00:00	Office	
3	ASP.NET Fachle...	Fachlektorat von Holger Schwichtenbergs A...	24.12.2005 00:00:00	30.06.2006 00:00:00	Office	
4	HeckTick	Erstellung einer Client-/Server-fähigen Soft...	24.12.2005 00:00:00	30.06.2006 00:00:00	Vor Ort	
5	Visual Basic 200...	Erstellung eines 1.100 Seiten Buches für Mic...	01.02.2005 00:00:00	15.03.2006 00:00:00	Office	
6	Adresso.NET	Erstellung einer Client-/Server-fähigen Adre...	24.12.2005 00:00:00	30.06.2006 00:00:00	Office	
7	ASP.NET Fachle...	Fachlektorat von Holger Schwichtenbergs A...	24.12.2005 00:00:00	30.06.2006 00:00:00	Office	
8	HeckTick	Erstellung einer Client-/Server-fähigen Soft...	24.12.2005 00:00:00	30.06.2006 00:00:00	Vor Ort	
9	Visual Basic 200...	Erstellung eines 1.100 Seiten Buches für Mic...	01.02.2005 00:00:00	15.03.2006 00:00:00	Office	
10	Adresso.NET	Erstellung einer Client-/Server-fähigen Adre...	24.12.2005 00:00:00	30.06.2006 00:00:00	Office	
11	ASP.NET Fachle...	Fachlektorat von Holger Schwichtenbergs A...	24.12.2005 00:00:00	30.06.2006 00:00:00	Office	
12	HeckTick	Erstellung einer Client-/Server-fähigen Soft...	24.12.2005 00:00:00	30.06.2006 00:00:00	Vor Ort	
18	Orcas	(VS 2007) Schreiben des Entwicklerbuchs	01.02.2007 00:00:00	31.12.2007 00:00:00	Büro	
▶ 19	Katmai	(SQL Server 2005) Schreiben des Crash-Kurs	01.03.2007 00:00:00	31.01.2008 00:00:00	Büro	
*	NULL	NULL	NULL	NULL	NULL	

**Abbildung 32.20:** Mit den zwei *INSERT*-SQL-Anweisungen wurden die Datensätze der Tabelle hinzugefügt

Wechseln Sie anschließend zurück zum Konsolenfenster, und drücken Sie **Return**. Die Bildschirmausgabe wird anschließend um die folgenden Zeilen ergänzt:

```
UPDATE [Projekte] SET [Projektbeschreibung]='(SQL Server 2007) Schreiben des Crash-Kurses',
[Startzeitpunkt]='01.02.2007',
[Endzeitpunkt]='31.10.2007',
[Ausführungsort]='Ruprechts Büro'
WHERE Projektname='Katmai'
```

Betrachten Sie die geänderten Daten nun im Server-Explorer  
Drücken Sie anschließend Return

Wenn Sie anschließend zurück zur Datenansicht der Projekttabelle wechseln, dort das Kontextmenü öffnen und den Eintrag *Ausführen* wählen, aktualisiert der Server-Explorer das angezeigte Resultset. Sie sehen dann, dass sich der letzte Datensatz in verschiedenen Feldern geändert hat.

## Was machte das »alte« ADO?

Wie war das jetzt mit ADO und dem überaus einfachen Updaten von Daten in Recordset-Objekten? Eine simple SELECT-Abfrage genügte doch damals, um das Resultset einer Abfrage zu erhalten. Anschließend konnte man einfach durch Ändern der Feldeigenschaften oder AddNew-Befehle Änderungen und Ergänzungen an die Datenbank zurück übermitteln.

Tatsache ist: Auch das alte ADO hat mit UPDATE- und INSERT-SQL-Anweisungen gearbeitet, und zwar genauso, wie Sie es im vorherigen Beispiel beobachten konnten. Auf Grund des Resultset-Schemas, das ADO im Rahmen einer SELECT-SQL-Anweisung erhalten konnte, hat ADO hinter den Kulissen die zum Aktualisieren oder Ergänzen notwendigen SQL-Befehle selbst zusammengestellt und anschließend verwendet. Diese Vorgehensweise hatte für den Programmierer zwar den Vorteil, sehr schnell zum Ergebnis zu kommen, allerdings waren die Abfragen, da sie keinen Eingriff von außen zuließen, starr und unflexibel.

Das hat sich mit ADO.NET grundlegend geändert.

## Verwenden von DataAdapter, DataTable und Command-Objekten zur Übermittlung von Aktualisierungen

Sie werden mir zustimmen, dass die Vorgehensweise aus dem letzten Beispiel zum Hinzufügen von Daten zu einer Tabelle bzw. zum Ändern von Daten in einer Tabelle zwar zum Ziel führt, aber viel zu aufwändig ist, um für größere Vorhaben geeignet zu sein. Es wäre einfach undenkbar, müsste man, nachdem man beispielsweise Änderungen an einem `DataTable`-Objekt selber vorgenommen hat, danach zusätzlich noch dafür sorgen, dass das eigene Programm entsprechende SQL-Anweisungen generiert, die Tabelle durchsucht und auf diese Weise Änderungen an die Datenbank übermittelt.

Prinzipiell funktioniert die Vorgehensweise zum Übermitteln von Änderungen zwar genauso, doch Sie müssen sie nicht selber machen. `DataAdapter` und `DataTable` spielen hier in einem Team zusammen, und Sie geben nur die groben Rahmenparameter vor, damit Übermittlungen von Änderungen an Datentabellen erfolgreich zum Ziel führen.

Die Kunst, die ADO.NET dabei zu finden versucht, ist, dem Entwickler auf der einen Seite zwar die Änderungen, die er über Vorgaben im `DataTable`-Objekt macht, möglichst automatisch in der Datenbank via `DELETE`-, `INSERT`- bzw. `UPDATE`-Anweisungen umzusetzen, ihn aber dennoch nicht durch zu starre Vorgaben einzuschränken.

Das heißt für das, was ADO.NET leisten muss: Der Entwickler sollte Änderungen an den Daten bequem in den `DataRow`-Objekten einer `DataTable` vornehmen können. Nachdem eine `DataTable` durch `Fill` oder `FillSchema` gefüllt bzw. vorbereitet wurde, kann er mit Modifizierungen beginnen.

- Wenn Zeilen zur `DataTable` hinzugefügt worden sind, muss der `DataAdapter` diese Zeilen in einer `DataTable` als neu erkennen und automatisch eine entsprechende `INSERT`-Anweisung verwenden, um die Zeilen in der Datenbanktabelle einzufügen.
- Wenn Daten einer `DataTable` geändert wurden, muss der `DataAdapter` die geänderten Zeilen (`DataRow`-Objekte) ebenfalls erkennen. Er muss aber auch die ursprünglichen Daten noch herausfinden können (die Daten der jeweiligen `DataRow` *vor* der Änderung), damit diese im `WHERE`-Teil einer `UPDATE`-Anweisung eingesetzt werden können. Nur so kann der `DataAdapter` die Datensätze in der Datenbank überhaupt finden, die mit `UPDATE` geändert werden sollen (siehe 2. Bildschirmausgabe des vorherigen Beispiels)
- Wenn Zeilen einer `DataTable` gelöscht wurden, muss der `DataAdapter` dennoch noch Zugriff auf die gelöschten `DataRow`-Objekte der `DataTable` haben, damit er eine `DELETE`-Anweisung verwenden und mit entsprechenden Parametern versehen kann, die die Zeilen in der Tabelle löschen.

Das alte ADO sorgte dafür, dass alle SQL-Anweisungen zur Datenänderung in Datenbanktabellen hinter den Kulissen erstellt wurden. Wollte ein erfahrener Datenbankspezialist selbst Hand anlegen und optimierte Aktualisierungslogiken oder gar – für SQL Server – gespeicherte Prozeduren entwerfen, schaute er in die Röhre.

Der `DataAdapter` von ADO.NET schlägt hier eine Brücke zwischen Flexibilität und einfacher Handhabung: Sie können die Abfragelogik auf der einen Seite zwar komplett selbst implementieren, kön-

nen aber dennoch ein `DataTable`<sup>8</sup>-Objekt verwenden, um Änderungen an Resultsets auf einfachste Weise durchzuführen und sie zur Datenbank übertragen.

Die Gratwanderung, die ADO.NET – oder besser: der `DataAdapter` – beschreiten muss, ist, dabei ein halbwegs handhabbares Verfahren zur Verfügung zu stellen, mit dem `INSERT`-, `UPDATE`- und `DELETE`-Anweisungen parametrisiert werden können. Anstatt also eine vollständige Logik vorzugeben, etwa wie,

```
UPDATE [Projekte] SET [Projektname]='Katmai'  
    [Projektbeschreibung]=(SQL Server 2007) Schreiben des Crash-Kurses',  
    [Startzeitpunkt]='01.02.2007',  
    [Endzeitpunkt]='31.10.2007',  
    [Ausführungsort]='Ruprechts Büro'  
WHERE Projektname='Katmai'
```

ergibt es mehr Sinn, die eigentlichen Parameter zunächst auszulassen, etwa wie im folgenden Beispiel:

```
UPDATE [Projekte] SET [Projektname]=@Var1,  
    [Projektbeschreibung]=@Var2,  
    [Startzeitpunkt]=@Var3,  
    [Endzeitpunkt]=@Var4,  
    [Ausführungsort]=@Var5  
WHERE ([Projektname]=@Var6) AND ([Projektbeschreibung]=@Var7) AND  
    ([Startzeitpunkt]=@Var8) AND ([Endzeitpunkt]=@Var9) AND  
    ([Ausführungsort]=@Var10)
```

Diese Abfrage ist nun von den eigentlichen Parametern unabhängig (auch wenn sie in dieser Form zunächst nicht funktionieren würde) und kann an mehreren Stellen in einer Anwendung für unterschiedliche Modifikationen und nicht nur eine einzige Modifikation verwendet werden. Voraussetzung dafür ist natürlich, dass es eine Instanz gibt, die aus den einzelnen mit @ beginnenden Platzhaltervariablen wieder richtige Werte macht, die es auf der einen Seite abzufragen (Suchkriterien für den ursprünglichen Datensatz werden hinter `WHERE` angegeben, um den zu ändernden Datensatz in der Tabelle zu finden) und auf der anderen Seite zu aktualisieren gilt (der `SET`-Teil bestimmt die neuen Werte des Datensatzes). Diese Instanz muss aber nicht nur die Werte einsetzen, sie muss dazu auch wissen, welchen Typs die Werte sein müssen, die @-Variablen in der allgemeingültigen Abfrage ersetzen sollen.

Das `Command`-Objekt, das einen solchen Aktualisierungs-SQL-Befehl kapselt, stellt aus diesem Grund ein `Parameters`-Array zur Verfügung, das die eigentlichen Parameter in der Reihenfolge ihres Auftretens sozusagen »ver-typ«. Dabei wird ebenfalls festgehalten, welche Parameter für eine neue Wertbestimmung dienen (im Beispiel die ersten fünf @-Variablen) und welche für die Datensatzidentifizierung zuständig sind (die letzten @-Variablen).

Damit ist die Grundvoraussetzung geschaffen, um eine Verbindung zwischen völlig von der Datenbank losgelösten Datenzeilen und der Datenbank herzustellen. Wenn die Datenzeilen, also die `DataRow`-Objekte einer `DataTable`, anschließend aktualisiert, gelöscht oder ergänzt werden, greift der

---

<sup>8</sup> Das `DataTable`-Objekt ist nicht das einzige Objekt. Es ist aber das wohl am häufigsten verwendete (wenn auch in vielen Fällen nur indirekt über das `DataSet`-Objekt). Im Rahmen dieses Buches möchte ich mich auf das `DataTable`-Objekt beschränken.

*DataAdapter* auf insgesamt drei verschiedene *Command*-Objekte zurück, um die Änderungen an die Datenbank zu übermitteln. Er verfährt dabei folgendermaßen:

- Wenn er Zeilen in der *DataTable* findet, die verändert worden sind, greift er auf den *UpdateCommand* des *DataAdapter* zurück, um zunächst einmal die allgemeingültige Aktualisierungslogik (*UPDATE [Tabellenname] SET ... WHERE ...*) zu ermitteln. Er setzt dann – und dabei nimmt er die Parameters-Auflistung des *Commands* zu Hilfe – anstelle der @-Variablen die neuen Werte der *DataRow* ein. Um den zu verändernden Datensatz zu finden, ersetzt er ferner die hinter der *WHERE*-Klausel stehenden Fragezeichen durch die Originalwerte, die in einem *DataRow*-Objekt erhalten bleiben, auch wenn neue Werte zugewiesen wurden.<sup>9</sup>
- Findet er Zeilen in der *DataTable*, die neu hinzugekommen sind, greift er auf die *InsertCommand*-Eigenschaft des *DataAdapters* zurück, um die allgemeingültige Aktualisierungslogik zu ermitteln (*INSERT INTO ...*). Für die Parameter verfährt er anschließend wie bei der Erstellung des *UPDATE*-Befehls, mit dem Unterschied, dass der Part für die Originalwertbehandlung nicht zur Anwendung kommt (wozu auch – bei neuen Datenzeilen gibt es keine alten Originalwerte).
- Für gelöschte Zeilen in der *DataTable* greift er auf die *DeleteCommand*-Eigenschaft des *DataAdapter* zurück und verfährt für die weitere Aufbereitung der *DELETE*-SQL-Anweisung auf gleiche Weise.

Schauen wir uns an, wie sich die Anwendung dieser Verfahren in der Praxis darstellt.

---

**BEGLEITDATEIEN:** Sie finden das folgende Beispiel im Verzeichnis .\VB 2005 - Entwicklerbuch\G - Smart-Client\Kap32\Insert Update DataAdapter.

---

Das Programm macht exakt das Gleiche wie das vorherige Beispiel – es verwendet nur die gerade vorgestellte Vorgehensweise.

---

**HINWEIS:** Bitte achten Sie darauf, dass Sie vor dem Programmstart die vom vorherigen Beispiel hinzugefügten Datensätze wieder löschen, damit das Beispiel reibungslos funktionieren kann. Verwenden Sie auch dafür am besten wieder den Server-Explorer.

---

```
Imports System.Data.SqlClient

Module md1Main

    Sub Main()

        Dim locConnection As SqlConnection

        'Connection-Objekt holen, damit die Datenbankverbindung hergestellt werden kann
        locConnection = New SqlConnection("Data Source=.\SQLEXPRESS;Initial Catalog=Hecktick;Integrated Security=True")

        'Verbindung öffnen
        locConnection.Open()
```

---

<sup>9</sup> Sie können den ursprünglichen Wert eines *DataRow*-Feldinhaltes abrufen, indem Sie der *Item*-Eigenschaft nicht nur einen Index in Form des Feldnamens oder einer Ordnungszahl übergeben, sondern auch die Konstante *DataRowVersion.Original*.

```

'Damit sorgen wir dafür, dass die Verbindung wieder geschlossen wird,
'wenn locConnection aus dem Using-Scope herausläuft.
Using locConnection

    'Alle Commands im DataAdapter definieren
    Dim locAdapter As SqlDataAdapter = DataAdapterVorbereiten(locConnection)

    'Neue Datentable
    Dim locDataTable As New DataTable

    'Könnten wir selber definieren, aber so geht's schneller.
    locAdapter.FillSchema(locDataTable, SchemaType.Source)

    'Neue Datenzeile
    Dim locDataRow As DataRow = locDataTable.NewRow()

    'Entsprechend des Datenschemas der Tabelle mit Datenfüllen.
    '(IDProjekte wird von Sql Server selbst geschrieben, da es ein
    ' AutoIncrement-Feld ist. Deswegen übergeben wir hier 'Nothing')
    locDataRow.ItemArray = New Object() {Nothing, "Orcas", " (VS 2007) Schreiben des Entwicklerbuchs", _
        #2/1/2007#, #12/31/2007#, "Büro"}
    'Die Zeile der Tabelle hinzufügen
    locDataTable.Rows.Add(locDataRow)

    'Das Gleiche mit einer weiteren Zeile
    locDataRow = locDataTable.NewRow()
    locDataRow.ItemArray = New Object() {Nothing, "Katmai", "(SQL Server 2005) Schreiben des Crash-Kurs", _
        #3/1/2007#, #1/31/2008#, "Büro"}
    locDataTable.Rows.Add(locDataRow)

    'Und nun die neue Tabelle mit der Datenbank synchronisieren.
    'Hier wird SQL-INSERT zweimal hintereinander ausgeführt.
    locAdapter.Update(locDataTable)

    Console.WriteLine("Betrachten Sie die hinzugefügten Daten nun im Server-Explorer")
    Console.WriteLine("Drücken Sie anschließend Return")
    Console.WriteLine()
    Console.ReadLine()

    'DataTable neu einlesen
    locDataTable.Clear()
    locAdapter.Fill(locDataTable)

    'Jetzt verwenden wir eine View, um die 'Katmai'-Zeile zu ermitteln.
    'Das ist die, die wir als letztes geändert haben!

    Dim locDataView As New DataView(locDataTable, "Projektname='Katmai'", _
        "Projektname", DataViewRowState.CurrentRows)

    'Eigentlich können wir nur genau eines zurückbekommen.
    locDataRow = locDataView(0).Row

```

```

'Daten ändern.
locDataRow.ItemArray = New Object() {Nothing, "Katmai", "(SQL Server 2007) Schreiben des Crash-Kurses",
#2/1/2007#, #10/31/2007#, "Ruprechts Büro"}

'Änderungen zurückschreiben.
locAdapter.Update(locDataTable)

Console.WriteLine("Betrachten Sie die geänderten Daten nun im Server-Explorer")
Console.WriteLine("Drücken Sie anschließend Return")
Console.ReadLine()
End Using

End Sub

```

Sie sehen, dass die Daten über die `DataTable` sehr viel besser zu handhaben sind. Das Hinzufügen geht genau wie das Ändern von Daten recht einfach von der Hand.

### **Filtern und Sortieren von Zeilen einer `DataTable` mit der `DataView`-Klasse**

Eine kurze Anmerkung möchte ich aber an dieser Stelle über das Ermitteln des `DataRow`-Objektes verlieren, an dem anschließend das Ändern von Daten demonstriert wird. Zu diesem Zweck verwendet das Beispiel nämlich die so genannte `DataView`-Klasse, die Sie sich wie eine Abfrage-Engine für `DataTable`-Objekte vorstellen können. So, wie Sie mit der `SELECT`-Anweisung nach bestimmten Daten in einer Datenbank direkt selektieren und sortieren können, diese also auch zusätzlich filtern können, verwenden Sie eine `DataView` als Filter- oder Sortierhilfe für bereits mit Daten gefüllte `DataTable`-Objekte. Die Filter- und Sortierungsfunktionen von `DataView` sind dabei völlig eigenständig – was auch logisch ist, da ein `DataTable`-Objekt seine einzelnen Datenzeilen auch völlig von der Datenbank getrennt verwaltet. Ein Zugriff auf den »Datenbankmotor« wäre einem `DataView`-Objekt schon dadurch gar nicht möglich.

Sie übergeben einer `DataView`-Klasse im Konstruktor die zu bearbeitende `DataTable` und optional eine Zeichenkette mit einer Filterdefinition sowie eine weitere, die den Namen der Spalte der `DataTable` enthält, nach der die Daten sortiert werden sollen.

Die `DataView` packt beim Filtern oder Sortieren die ursprüngliche Tabelle aber gar nicht an; vielmehr legt Sie scheinbar eine Kopie der Ursprungstabelle an, die aus `DataViewRow`-Objekten bestehen, von denen aber jedes einzelne quasi nur »virtuell« existiert. Ein `DataViewRow`-Objekt ist nämlich im Grunde genommen nur ein einsortierter (oder bzw. und gefilterter) Zeiger auf die originalen, unangetasteten Datenzeilen der `DataTable`.

Über den Indexer der `DataView` können Sie anschließend auf einzelne `DataViewRow`-Objekte zugreifen, mit deren jeweiliger `Row`-Eigenschaft Sie die ursprüngliche `DataRow` der zuvor übergebenden `DataTable` ermitteln können.

Im Beispiel nutzen wir ein `DataView`-Objekt, um die Datenzeile zu ermitteln, auf die die Filterbedingung `Projektname='Katmai'` zutrifft; das ist nämlich exakt der Projektname der Datenzeile, die wir zuvor der Projekttabelle hinzugefügt haben.

## Einrichten der Command-Objekte des Beispiels

Der übrige Code befindet sich in einer einzigen Prozedur, die, wie in der Einleitung dieses Abschnittes beschrieben, dafür zuständig ist, die benötigten *Command*-Objekte für die entsprechenden Aufgaben (Aktualisieren, Einfügen und Löschen von Datensätzen) einzurichten.

Sie sehen auf den folgenden drei Codeseiten dabei nicht nur, wie das Einrichten der *Command*-Zeichenketten für die Steuerung der SQL-Befehle an sich funktioniert. Sie erkennen auch das Zusammenspiel mit der Parameters-Auflistung, auf die der *DataAdapter* später, beim Einsetzen der »wirklichen« Werte zurückgreift.

Und Sie sehen noch etwas: Wie vergleichsweise aufwändig die Einrichtung dieser *Command*-Objekte ist. Doch keine Angst: Visual Basic 2005 hält zwei weitere Techniken für Sie bereit, mit denen Sie diesen Aufwand minimieren können, aber ohne etwas von der Flexibilität dieses Konzeptes aufgeben zu müssen! Wie das funktioniert, zeigen die nächsten beiden Abschnitte.

```
Private Function DataAdapterVorbereiten(ByVal Conn As SqlConnection) As SqlDataAdapter

    Dim locDa As New SqlDataAdapter

    locDa = New System.Data.SqlClient.SqlDataAdapter

    locDa.SelectCommand = New SqlCommand("SELECT * FROM [Projekte]")
    locDa.SelectCommand.Connection = Conn

    locDa.DeleteCommand = New System.Data.SqlClient.SqlCommand
    locDa.DeleteCommand.Connection = Conn
    locDa.DeleteCommand.CommandText = "DELETE FROM [dbo].[Projekte] WHERE " &
        "(([IDProjekte] = @Original_IDProjekte) AND ([Projektname]" &
        "= @Original_Projektname) AND (@IsNull_Projektbeschreibung = 1 AND " &
        "[Projektbeschreibung] IS NULL) OR ([Projektbeschreibung] = @Original_Projektbeschreibung))" &
        " AND ([Startzeitpunkt] = @Original_Startzeitpunkt) AND ([Endzeitpunkt] " &
        "= @Original_Endzeitpunkt) AND (@IsNull_Ausfuehrungsort = 1 AND [Ausfuehrungsort] " &
        "IS NULL) OR ([Ausfuehrungsort] = @Original_Ausfuehrungsort))"
    locDa.DeleteCommand.CommandType = System.Data.CommandType.Text
    locDa.DeleteCommand.Parameters.Add( _
        New System.Data.SqlClient.SqlParameter("@Original_IDProjekte", System.Data.SqlDbType.Int, 0, _
        System.Data.ParameterDirection.Input, 0, 0, _
        "IDProjekte", System.Data.DataRowVersion.Original, False, Nothing, "", "", ""))
    locDa.DeleteCommand.Parameters.Add( _
        New System.Data.SqlClient.SqlParameter("@Original_Projektname", System.Data.SqlDbType.NVarChar, 0,
        System.Data.ParameterDirection.Input, 0, 0, _
        "Projektname", System.Data.DataRowVersion.Original, False, Nothing, "", "", ""))
    .
    .
    . ' Rest aus Platzgründen weggelassen
    .
    .
    locDa.InsertCommand = New System.Data.SqlClient.SqlCommand
    locDa.InsertCommand.Connection = Conn
    locDa.InsertCommand.CommandText = "INSERT INTO [dbo].[Projekte] ([Projektname], [Projektbeschreibung], " &
        "[Startzeitpunkt], [Endzeitpunkt], [Ausfuehrungsort]) VALUES (@Projektname, " &
        "@Projektbeschreibung, @Startzeitpunkt, @Endzeitpunkt, @Ausfuehrungsort);"
```

```

    locDa.InsertCommand.CommandType = System.Data.CommandType.Text
    locDa.InsertCommand.Parameters.Add(
        New System.Data.SqlClient.SqlParameter("@Projektname", System.Data.SqlDbType.NVarChar, 0, _
        System.Data.ParameterDirection.Input, 0, 0, _
        "Projektname", System.Data.DataRowVersion.Current, False, Nothing, "", "", ""))
    locDa.InsertCommand.Parameters.Add(
        New System.Data.SqlClient.SqlParameter("@Projektbeschreibung", System.Data.SqlDbType.NVarChar, 0, _
        System.Data.ParameterDirection.Input, 0, 0, _
        "Projektbeschreibung", System.Data.DataRowVersion.Current, False, Nothing, "", "", ""))
    .
    .
    .
    ' Rest aus Platzgründen weggelassen
    .
    .
    locDa.UpdateCommand = New System.Data.SqlClient.SqlCommand
    locDa.UpdateCommand.Connection = Conn
    locDa.UpdateCommand.CommandText = "UPDATE [dbo].[Projekte] " & _
        "SET [Projektname] = @Projektname, [Projektbeschreibung] =" & _
        "@Projektbeschreibung, [Startzeitpunkt] = @Startzeitpunkt, [Endzeitpunkt] = " & _
        "@Endzeitpunkt, [Ausführungsort] = @Ausführungsort WHERE (([IDProjekte] = " & _
        "@Original_IDProjekte) AND ([Projektname] = @Original_Projektname) AND " & _
        "((@IsNull_Projektbeschreibung = 1 AND [Projektbeschreibung] IS NULL) OR ([Projektbeschreibung] " & _
        "= @Original_Projektbeschreibung)) AND ([Startzeitpunkt] = @Original_Startzeitpunkt) " & _
        "AND ([Endzeitpunkt] = @Original_Endzeitpunkt) AND (@IsNull_Ausführungsort = 1 AND " & _
        "[Ausführungsort] IS NULL) OR ([Ausführungsort] = @Original_Ausführungsort));"
    locDa.UpdateCommand.CommandType = System.Data.CommandType.Text
    locDa.UpdateCommand.Parameters.Add(
        New System.Data.SqlClient.SqlParameter("@Projektname", System.Data.SqlDbType.NVarChar, 0, _
        System.Data.ParameterDirection.Input, 0, 0, _
        "Projektname", System.Data.DataRowVersion.Current, False, Nothing, "", "", ""))
    locDa.UpdateCommand.Parameters.Add(
        New System.Data.SqlClient.SqlParameter("@Projektbeschreibung", System.Data.SqlDbType.NVarChar, 0, _
        System.Data.ParameterDirection.Input, 0, 0, _
        "Projektbeschreibung", System.Data.DataRowVersion.Current, False, Nothing, "", "", ""))
    .
    .
    .
    ' Rest aus Platzgründen weggelassen
    .
    .
    locDa.UpdateCommand.Parameters.Add(
        New System.Data.SqlClient.SqlParameter("@Original_IDProjekte", System.Data.SqlDbType.Int, 0, _
        System.Data.ParameterDirection.Input, 0, 0, _
        "IDProjekte", System.Data.DataRowVersion.Original, False, Nothing, "", "", ""))
    locDa.UpdateCommand.Parameters.Add(
        New System.Data.SqlClient.SqlParameter("@Original_Projektname", System.Data.SqlDbType.NVarChar, 0, _
        System.Data.ParameterDirection.Input, 0, 0, _
        "Projektname", System.Data.DataRowVersion.Original, False, Nothing, "", "", ""))
    locDa.UpdateCommand.Parameters.Add(
        New System.Data.SqlClient.SqlParameter("@IsNull_Projektbeschreibung", System.Data.SqlDbType.Int, 0, _
        System.Data.ParameterDirection.Input, 0, 0, _
        "Projektbeschreibung", System.Data.DataRowVersion.Original, True, Nothing, "", "", ""))
    locDa.UpdateCommand.Parameters.Add(
        New System.Data.SqlClient.SqlParameter("@Original_Projektbeschreibung", System.Data.SqlDbType.NVarChar, 0, _
        System.Data.ParameterDirection.Input, 0, 0, _
        "Projektbeschreibung", System.Data.DataRowVersion.Original, False, Nothing, "", "", ""))

```

```

    locDa.UpdateCommand.Parameters.Add(
        New System.Data.SqlClient.SqlParameter("@Original_Startzeitpunkt", System.Data.SqlDbType.DateTime,
        0, System.Data.ParameterDirection.Input, 0, 0,
        "Startzeitpunkt", System.DataDataRowVersion.Original, False, Nothing, "", "", ""))
    locDa.UpdateCommand.Parameters.Add(
        New System.Data.SqlClient.SqlParameter("@Original_Endzeitpunkt", System.Data.SqlDbType.DateTime, 0,
        0, System.Data.ParameterDirection.Input, 0, 0,
        "Endzeitpunkt", System.DataDataRowVersion.Original, False, Nothing, "", "", ""))
    .
    .
    .
    ' Rest aus Platzgründen weggelassen
    .
    .
End Module

```

## Für »Mal-Eben-Abfragen« – die CommandBuilder-Klasse

Nun finden Sie es vielleicht ganz toll, Welch enorme Flexibilität Ihnen die verschiedenen *Command*-Objekte des *DataAdapters* einbringen. Möglicherweise benötigen Sie diese Flexibilität aber überhaupt nicht für kleinere Datenbankprojekte, sondern sind ein wenig entsetzt über die Vorleistung, die Sie erbringen müssen, um überhaupt die *Update*-Methode des *DataAdapters* anwenden zu können.

Doch keine Angst, ich kann Sie beruhigen, denn auch für diesen Fall haben die Entwickler von ADO.NET vorgesorgt. Ähnlich wie beim alten ADO, das sowohl Schemata als auch die notwendigen Aktualisierungs-SQL-Befehle selbstständig erstellen konnte, besteht auch bei ADO.NET die Möglichkeit, diese Aktualisierungslogiken *ausarbeiten zu lassen*. Der Schlüssel zum Glück ist dabei das so genannte *CommandBuilder*-Objekt.

Mit seiner Hilfe benötigen Sie lediglich eine gültige *SELECT*-Abfrage, aus der der *DataAdapter* ein Schema für eine *DataTable* erstellen kann, das aus einem Resultset hervorgeht. Die übrigen Anweisungen erstellt dann anschließend der *CommandBuilder* – im Falle von SQL Server nennt es sich übrigens *SqlCommandBuilder*.

---

**BEGLEITDATEIEN:** Sie finden das folgende Beispiel im Verzeichnis *.\VB 2005 - Entwicklerbuch\G - Smart-Client\Kap32\Insert Update CommandBuilder*. Bitte achten Sie auch hier wieder darauf, dass Sie vor dem Programmstart die vom vorherigen Beispiel hinzugefügten Datensätze löschen, damit das Beispiel reibungslos funktionieren kann.

---

Das folgende Codelisting zeigt die Vorgehensweise zur Anwendung des *SqlCommandBuilder*.

```

Imports System.Data.SqlClient

Module mdlMain
    Sub Main()

        Dim locConnection As SqlConnection

        'Connection-Objekt holen, damit die Datenbankverbindung hergestellt werden kann
        locConnection = New _
            SqlConnection("Data Source=.\SQLEXPRESS;Initial Catalog=Hecktick;Integrated Security=True")

        'Verbindung öffnen
        locConnection.Open()

        'Damit sorgen wir dafür, dass die Verbindung wieder geschlossen wird,

```

```

'wenn locConnection aus dem Using-Scope herausläuft.
Using locConnection

    'Alle Commands im DataAdapter definieren
    Dim locAdapter As New SqlDataAdapter

    'Die Selectabfrage für den Adapter definieren,
    'dann daraus generiert der CommandBuilder die
    'anderen Command-Objekte für Einfügen, Aktualisieren und Löschen
    locAdapter.SelectCommand = New SqlCommand("SELECT * FROM Projekte")
    locAdapter.SelectCommand.Connection = locConnection

    'Den CommandBuilder zuhilfe nehmen und dem Adapter zuweisen.
    Dim locCmdBuilder As New SqlCommandBuilder(locAdapter)

    'Neue Datentabelle
    Dim locDataTable As New DataTable

    'Könnten wir selber definieren, aber so geht's schneller.
    locAdapter.FillSchema(locDataTable, SchemaType.Source)

    'Neue Datenzeile
    Dim locDataRow As DataRow = locDataTable.NewRow()

    'Entsprechend des Datenschemas der Tabelle mit Datenfüllen.
    '(IDProjekte wird von Sql Server selbst geschrieben, da es ein
    ' AutoIncrement-Feld ist. Deswegen übergeben wir hier 'Nothing')
    locDataRow.ItemArray = New Object()
        {Nothing, "Orcas", " (VS 2007) Schreiben des Entwicklerbuchs", _
         #2/1/2007#, #12/31/2007#, "Büro"}
    'Die Zeile der Tabelle hinzufügen
    locDataTable.Rows.Add(locDataRow)

    'Das Gleiche mit einer weiteren Zeile
    locDataRow = locDataTable.NewRow()
    locDataRow.ItemArray = New Object()
        {Nothing, "Katmai", "(SQL Server 2005) Schreiben des Crash-Kurs", _
         #3/1/2007#, #1/31/2008#, "Büro"}
    locDataTable.Rows.Add(locDataRow)

    'Und nun die neue Tabelle mit der Datenbank synchronisieren.
    'Hier wird SQL-INSERT zweimal hintereinander ausgeführt.
    locAdapter.Update(locDataTable)

    Console.WriteLine("Betrachten Sie die hinzugefügten Daten nun im Server-Explorer")
    Console.WriteLine("Drücken Sie anschließend Return")
    Console.WriteLine()
    Console.ReadLine()

    'DataTable neu einlesen
    locDataTable.Clear()
    locAdapter.Fill(locDataTable)

```

```

'Jetzt verwenden wir eine View, um die 'Katmai'-Zeile zu ermitteln.
'Das ist die, die wir als letztes geändert haben!

Dim loc DataView As New DataView(locDataTable, "Projektname='Katmai'", _
    "Projektname", DataViewRowState.CurrentRows)

'Eigentlich können wir nur genau ein zurückbekommen.
locDataRow = loc DataView(0).Row

'Daten ändern.
locDataRow.ItemArray = New Object() _
    {Nothing, "Katmai", "(SQL Server 2007) Schreiben des Crash-Kurses", _
     #2/1/2007#, #10/31/2007#, "Ruprechts Büro"}

'Änderungen zurückschreiben.
locAdapter.Update(locDataTable)

Console.WriteLine("Betrachten Sie die geänderten Daten nun im Server-Explorer")
Console.WriteLine("Drücken Sie anschließend Return")
Console.ReadLine()
End Using

End Sub
End Module

```

---

**WICHTIG:** Das reibungslose Funktionieren von SqlCommandBuilder ist jedoch an einige Voraussetzungen gebunden. So muss die Tabelle einen Primärschlüssel (*PrimaryKey*) haben, der Teil der SELECT-Anweisung ist. Zudem darf die SELECT-Anweisung sich nur auf Spalten einer Tabelle beziehen. Falls Sie verknüpfte Tabellen aktualisieren wollen, müssen Sie daher beide Tabellen einzeln mit einem DataAdapter in ein DataSet laden und können dort ein DataRelation-Objekt erstellen. So können dann auch zwei CommandBuilder die verknüpften Tabellen aktualisieren.

---

## DataSets und typisierte DataSets

Von der eigentlich bekanntesten ADO.NET-Klasse, der **DataSet**-Klasse, war bislang mit noch keinem Wort die Rede, aber das hat auch seinen Grund. Die bisherigen Beispiele haben ausschließlich mit dem **DataTable**-Objekt für die von der Datenbank getrennte Speicherung von Daten gearbeitet. Erst wenn es darum geht, Tabellenrelationen zu verarbeiten, dann stellen die so genannten **DataSet**-Objekte in Zusammenarbeit mit den **DataRelation**-Objekten eine gute Alternative zu JOIN-SQL-Abfragen dar – in vielen Fällen lässt sich damit sogar eine erhebliche Leistungssteigerung von Abfragen und Bearbeitungen erreichen. Und jetzt ahnen Sie es vielleicht schon: Wenn derlei Aktionen mit **DataSet**-Objekten möglich sind, dann dienen sie notwendigerweise auch als Container für mehrere Tabellen, also **DataTable**-Objekten. Und so ist es auch.

Auf das komplette Spektrum der **DataSet**-Klasse kann und will dieses Buch nicht eingehen, dafür sind sie einfach zu mächtig, und die Beschreibungen ihres Umgangs füllen locker halbe Bücher.

Doch eine Sache dieses großen Themas möchte ich Ihnen dennoch nicht vorenthalten, und das ist das interaktive Erstellen von typisierten DataSets mit der Visual Studio IDE.

Der Vorteil von typisierten DataSet-Objekten wird deutlich, wenn Sie sich die folgende Abbildung anschauen.

```
'Die Definitionen für typisierte Tabellen befinden sich in der DataSet-Klasse  
Dim locProjekteTable As New HecktickDataSet.ProjekteDataTable  
  
'Die Definitionen für typisierte Adapter für Tabellen befinden sich in  
'in einem separaten Namespace  
Dim locProjekteAdapter As New HecktickDataSetTableAdapters.ProjekteTableAdapter  
  
'Eine neue Datenzeile hinzufügen. Intellisense hilft Ihnen hier beim  
'Finden der richtigen Parameter, weil diese Datenzeile jetzt typisiert ist.  
locProjekteTable.AddProjekteRow("Orcas", " (VS 2007) Schreiben des Entwicklerberu  
#2/1AddProjekteRow (Projektname As String, Projektbeschreibung As String, Startzeitpunkt As Date, Endzeitpunkt As Date, Ausführungsor
```

**Abbildung 32.21:** Mit typisierten Datenobjekten wird das Entwickeln von Datenanwendungen dank IntelliSense und Typsicherheit nicht nur zum Kinderspiel, sondern macht auch richtig Spaß und spart eine Menge Zeit!

Tabellen in Form von DataTable-Objekten, die sich in typisierten DataSet-Objekten befinden, lassen sich *typischer* bearbeiten. Sie arbeiten beim Hinzufügen von Datenzeilen nicht mehr »nur« mit einem DataRow-Objekt, sondern vielleicht mit einem ProjekteRow-Objekt. Und Sie fragen ein Datenfeld nicht mehr mit

```
Dim locString As String = locDataRow("Projektname").ToString  
sondern typischer auf die folgende Art und Weise:
```

```
Dim locProjektname As String = locProjekteRow.Projektname
```

Das Schöne: Das Erstellen von typisierten DataSet-Objekten geht in Visual Basic 2005 einfach wie nie, und der folgende Abschnitt zeigt, wie es geht.

---

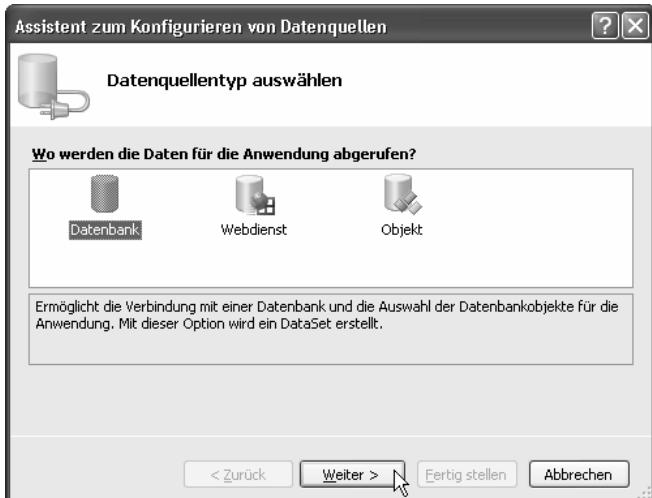
**TIPP:** Am Ende dieses Kapitels finden Sie als abschließendes Schmankerl noch eine komplett implementierte Anwendung, die Ihnen demonstriert, wie Sie die Demodatenbank dieses Kapitels auch tatsächlich in Form einer voll funktionsfähigen *SmartClient*-Anwendung nutzen können!

---

## Erstellen eines typisierten DataSets mit der Visual Studio IDE

Die folgende Schritt-für-Schritt-Anleitung zeigt, wie Sie mit Visual Studio 2005 (Sie benötigen mindestens die Standard-Edition) ein typisiertes DataSet erstellen.

- Legen Sie ein neues Windows Forms-Projekt unter dem Namen *TypedDataSet* an.
- Wählen Sie aus dem Menü *Daten* den Menübefehl *Datenquelle hinzufügen*.



**Abbildung 32.22:** Dieser Assistent hilft Ihnen beim Erstellen eines neuen typisierten *DataSet*

- Wählen Sie Datenbank, und klicken Sie auf Weiter.



**Abbildung 32.23:** Bestimmen Sie, welche SQL Server-Datenbank als Datenquelle fungieren soll

- Falls Sie bereits eine Datenverbindung in den letzten Beispielen zum Server Explorer hergestellt haben, befindet sich die Datenverbindung bereits in der Aufklappliste, und Sie wählen aus dieser Liste die Beispieldatenbank »HeckTick« aus. Andernfalls klicken Sie auf Neue Verbindung, und richten eine neue Verbindung zur Beispieldatenbank ein. Auf Seite 996 finden Sie die Beschreibung des entsprechenden Dialogs, der Ihnen dabei behilflich ist. Klicken Sie anschließend auf Weiter.



**Abbildung 32.24:** Legen Sie fest, ob die Verbindungszeichenfolge in der Anwendungseinstellungsdatei gespeichert werden soll

- Bestimmen Sie im nächsten Schritt (siehe Abbildung 32.24), ob die Verbindungszeichenfolge in den Einstellungen des Projektes gespeichert werden soll. Für dieses Beispiel habe ich mich dagegen entschieden. Mehr zu Projekteinstellungen erfahren Sie übrigens in ► Kapitel 25, aber auch ► Kapitel 3 und ► Kapitel 26 liefern hierzu wertvolle Tipps.
- Klicken Sie anschließend auf *Weiter*.



**Abbildung 32.25:** In diesem Assistentenschritt bestimmen Sie, aus welchen Tabellen *DataTable*-Objekt-Definitionen in das typisierte *DataSet* einfließen sollen

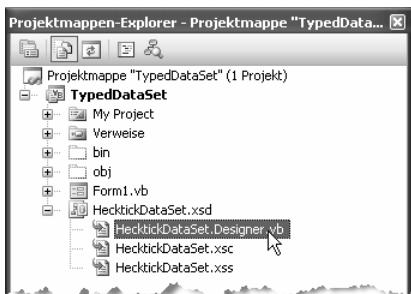
- Im letzten Schritt des Assistenten, den Sie auch in Abbildung 32.25 sehen können, definieren Sie, auf welche Tabellen innerhalb des typisierten *DataSet* Sie zugreifen wollen

---

**HINWEIS:** Sie könnten natürlich für jede Tabelle auch ein eigenes DataSet erstellen, doch weder ist es nötig, diesen Aufwand zu betreiben, noch würde es Sinn ergeben. Wenn Sie später nämlich Relationen zwischen den verschiedenen DataTable-Objekten herstellen wollen, *sollten* Sie die verschiedenen Tabellen (oder auch Ansichten, bzw. Resultsets, die aus gespeicherten Prozeduren hervorgehen) sogar unter einem DataSet zusammenfassen.

---

- Klicken Sie schließlich auf *Fertigstellen*.



**Abbildung 32.26:** Das typisierte DataSet befindet sich anschließend im Projektmappen-Explorer

## Grundsätzliches zu typisierten DataSets

Im Projektmappen-Explorer finden Sie anschließend eine neue Datei namens *HeckTickDataSet.xsd*. Wenn Sie sich alle Dateien des Projektes durch Mausklick auf das entsprechende Symbol im Projektmappen-Explorer anzeigen lassen, sehen Sie, dass es – ähnlich wie bei Formularen – weitere Dateien zu dieser DataSet-Definition gibt. Eine davon nennt sich *HeckTickDataSet.Designer.vb*, und diese Visual Basic-Codedatei enthält den Code, der vom Designer erstellt wurde und eine spezielle Klasse darstellt, der die HeckTickDataSet-Klasse auf Basis der DataSet-Klasse typisiert.

Lassen Sie uns an einem Beispiel anschauen, was Ihnen dieser Code im Detail beim Entwickeln für Vorteile bringt.

In den vergangenen Abschnitten haben Sie schon kennen gelernt, welche Schritte notwendig sind, um eine Verbindung zu Datenbank aufzubauen, einer Tabelle dieser Datenbank zwei Datenzeilen hinzuzufügen und anschließend eine dieser neu hinzugefügten Zeilen zu verändern.

- Fügen Sie eine Schaltfläche in das Formular ein. Definieren Sie *Test* als Schaltflächenbeschriftung und nennen Sie die Schaltfläche *btnTest*.
- Doppelklicken Sie auf die Schaltfläche, um den Codeeditor zu öffnen und die Ereignisbehandlungsprozedur für das Click-Ereignis auszuformulieren

---

**BEGLEITDATEIEN:** Der Einfachheit halber können Sie eine vorbereitete Textdatei mit den benötigten Codezeilen verwenden, die Sie unter dem Namen *TypedDataSet.txt* im Verzeichnis *.\VB 2005 - Entwicklerbuch\G - Smart-Client\Kap32* finden.

---

Der Code, für diese Prozedur, sieht folgendermaßen aus:

```
Public Class Form1
    Private Sub btnTest_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
        Handles btnTest.Click
```

```

'Die Definitionen für typisierte Tabellen befinden sich in der DataSet-Klasse
Dim locProjekteTable As New HecktickDataSet.ProjekteDataTable

'Die Definitionen für typisierte Adapter für Tabellen befinden sich
'in einem separaten Namespace
Dim locProjekteAdapter As New HecktickDataSetTableAdapters.ProjekteTableAdapter

'Eine neue Datenzeile hinzufügen. IntelliSense hilft Ihnen hier beim
'Finden der richtigen Parameter, weil diese Datenzeile jetzt typisiert ist.
locProjekteTable.AddProjekteRow("Orcas", " (VS 2007) Schreiben des Entwicklerbuchs", _
#2/1/2007#, #12/31/2007#, "Büro")

'Eine typisierte Datenzeile deklarieren.
Dim locProjektRow As HecktickDataSet.ProjekteRow

'Vorteil: AddProjektRow liefert die hinzugefügte Zeile als typisierte Row zurück.
locProjektRow = locProjekteTable.AddProjekteRow("Katmai", _
"(SQL Server 2005) Schreiben des Crash-Kurs", _
#3/1/2007#, #1/31/2008#, "Büro")

'Aktualisieren funktioniert mit dem Projekte-TableAdapter.
locProjekteAdapter.Update(locProjekteTable)

'Bei AutoIncrement-Feldern wird automatisch dafür gesorgt,
'dass das Feld aktualisiert wird.
Dim locIDProjekte As Integer = locProjektRow.IDProjekte

MessageBox.Show("Datensätze wurden hinzugefügt. Sie können Sie im Server-Explorer betrachten!")

'Daten komplett neu einlesen
locProjekteTable.Clear()

'Funktioniert wie beim Adapter.
locProjekteAdapter.Fill(locProjekteTable)

'Die Zeile mit der entsprechenden ID finden
locProjektRow = locProjekteTable.FindByIDProjekte(locIDProjekte)

'Jetzt können wir die Datenfelder direkt per Eigenschaftennamen ändern,
'die genau so heißen, wie die Datenfelder in der Datenbank.
locProjektRow.Projectbeschreibung = "(SQL Server 2007) Schreiben des Crash-Kurses"
locProjektRow.Startzeitpunkt = #2/1/2007#
locProjektRow.Endzeitpunkt = #10/31/2007#
locProjektRow.Ausführungsort = "Ruprechts Büro"

'Tabelle aktualisieren.
locProjekteAdapter.Update(locProjekteTable)
End Sub
End Class

```

## Zusammenspiel zwischen TableAdapter und typisierter Table-Klasse

Im vorherigen Beispiellisting können Sie gut das Zusammenspiel zwischen der typisierten `DataTable` einer typisierten `DataSet`-Klasse und dem so genannten `TableAdapter`-Objekt erkennen. Ein `TableAdapter` ist eine Klasse, nach deren Basistyp Sie vergeblich in der .NET-Framework-Bibliothek suchen. Diese Klasse wird nämlich komplett vom `DataSet`-Designer erstellt, sie wird nicht von `DataAdapter` abgeleitet, ersetzt aber, neben anderen Funktionen, die sie implementiert, die Funktionalität der `DataAdapter`-Klasse, die Sie in den vorherigen Beispielen kennen gelernt haben, vollständig.

So bleibt die prinzipielle Vorgehensweise hinter den Kulissen für typisierte `DataTable`-Objekte gleich, denn:

- Für jede Zeile, die Sie einer typisierten `DataTable` (`ProjekteDataTable` im Beispiel) hinzufügen, kapselt die entsprechende `TableAdapter`-Klasse (`ProjekteTableAdapter` im Beispiel) eine SQL-`INSERT`-Abfrage. Für das Aktualisieren von Zeilen wird eine entsprechende `UPDATE`-Anweisung und für das Löschen eine `DELETE`-Anweisung jeweils mit den entsprechenden Parameters-Auflistungen generiert.
- Um eine typisierte `DataTable` mit Daten zu füllen, verwenden Sie die `Fill`-Methode des entsprechenden `TableAdapter`.
- Änderungen führen Sie, wie im vorherigen Beispiel zu sehen, an der typisierten `DataTable` durch. Sie können mit `AddTabellennameRow` eine neue Zeile einer Tabelle hinzufügen, Zeilen mit `RemoveTabellennameRow` aus der Tabelle entfernen oder direkt durch typsichere Eigenschaften auf die Datenfelder eines Datensatzes zugreifen, der durch ein entsprechendes `TabellennameRow`-Objekt repräsentiert wird (`Tabellenname` steht dabei jeweils für den Namensausschnitt im typisierten Element, zum Beispiel `ProjekteRow` für eine Datenzeile der `Projekte`-Tabelle im Beispiel).
- Um die Änderungen an die Datenbank zu übermitteln, verwenden Sie die `Update`-Methode des jeweiligen `TableAdapter`-Objektes.
- Verfügte die Ausgangstabelle über einen primären Schlüssel, befindet sich im jeweiligen `TabellennameTableAdapter` eine Methode namens `FindByIdPrimärSchlüsselname`, die ein `TabellennameRow`-Objekt zurückliefert. Im Beispiel gibt es die Methode `FindByIdProjekte`, die versucht eine Zeile mit der entsprechenden ID innerhalb einer `ProjekteDataTable`-Instanz zu finden, und diese Zeile als `ProjekteRow`-Objekt zurückliefert.

---

**TIPP:** Sie können den jeweiligen `TableAdapter` um weitere Abfrage-Methode erweitern, die jeweils auf neuen SQL-SELECT-Abfragen basieren. Das Beste: Diese Abfragen lassen sich interaktiv mit dem Designer erstellen. Wie das geht, zeigt der folgende Abschnitt.

---

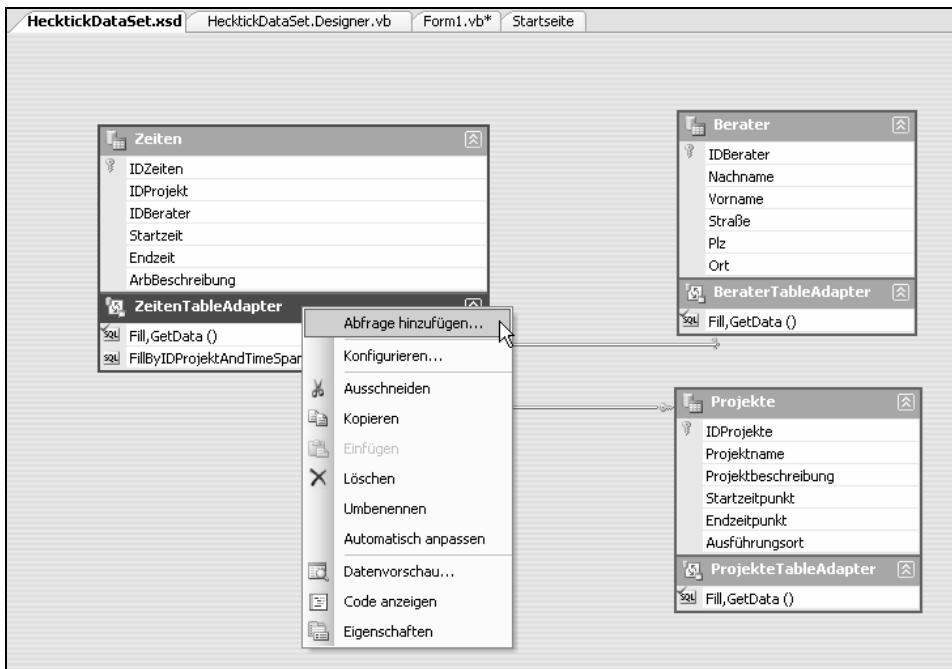
## Erweitern des TableAdapters um zusätzliche Abfragefunktionen, denen Parameter übergeben werden können

Standardmäßig implementiert ein `TableAdapter`, der lediglich aus einer Tabellendefinition hervorgegangen ist, eine simple `SELECT`-Abfrage an die Datenbank, die *alle* Felder und Daten einer Tabelle zurückliefert. Für komplexere Business-Anwendungen reicht das natürlich nicht aus; hier müssen Tabellenselektionen oftmals viel individueller und flexibler von statten gehen. Bedenken Sie einmal, was passieren würde, wenn Sie mit in einem großen Konzern alle Daten der Buchungstabelle mit `SELECT` abrufen würden: Mehrere Millionen Datenzeilen würden übertragen. In der Regel benötigt

man hier auch Parameter, die einer Abfragefunktion übergeben werden müssen, um entsprechende Datensätze aus der Datenbank zu ermitteln, die gemäß der übergebenen Parameter in irgendeiner Form gefiltert wurden.

Anstatt nun selbst Hand anzulegen, und komplizierte SELECT-Abfragen zu formulieren, verwenden Sie abermals den Designer. Wie das geht, zeigt die folgende Schritt-für-Schritt-Anleitung, die auf dem bereits erstellten Projekt des vorherigen Abschnitts aufsetzt.

- Zu diesem Zweck schalten Sie in die Designer-Ansicht des DataSet-Objektes, indem Sie im Projektmappen-Explorer einen Doppelklick auf den Wurzelzweig des DataSets ausführen.

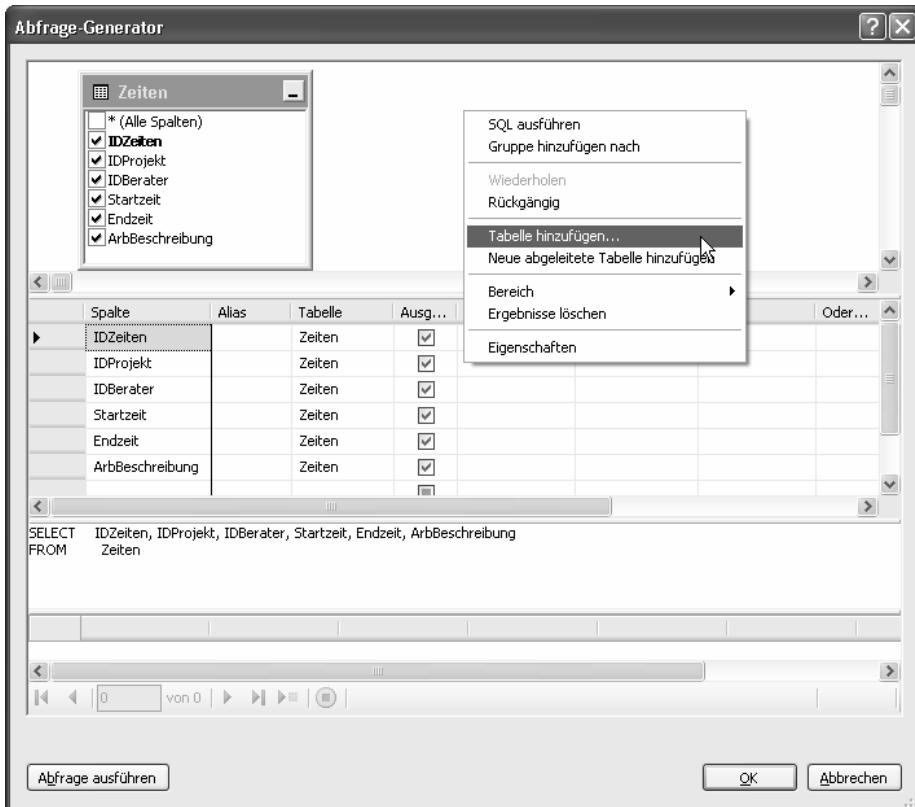


**Abbildung 32.27:** Sie fügen einem *TableAdapter* eine neue Abfragefunktion hinzu, indem Sie das Kontextmenü über der *TableAdapter*-Titelzeile öffnen und den Menübefehl *Abfrage hinzufügen* wählen

- Fahren Sie mit der Maus auf die Titelzeile eines Tabellen-Adapters (nicht der Tabelle!), öffnen Sie das Kontextmenü per Rechtsklick, und wählen Sie den Menübefehl *Abfrage hinzufügen*. Für dieses Beispiel wählen Sie bitte den *ZeitenTableAdapter*.
- Sie befinden sich anschließend im *Konfigurationsassistenten für TableAdapter-Abfragen*, dessen Einstellungen Sie auf den ersten beiden Seiten belassen, wie sie sind, und mit zweimaligem *Weiter* zur übernächsten Seite wechseln.
- Auf der Seite, die jetzt erscheint, klicken Sie auf *Abfragegenerator*. Sie sehen anschließend einen Dialog, etwa wie in Abbildung 32.28.

Das Ziel ist es jetzt, mit dem Abfragegenerator die Basis für eine neue *TableAdapter*-Funktion zu schaffen, die Datensätze der Zeitentabelle der Beispieldatenbank zurückliefert, die nur für ein be-

stimmtes Projekt und innerhalb eines bestimmten Zeitbereichs getätigten wurden. Die Tabelle soll ferner nach den Beraternachnamen und der Anfangszeit sortiert werden.



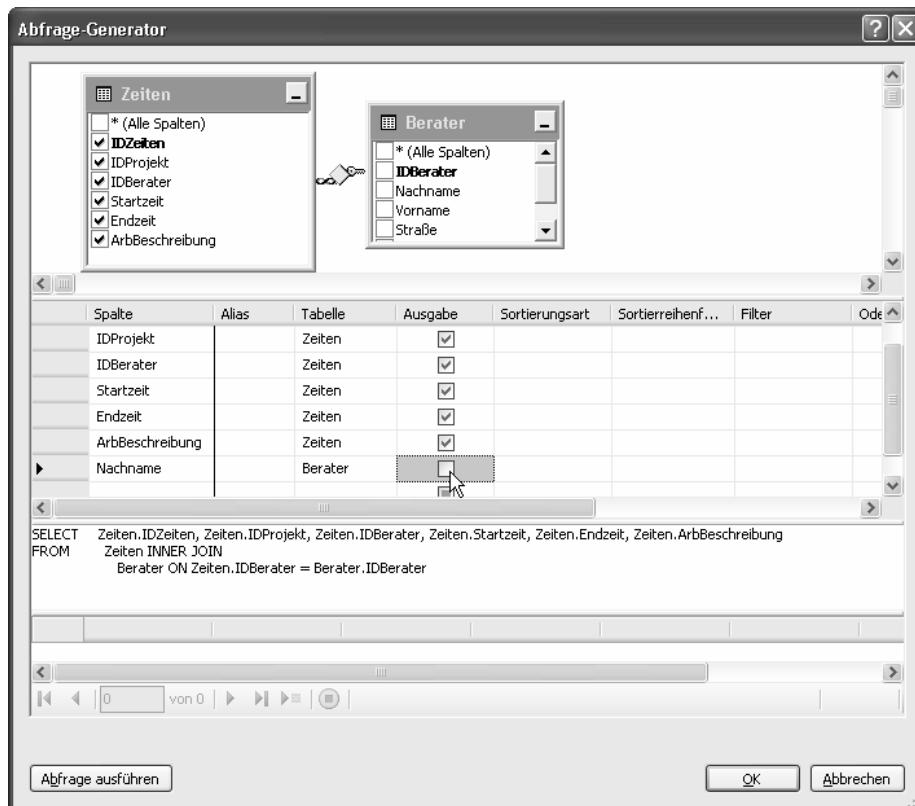
**Abbildung 32.29:** Mit dem Abfragegenerator können Sie *SELECT*-Abfragen unter anderem für neue Abfragefunktionen des *TableAdapter* erstellen

---

**WICHTIG:** Wenn Sie eine Abfrage für eine neue *TableAdapter*-Funktion erstellen, achten Sie darauf, das Schema des vorhandenen *TableAdapters* nicht zu erweitern. Sie dürfen also – wie es beispielsweise bei *JOIN*-Abfragen passiert – keine zusätzlichen Felder hinzufügen. Das bedeutet aber nicht, dass Sie keine *JOIN*-Abfragen verwenden dürfen, um beispielsweise nach Feldern in verknüpften Tabellen zu selektieren oder sortieren, wie das folgende Beispiel zeigt.

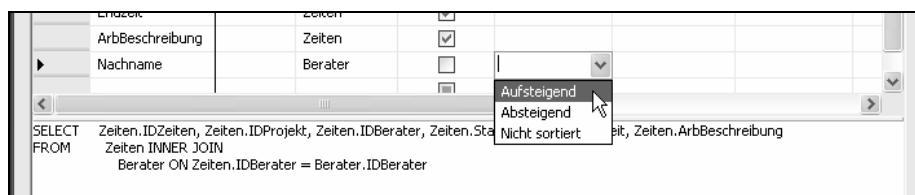
- Da wir für dieses Beispiel das Resultset nach dem Berater-Nachnamen sortieren lassen wollen, dieser aber über die Zeitentabelle selbst nicht verfügbar ist, müssen wir einen *INNER JOIN* erstellen, bei dem *IDBerater* der Zeitentabelle als Indikator auf den eigentlichen Datensatz des Beraters in der Beratertabelle herhält. Neben der standardmäßig schon angezeigten Zeitentabelle benötigen wir also die Beratertabelle, um dort das benötigte Feld auszuwählen und die *JOIN*-Verknüpfung herzustellen. Dazu wählen Sie aus dem Kontextmenü, das Sie über dem freien Tabellenbereich öffnen, den Menübefehl *Tabelle hinzufügen*.
- Wählen Sie aus dem Dialog die Beratertabelle aus.

- Setzen Sie nun in der Beratertabellendefinition im oberen Bereich des Dialogs für das Feld Nachname ein Häkchen. In diesem Moment erscheint das entsprechende Datenfeld auch in der Datenfeldliste darunter. In der Spalte Ausgabe dieser Datenfeldliste entfernen Sie das Häkchen anschließend wieder, sodass sich anschließend ein Bild ergibt, das Sie auch in Abbildung 32.30 sehen.



**Abbildung 32.30:** Um ein Feld einer anderen Tabelle nur für Selektionszwecke verfügbar zu machen, ist es am einfachsten, es in der Tabellendefinition kurz anzuklicken, um es der Datenfeldliste hinzuzufügen, seine Ausgabe in der Datenfeldliste aber anschließend zu unterdrücken

- Nun bestimmen Sie die entsprechenden Filter und Sortierkriterien. Da nach dem Feld *Nachname* sortiert werden soll, klicken Sie in der entsprechenden Zeile in die Zelle der Spalte *Sortierungsart*.

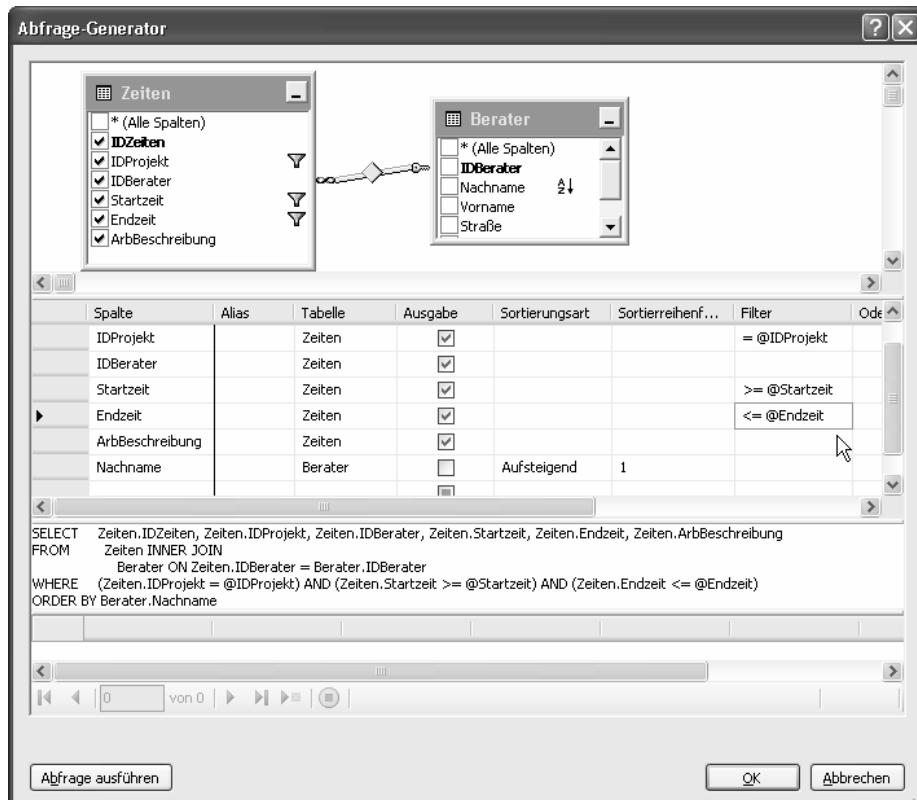


**Abbildung 32.31:** Bestimmen Sie die Sortierungsart eines Feldes mit der Aufklappliste in der jeweiligen Datenfeldlistenzeile

- Wählen Sie aus der Aufklappliste, die jetzt erscheint und sich öffnet, den Eintrag *Aufsteigend*.
- Bestimmen Sie als nächstes die zu übergebenden Parameter, indem Sie mit @ beginnende Platzhalter für die Selektionskriterien verwenden. Möchten Sie beispielsweise, dass Sie der zu generierenden Funktion später einen Parameter IDProjekte zur Selektion der Zeitdaten nur nach bestimmten Projekten übergeben können, definieren Sie in der Datenfeldlistenzeile *IDProjekt* den Ausdruck =@IDProjekt als Filterkriterium.

**HINWEIS:** Diese Syntax gilt übrigens für den SQL Server. In anderen Namespaces wie OLEDB oder Oracle werden Parameter teilweise durch »?« oder andere Sonderzeichen notiert. Schauen Sie dann einfach in Dokumentation, das Prinzip ist das Gleiche.

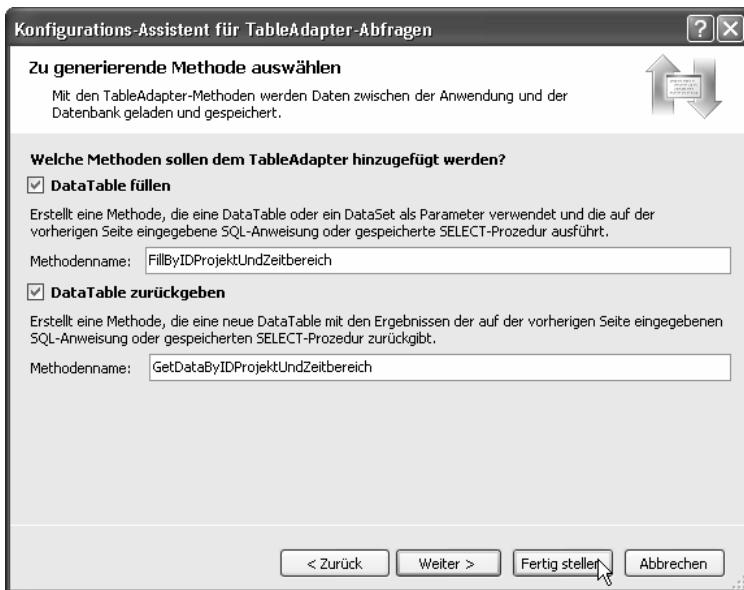
- Für die Parameter Startzeit und Endzeit, die den auszuwertenden Zeitbereich definieren sollen, verfahren Sie äquivalent. Orientieren Sie sich dabei am besten an Abbildung 32.32.



**Abbildung 32.32:** So definieren Sie die Filterkriterien mit der Angabe von Variablen, die später in Funktionsparameter umgewandelt werden

- Und das war es auch schon! Klicken Sie auf *OK*, um den Abfrage-Generator zu verlassen.
- Klicken Sie auf *Weiter*, um zum letzten Assistentenschritt zu gelangen.

- Schließlich bestimmen Sie noch, so wie in Abbildung 32.33 zu sehen, die Funktionsnamen, die der DataSet-Designer für den *TableAdapter* generieren soll. Für unser Beispiel verwenden Sie die Namen *FillByIDProjektUndZeitbereich* sowie *GetDataByIDProjektUndZeitbereich*.



**Abbildung 32.33:** Bestimmen Sie schließlich noch den Namen der *TableAdapter*-Funktionen

- Klicken Sie anschließend auf *Fertig stellen*, um den Assistenten abzuschließen.
- Wechseln Sie anschließend zur Design-Ansicht von *Form1.vb*.
- Fügen Sie im Formular eine *Test2*-Schaltfläche namens *btnTest2* ein.
- Doppelklicken Sie auf die Schaltfläche, um das Codefenster zu öffnen, und den Rumpf der entsprechenden Ereignisbehandlungsprozedur einzufügen.

---

**BEGLEITDATEIEN:** Der Einfachheit halber können Sie eine vorbereitete Textdatei mit den benötigten Codezeilen verwenden, die Sie unter dem Namen *TypedDataSet2.txt* im Verzeichnis *.\VB 2005 - Entwicklerbuch\G - Smart-Client\Kap32* finden.

---

```
Private Sub btnTest2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
Handles btnTest2.Click

    'Wir benötigen alle Adapter, um an die Daten heranzukommen.
    Dim locZeitenAdapter As New HecktickDataSetTableAdapters.ZeitenTableAdapter
    Dim locProjekteAdapter As New HecktickDataSetTableAdapters.ProjekteTableAdapter
    Dim locBeraterAdapter As New HecktickDataSetTableAdapters.BeraterTableAdapter

    'Dieses Mal verwenden wir wegen der Relation ein DataSet
    Dim locHeckTickDataSet As New HecktickDataSet()
    locProjekteAdapter.Fill(locHeckTickDataSet.Projekte)
    locBeraterAdapter.Fill(locHeckTickDataSet.Berater)
```

```

'TODO: Passen Sie den Zeitbereich Ihrer Beispieldatenbank an.
locZeitenAdapter.FillByIDProjektUndZeitbereich(locHeckTickDataSet.Zeiten, _
    locHeckTickDataSet.Projekte(0).IDProjekte, _
    #3/15/2006#, #3/15/2006 11:59:59 PM#)

'Alle Datensätze mit Nachnamen ausgeben; die Relation zwischen Zeiten und
'Berater wurde einerseits durch die Verwendung eines typisierten DataSets
'und andererseits durch die Erstellung der Relationen der Tabellen im SQL Server
'automatisch hergestellt.
For Each locZeitenRow As HecktickDataSet.ZeitenRow In locHeckTickDataSet.Zeiten.Rows
    Debug.Print(locZeitenRow.BeraterRow.Nachname & ": " &
        locZeitenRow.Startzeit.ToShortDateString & " - " &
        locZeitenRow.Endzeit.ToShortDateString)
Next

End Sub

```

Sie sehen im Codelisting, dass die *TableAdapter*-Funktion, die wir gerade erstellt haben, nun am *ZeitenTableAdapter* zur Verfügung steht.

Wenn Sie das Beispiel starten und auf die neue Schaltfläche klicken, sehen Sie folgende Ausgabe im Ausgabefenster.<sup>10</sup>

```

Ausgabe
Ausgabe anzeigen von: Debuggen
Ademmer: 15.03.2006 - 15.03.2006
Ademmer: 15.03.2006 - 15.03.2006
Albrecht: 15.03.2006 - 15.03.2006
Braun: 15.03.2006 - 15.03.2006
Hörstmann: 15.03.2006 - 15.03.2006
Müller: 15.03.2006 - 15.03.2006
Sonntag: 15.03.2006 - 15.03.2006
Sonntag: 15.03.2006 - 15.03.2006
Tiemann: 15.03.2006 - 15.03.2006
Vielstedde: 15.03.2006 - 15.03.2006
Vielstedde: 15.03.2006 - 15.03.2006
Weichel: 15.03.2006 - 15.03.2006
Weichel: 15.03.2006 - 15.03.2006
Weigel: 15.03.2006 - 15.03.2006

```

**Abbildung 32.34:** Das Programm verwendet die neue Funktion des *TableAdapter*

## Erstellen von datenbankgebundenen Formularen in Windows Forms-Anwendungen

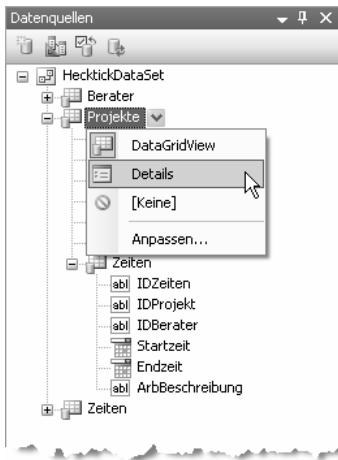
Das Repertoire des Designers in Sachen interaktives Anwendungsdesign ist damit allerdings noch lange nicht erschöpft.

---

<sup>10</sup> Um eine Anzeige im Ausgabefenster zu bekommen, muss im Optionsdialog von Visual Studio unter Umständen im Bereich *Debuggen/Allgemein* die Option *Gesamten Text aus Ausgabefenster an Direktfenster umleiten* ausgeschaltet sein.

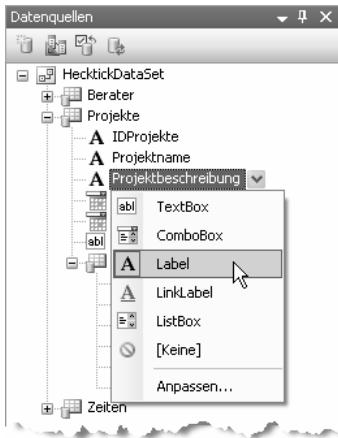
Die folgende Anleitung zeigt, wie Sie auf Basis der Anwendung, die wir bislang entwickelt haben, unser Formular, das sich ja bislang noch nicht mit übermäßigem Funktionsreichtum rühmen kann, in Nullkommanchts zu einem halbwegs brauchbaren Abfragewerkzeug von Projektzeiten ausbauen können.

- Zu diesem Zweck wechseln Sie in die Designer-Ansicht des Formulars.
- Wählen Sie aus dem Menü *Daten* den Menübefehl *Datenquellen anzeigen* aus.
- Öffnen Sie den Zweig vor *HeckTickDataSet*, den vor *Projekte* und schließlich den *Projekte* untergeordneten Zweig *Zeiten*. Orientieren Sie sich dabei am besten an Abbildung 32.35. Achten Sie darauf, dass es zwei Zeiten-Elemente gibt. Verwenden Sie im Folgenden den *Projekte* untergeordneten Zweig und nicht den am Ende der Liste stehenden.
- Öffnen Sie die Aufklappliste von *Projekte*.



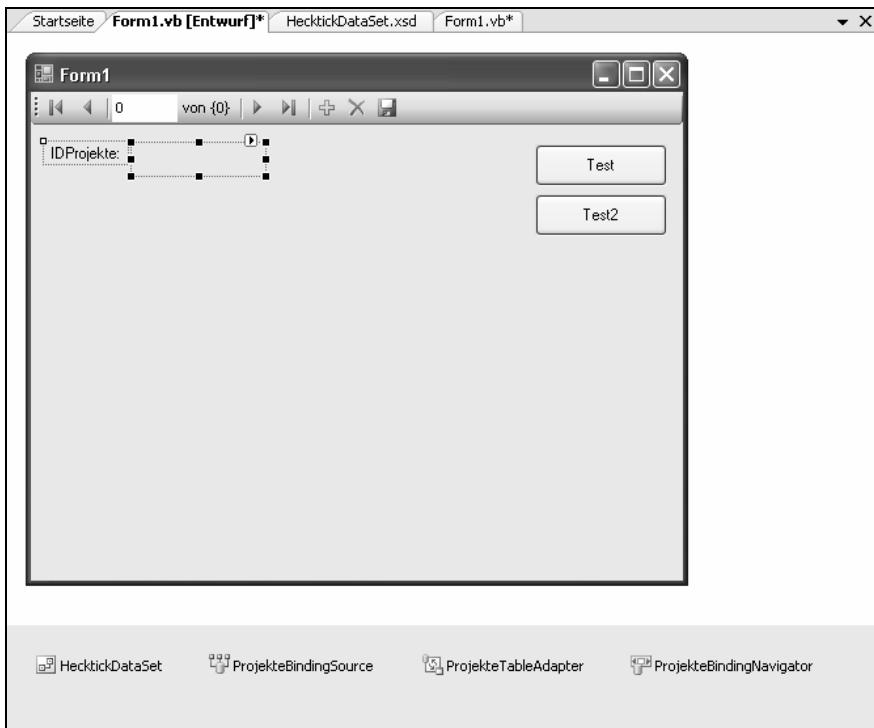
**Abbildung 32.35:** Erweitern Sie die entsprechenden Zweige im Datenquellenfenster und öffnen Sie die Aufklappliste im Projekte-Element

- Wählen Sie aus dem Menü den Menüpunkt *Details*.



**Abbildung 32.36:** Wählen Sie anschließend die Steuerelemententsprechungen jedes Datenfeldes aus den Aufklapplisten aus

- Öffnen Sie jede Aufklappliste, die sich hinter *IDProjekte*, *Projektname* und *Projektbeschreibung* befindet, und stellen Sie alle zugeordneten Steuerelemente für die Datenwiedergabe im Formular auf *Label*. Orientieren Sie sich an Abbildung 32.36.
- Nun ziehen das Feld *IDProjekte* vom Datenquellen-Fenster in das Formular, und beobachten Sie, was passiert.

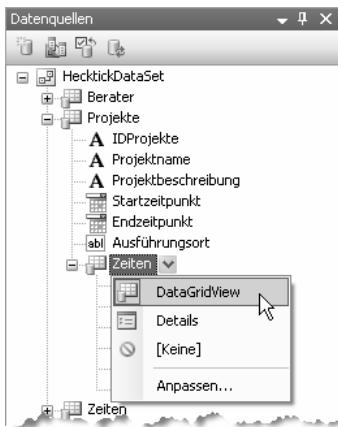


**Abbildung 32.37:** Nach dem Einfügen des ersten Feldes aus dem Datenquellen-Fenster hat der Designer automatisch eine Navigationsleiste und die entsprechenden Komponenten eingefügt, um die Datenbindung funktionsfähig zu machen

Mit dem letzten Schritt hat der Designer das HeckTickDataSet dem Komponentenfach hinzugefügt. Ferner hat er eine ProjekteBindingSource, einen entsprechenden ProjekteTableAdapter sowie einen ProjekteBindingNavigator (das ist das Steuerelement mit den Schaltflächen im Stile eines Videorekorders) dem Komponentenfach bzw. Formular hinzugefügt, und damit die komplette Funktionalität zum Browsen von Daten einer bestimmten Datenquelle implementiert. Sie können das Projekt zu diesem Zeitpunkt schon starten und werden feststellen, dass Sie bereits Zugriff auf die Projekte-Datentabelle nehmen können – auch wenn sich das Blättern in den Daten mit dem BindingNavigator zur Zeit nur im Aktualisieren der Projekte-ID bemerkbar macht.

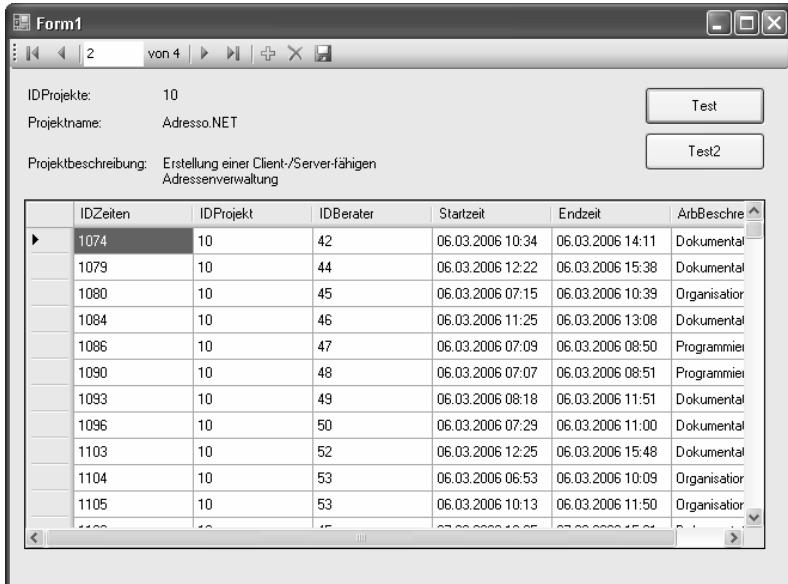
Die ProjekteBindingSource wird hier verwendet, um die Bindung der Datenquelle (die Projekte-Tabelle) an bestimmte Eigenschaften von Steuerelementen im Formular zu ermöglichen. So wurde mit der letzten Drag&Drop-Aktion im Beispiel die Text-Eigenschaft des Label-Steuerelements an das Datenfeld *IDProjekte* gebunden. (Mehr zum Thema BindingSource erfahren Sie übrigens in ▶ Kapitel 27.)

- Fügen Sie auf diese Weise die Felder *Projektname* und *Projektbeschreibung* ebenfalls im Formular ein.
- Vergrößern oder verkleinern Sie die Felder bzw. Beschriftungen ein wenig, sodass sie übersichtlich im Formular angeordnet sind.
- Öffnen Sie anschließend die Aufklapliste des untergeordneten *Zeiten*-Zweigs.



**Abbildung 32.38:** Öffnen Sie die Aufklapliste des *Zeiten*-Elements

- Wählen Sie *DataGridView*.



**Abbildung 32.39:** Das Ergebnis nach ein paar Minuten im Designer kann sich sehen lassen. Und das Beste ist: Sie haben dazu nicht eine einzige Zeile Code programmieren müssen!

- Verschieben Sie nun per Drag&Drop den kompletten Zeitenzweig in das Formular. Wieder wird der Designer aktiv, und fügt zusätzliche Komponenten in das Komponentenfach ein – dieses Mal um eine so genannte Master/Details-Ansicht zwischen Label-Feldern und DataGridView zu realisieren. Wenn Sie das Programm anschließend starten, dann verstehen Sie, was damit gemeint ist: Sie blättern mit dem BindingNavigator durch die Datensätze der *Projekte*-Tabelle (*Master*). Gleichzeitig zeigen Sie in der DataGridView alle Zeitendatensätze an, die zum gerade »aufgeschlagenen« Projekte-Datensatz gehören (*Details* – siehe Abbildung 32.39)

## Und so geht es weiter

Im Rahmen dieses Buches kann ein Kapitel, das ein Thema wie ADO.NET behandelt, allenfalls die Grundlagen klären. Das Thema ADO.NET liefert alleine Stoff für mehrere Kapitel Bücher, ja sogar ganze Bücher und Buchfortsetzungen. Das liegt nicht nur daran, dass das Framework mit ADO.NET eine extrem umfangreiche Klassenbibliothek zur Verfügung stellt, sondern dass ein Entwickler sich auch intensiv mit der SQL-Abfragesprache und dem SQL Server 2005 selbst auseinander setzen muss, um professionelle Datenbankanwendungen erstellen zu können.

Ihr Wissensstand nach der Lektüre dieses Kapitels sollte aber in jedem Fall ausreichend sein, um erste Datenbankapplikationen sicher entwickeln zu können. In vielen Fällen wissen Sie schon jetzt mehr, als viele VB6-Entwickler jemals zu »alten« ADO-Zeiten gewusst haben.

Sie haben gelernt, wie Sie »manuell« mit den ADO.NET-Komponenten umgehen müssen, und Sie haben erfahren, wie Sie Formulare auf einfache Weise und ohne Code zu programmieren zur Datenanzeige verwenden können. Das Geheimnis einer Anwendung liegt sicherlich in einer Mischung aus beidem, und es gibt viele weitere Aspekte von ADO.NET, die es zu ergründen gilt, und die Ihnen schließlich helfen, Ihre Meisteranwendung effizient und robust zu entwickeln.

Die Online-Hilfe von Visual Studio .NET stellt für weitere Nachforschungen zu diesem Thema eine großartige Wissensquelle dar, und letzten Endes wird Ihnen auch das Internet einmal mehr helfen, interessante Artikel zu den Themen zu finden.

Und schließlich haben wir auch noch die versprochene dokumentierte Beispieldatenbank für Sie parat (siehe Abbildung 32.40), die mithilfe der Techniken entstanden ist, die Sie in diesem Kapitel kennen gelernt haben.

---

**BEGLEITDATEIEN:** Sie finden das fertige Projektbeispiel im Verzeichnis \VB 2005 - Entwicklerbuch\G - Smart-Client\Kap32\HeckTick\HeckTick.sln. Und – soviel Kalauer muss sein: Schauen Sie sich es in Ruhe an – lassen Sie dabei keine HeckTick aufkommen!

---

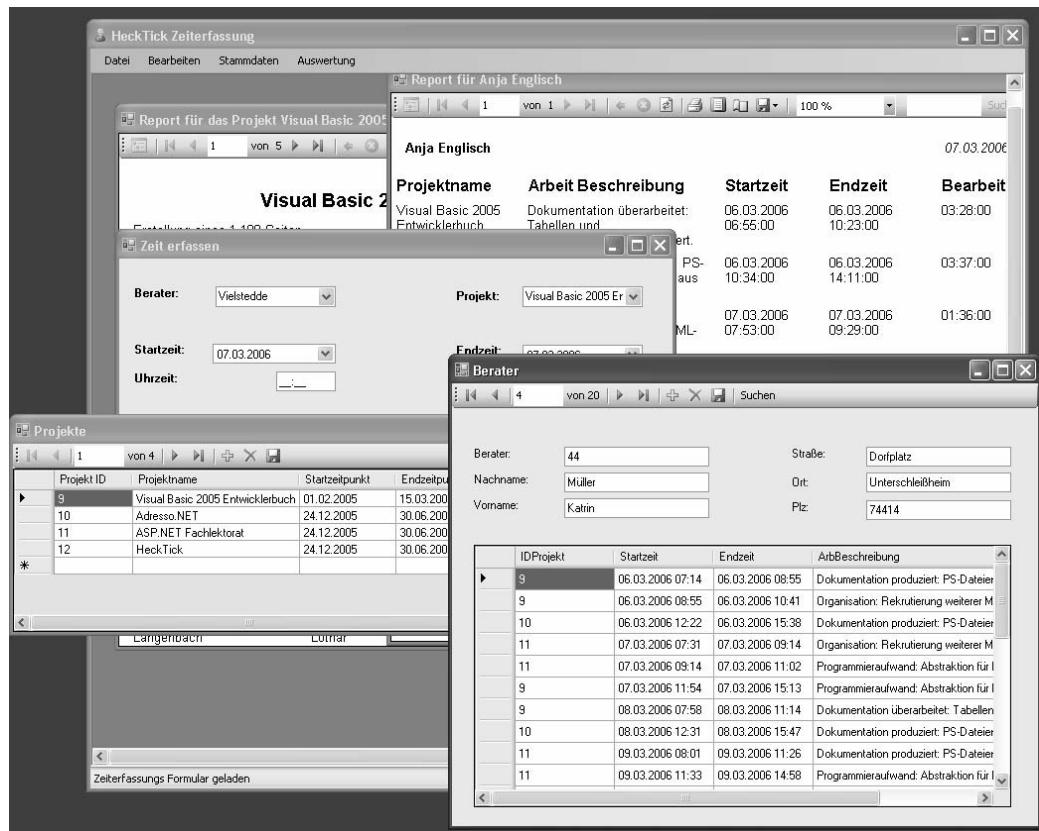


Abbildung 32.40: Hier kommt HeckTick auf!

**TIPP:** Falls Sie sich dazu entschließen, sich wirklich intensiv mit dem Thema zu beschäftigen, sollten Sie den Erwerb weiterer Bücher zum Thema in Erwägung ziehen. Ich selbst möchte Ihnen an dieser Stelle ein Buch empfehlen, das sich mit dem Thema SQL Server 2005 beschäftigt. Sein Titel: *SQL Server 2005, Konfiguration, Administration, Programmierung, 2. Auflage* (Microsoft Press, ISBN: 3-86063-997-8). Mit dem Autorenteam Ruprecht Dröge und Markus Raatz haben sich zwei professionelle SQL Server-Experten gefunden und ein rundum gutes und informatives Werk geschaffen.

# Stichwortverzeichnis

- # [End] Region 104  
&= -Operator 165  
\*= -Operator 166  
.cctor 239  
.ctor 239, 267  
.NET Framework  
    DateTime *siehe* Date-Datentyp  
    Int16 *siehe* Short-Datentyp  
    Int32 *siehe* Integer-Datentyp  
    Int64 *siehe* Long-Datentyp  
    UInt16 *siehe* UShort-Datentyp  
    UInt32 *siehe* UInteger-Datentyp  
    UInt64 *siehe* ULong-Datentyp  
.NET-Framework  
    Assembly 196  
    Auflistungen 565  
    Base Class Library 198  
    BCL 198  
    Class Library 198  
    CLI 198  
    CLR 198  
    Common Language Infrastructure 198  
    Common Language Runtime 198  
    Common Type System 199, 201  
    CTS 199, 201  
    Eigenschaften überschreiben 274  
    FCL 198  
    Framework, Aufbau 200  
    IML 199  
    Intermediate Language 199  
    JITter 199  
    Klassen erweitern 274  
    Managed Heap 275, 327  
    Methoden überschreiben 274  
    Namespace 197  
    Polymorphie *siehe* Polymorphie  
    statische Methoden 223  
    Terminologien 196  
    Typsicherheit 201  
    zusätzliche Werkzeuge 238  
/= 166  
:IML  
    Klassen, Umsetzung 267  
+= Operator 165  
<<-Operator 166  
= 166  
-= -Operator 165  
>>-Operator 166
- A
- Abfangen von Fehlern 167  
Abstrakte Klassen 294  
Accessor *siehe* Eigenschaften, Accessor  
    unterschiedliche für Lesen/Schreiben (Eigenschaften) 259  
Accessors: 256  
ActiveX Data Objects *siehe* ADO.NET  
Add 459  
AddHandler 423  
AddressOf 432  
    Anwendungsbeispiel 640  
ADLabelEx-Benutzersteuerelement 916  
ADO  
    Arbeitsweise 1011  
ADO.NET 983  
    Abfragen 994  
    Abfragen generieren 1027  
    Abfragen mit Parametern 1031  
    Änderungen übermitteln 1012  
    Anwendungsbeispiel 1037  
    Begriffserklärungen 994  
    CommandBuilder-Klasse 1019  
    Command-Klasse 994, 999  
    Connection-Klasse 994, 998  
    DataAdapter-Klasse 1002  
    DataReader-Klasse *siehe* DataReader-Klasse  
    DataRelation-Klasse 1021  
    DataSet-Klasse *siehe* DataSet-Klasse  
    DataTable *siehe* DataTable-Klasse  
    Daten ändern 1009  
    Daten modifizieren 994  
    HeckTick (Anwendungsbeispiel) 1037  
    Parameter in Abfragen 1031  
    Programmierung 998  
    Resultsets *siehe* Resultsets  
    SELECT-Anweisung *siehe* SELECT-Anweisung (SQL)  
    Server-Explorer 995  
    SqlCommandBuilder-Klasse 1019  
    SqlCommand-Klasse 999  
    SqlConnection-Klasse 994, 998  
    SqlDataAdapter-Klasse 1002  
    SqlDataReader-Klasse *siehe* DataReader-Klasse  
    Tabellen 997  
    TableAdapter-Klasse 1027

ADO.NET (*Fortsetzung*)  
     unverbundene Daten 1001  
     weitere Möglichkeiten 1037  
 Aktionen (List (Of)) 608  
 Aktivierreihenfolge 60  
 Aktivierungsreihenfolge 774  
 Alpha Centauri 9  
 AndNot 174  
 Anwendungen  
     Mehrgefachstart verhindern 733  
 Anwendungseinstellungen 727, *siehe* Settings  
     automatisch speichern 733  
     Bereiche 729  
     veränderliche 729  
 Anwendungereignisse 734  
     NetworkAvailabilityChanged 736  
     ShutDown 735  
     Startup 735  
     StartUpNextInstance 736  
     UnhandledException 736  
 Anwendungsframework 731  
     Aktivieren 732  
     Authentifizierungsmodus für Benutzer 733  
     Ereignisse *siehe* Anwendungereignisse  
     Herunterfahren, Modus 734  
     Mehrgefachstart verhindern 733  
     Splash-Dialoge 734  
     Windows XP-Stile verwenden 733  
 Anwendungsschichten 743  
     Anwendungslogik 743  
     Business Logic 743  
     Datenschicht 743  
     Geschäftslogik 743  
     Any-Zieltyp 200  
     Argumente *siehe* Parameter  
 ArrayList-Auflistung 565  
     AddRange-Methode 565  
     Arrays, umwandeln in 566  
     Clear-Methode 566  
     Count-Eigenschaft 565  
     RemoveAt-Methode 566  
     Remove-Methode 566  
 Arrays 539  
     ArrayList, aus 566  
     BinarySearch-Methode 550  
     Dimensionen, ändern 542  
     durchsuchen 550  
     Eigenschaften, wichtige 548  
     Enumeratoren 558  
     Funktionsergebnis, als 541  
     Grundsätzliches 540  
     konservieren, nach ReDim 544  
     Konstanten 547  
     Länge ermitteln 549  
     Length-Methode 549  
     mehrdimensional 547  
     Methoden, wichtige 548  
     Mutex-Objekte (Threading) 949  
     Objekte, Betrachtung als 544  
     Parameter, als 541  
     Preserve-Anweisung 544  
     ReDim-Anweisung 542

Arrays (*Fortsetzung*)  
     Reihenfolge ändern, Elemente 550  
     Reverse-Methode 550  
     Rückgabewert, als 541  
     Sortieren 549  
     Sortierung, benutzerdefiniert 551  
     Sort-Methoden 549  
     statische (Beispiel) 243  
     suchen, Elemente, benutzerdefiniert 551  
     Unterschiede zu VB6 164  
     verschachtelt 547  
     Werteverteilung 547  
 ASCII umwandeln in Char-Datentyp 460  
 AscW-Methode 460  
 Assemblies 176  
     Erklärung 196  
     mit mehreren Namespace-Definitionen 182  
     Namen bestimmen für Projekte 181  
     Verweise auf, hinzufügen 177  
 Assembly (CTS) 257, 258  
 Assembly-Eigenschaft 694  
 AssemblyQualifiedName-Eigenschaft 694  
 Assistenten  
     Datenquellen konfigurieren 787  
 Attribute 687  
     AttributeUsage 691  
     benutzerdefinierte ermitteln 700  
     COMClass 690  
     DllImport 691  
     einfaches Beispiel 688  
     Flags 537  
     genereller Umgang 688  
     MarshalAs 691  
     Obsolete 230, 688  
     Serializable 690  
     StructLayout 340  
     Synchronization, zur (Threading) 950  
     VBFixedArray 690  
     VBFixedString 690  
     WebMethod 690  
     zur Laufzeit ermitteln 697  
 Attributes-Eigenschaft 694  
 AttributeUsage-Attribut 691  
 Aufgabe durch Kommentar 29  
 Aufgabenliste 29  
     Aufgabe hinzufügen 29  
     Codekommentar 29  
     Codetoken einrichten 29  
     Navigieren im Code, mit 30  
     Punkt hinzufügen 107  
     Tipps 30  
 Auflistungen  
     ArrayList 565  
     Captures (Reguläre Ausdrücke) 632  
     CollectionBase-Auflistung 568  
     Darstellen in Steuerelementen 777  
     Enumeratoren 558  
     For/Each für benutzerdefinierte 558  
     generische *siehe* Generische Auflistungen  
     Groups (Reguläre Ausdrücke) 632  
     Grundsätzliches 562  
     Hashtable 571

- Auflistungen (*Fortsetzung*)  
 IList-Schnittstelle 570  
 Items (ListBox) 313  
 Key *siehe* Schlüssel, Abruf durch  
 Matches (Reguläre Ausdrücke) 631  
 Queue 587  
 Schlüssel, Abruf durch 571  
 SortedList 590  
 Stack 589  
 typsichere 568, *siehe* Generische Auflistungen  
 vorhandene 565  
 Aufrufliste 832  
 Aufzählungen 533  
 Ausblenden von Projektdateien 22  
 Ausdrücke, reguläre *siehe* Reguläre Ausdrücke  
 Ausgabefenster 30  
     Debugger-Meldungen anzeigen 32  
     lösen 31  
     Navigieren im Code, mit 31  
     Quelle einstellen 30  
     Zeilenumbruch einstellen 31  
 Ausnahmen 167  
     Behandeln mit Try/Catch/Finally 169  
     Codeausführung sicherstellen 172  
     globale Behandlung 736  
     OutOfRangeException 447  
     typabhängig behandeln 170  
 Ausrichtungen 514  
 Aussageprüfer (List (Of)) 610  
 Auswahlrand 101  
 Autokorrektur für intelligentes Kompilieren 79  
 Automatischer Zeilenumbruch 102  
 Automatisches Vervollständigen *siehe* IntelliSense,  
     Vervollständigungsliste  
 AutoResetEvent-Klasse (Threading) 952  
 AutoScroll-Eigenschaft 58
- B**
- Background Compiler 27, 76  
 Base Class Library 198  
 BaseType-Eigenschaft 694  
 BCL 159  
 Bearbeitungshilfen *siehe* IntelliSense  
 Befehlszeilen  
     Space Invaders 217  
 Befehlszeilenanwendungen  
     in .NET Programmieren 217  
 Befehlszeilenargumente auslesen 716  
 BeginUpdate-Methode (ListView) 779  
 Begleitdateien  
     auf Buch-CD 10  
     Einrichtung 10  
     PDF-Dateien Visual Basic .NET Entwicklerbuch 10  
     Softwarevoraussetzungen 3  
     Updates und Errata 9  
 Begrüßungsdialoge *siehe* Splash-Dialoge  
 Beispieldatenbank 992
- Benutzer  
 Authentifizierungsmodus 733  
 Benutzereingaben überprüfen *siehe* Validieren  
 Benutzersteuerelemente 893  
     ADLabelEx 916  
     als Container 903  
     Anordnen von Steuerelementen 907  
     aus vorhandenen 894  
     Basis-Window 918  
     Beschriftung 917  
     Blinken 926  
     Borderline-Eigenschaft 919  
     Browsable-Attribut 901  
     Category-Attribut 901  
     Codegenerierung steuern 929  
     ContainerControl 903  
     Control, Ableiten von 894  
     CreateParams-Eigenschaft 918  
     DefaultValue-Attribut 901  
     Delegieren von Funktionen 904  
     Description-Attribut 901  
     Designer 907  
     Designerreglementierungen 931  
     Dockem mit Dock 908  
     DrawString-Methode 917  
     Eigenschaften als Code serialisieren 929  
     Eigenschaften, werteerbende 929  
     Eigenschaftenfenster steuern 900  
     Ereignisse delegieren 911  
     erstellen von Grund auf 916  
     FocusColoredTextBox, Beispiel 895  
     Größe festschreiben 931  
     Größenbeeinflussung durch Eigenschaften 924  
     Grundfenster erstellen 918  
     Grundlagen 894  
     Initialisieren 908  
     Konstituierende 903  
     Methoden delegieren 911  
     Neuziechnen auslösen 923  
     ohne Vorlage 916  
     Projektmappe für eigene Assembly 904  
     ShouldSerializeXXX-Methode 929  
     Standardwerte für Eigenschaften 929  
     Stil 918  
     Toolbox, einfügen in 896  
     unveränderliche Größe 931  
     Windows-Stil 918  
     zeichnen, nativ 920  
     Z-Reihenfolge 908  
 Beschränkungen 400  
 Betriebssystemaufrufe 480  
 Binäre Suche (Arrays) 550  
 BinaryFormatter (Serialisierung) 661  
 BinarySearch 333  
 BinarySearch-Methode (Arrays) 550  
 BindingNavigator-Klasse 1035  
 BindingSource *siehe* Bindungsquellen  
 BindingSource-Klasse 785  
 Bindungsquellen 784  
 Block-Gültigkeitsbereiche 162  
 Boolean-Datentyp 482  
     numerische Äquivalente 483

Boolean-Datentyp (*Fortsetzung*)  
 String, wandeln in 483  
 Vergleiche 484  
 Borderline-Eigenschaft 919  
 Boxing  
     Details zum 352  
     Schnittstellen, Besonderheiten 354  
     Unboxing 352  
     Wertetypen 349  
 Breite Linienzüge zeichnen 884  
 Brush-Objekt 867  
 Bucket (Hashtable) 585  
 Build 21  
 Business Logic 743  
 ByRef 336  
 Byte-Datentyp 445  
 ByVal 336

**C**

CallBack-Funktion 830  
 Captures-Auflistung (Reguläre Ausdrücke) 625, 632  
 Case-Anweisung 487  
 Casting 343  
 Catch 170  
 CByte 446  
 CDbl 451  
 CDec 451  
 CellTemplate-Eigenschaft (DataGridView) 795  
 CellValidating-Ereignis 801  
 Char-Datentyp 459  
     Werte umwandeln in 460  
 CheckBox-Steuerelement  
     ThreeState-Eigenschaft, Besonderheiten bei 597  
 Child Window 805  
 ChrW-Methode 460  
 CInt 344, 448  
 CLI 198  
 ClientArea-Eigenschaft 869  
 CLng 449  
 Cloning *siehe* Klonen  
 Close-Methode  
     Formulare 775  
 Closing-Ereignis 758  
 CLR 198  
     Execution Engine 198  
     Garbage Collector 198  
     mscoree.dll 198  
 CLS  
     Compliance 444  
 Code  
     Aufgabe aus Kommentar 29  
     aus- und einblenden 103  
     Ausführung bei Fehlern sicherstellen 172  
     automatisch durch Designer 822  
     bearbeiten *siehe* Codeeditor  
     Bearbeiten, Hilfe beim *siehe* IntelliSense  
     Editor reagiert träge 32  
     Fehlerarten im Editor 27

Code (*Fortsetzung*)  
 für Kulturen 499  
 gleicher für mehrere Ereignisse 26  
 Gültigkeitsbereiche durch Blöcke 162  
 inkrementelles Suchen 106  
 Klassencode über Dateien aufteilen 190  
 Lesezeichen 107  
 mehrere Suchergebnisse 107  
 Meldungen 28  
 rechteckig markieren 103  
 Refactoring 86  
 Region 104  
 reguläre Ausdrücke 104  
 Rumpf für Ereignisse einfügen 25  
 Sprung zu Zeilennummer 106  
 statt Formular öffnen 22  
 Suchen und Ersetzen 104  
 Token für Aufgaben 29  
 Umgestalten (Refactoring) 86  
 Vererben von Klassencode 261  
 Warnungen 28  
 Warnungen konfigurieren 28  
 Wiederverwandlung 261  
 Wiederverwenden 290  
 Zugriff auf Settings-Variablen 95

Code Snippets  
 Assembly-Verweise berücksichtigen 130  
 Bibliothek, hinzufügen zur 131  
 Codeausschnittsbibliothek 131  
 Datei-Öffnen-Programmlogik, als 127  
 Eigene erstellen 127  
 Literale Ersetzungen 134  
 Neue Ausschnitte verwenden 133  
 Objektersetzungen 138  
 Parametriesieren 133  
 Verwenden 89  
 XML-Vorlagen 129  
 Codeausschnitte *siehe* Code Snippets  
 Codebereich 101  
 Codedateien  
     verteilt, für eine Klasse 190  
 Codedateien organisieren 23  
 Codedateien, durchsuchen 104  
 Codeeditor  
     Navigieren mit Aufgabenliste, Fehlerliste 30  
     Navigieren mit Ausgabefenster 31  
 Codeeditor 71  
     Abstrakte Klassen, Unterstützung für 308  
     Anzeigen von 72  
     aufrufen aus Projektmappen-Explorer 23  
     Auswahlrand 101  
     Autokorrektur für intelligentes Kompilieren 79  
     Automatischer Zeilenumbruch 102  
     automatisches Codeeinrücken 75  
     automatisches Vervollständigen *siehe* IntelliSense,  
         Vervollständigungsliste  
     Background Compiler 76  
     Code aus- und einblenden 103  
     Codeausschnitte *siehe* Code Snippets  
     Codebereich 101  
     Dokumentationskommentare, benutzerdefiniert 80  
     Ereignisbehandlungsroutinen, Unterstützung bei 415

Codeeditor (*Fortsetzung*)  
 Fehler, Unterstützung beim Korrigieren von 225  
 Fehlerbehebung bei Laufzeitfehlern 77  
 Fehlererkennung 76  
 Gliederungsansicht 103  
 Indikatorrand 101  
 Intellisense 232  
 IntelliSense *siehe* IntelliSense  
 Interfaces, Unterstützung für 304  
 Kennen lernen am Beispiel 38  
 Klassen, Unterstützung für abstrakte 308  
 Laufzeitfehler beheben 77  
 Lesezeichen 104  
 navigieren im Code 102  
 Neue Codedateien anlegen 84  
 Prozeduren-Beschreibungen, benutzerdefiniert 80  
 rechteckige Textmarkierung 103  
 Refactoring 86  
 Region 104  
 Schnittstellen, Unterstützung für 304  
 Smarttags 79  
 Smarttags, Verwendung von 225  
 sonstige Funktionen 101  
 Suchen und Ersetzen 104  
 Tipps für ermüdfreies Arbeiten 71  
 Umgestalten von Code (Refactoring) 86  
 Variablenzuweisung mit Konstanten 150  
 wird langsam 32  
 XML-Kommentare, benutzerdefiniert 80  
 Codefenster  
 Lesezeichen 107  
 CollectionBase-Auflistung 568  
 Add-Methode (Implementierung) 568  
 Collection(Of) (als Alternative) 568  
 Elementzugriff über List 570  
 IList 570  
 Item-Eigenschaft (Implementierung) 568  
 List 570  
 List(Of) (als Alternative) 568  
 Collections *siehe* Auflistungen  
 Columns-Eigenschaft (DataGridView) 791  
 Columns-Eigenschaft (ListView) 778  
 ColumnsHeaderCollection-Auflistung (ListView) 778  
 ColumnType-Eigenschaft (DataGridView) 794  
 COMClass-Attribut 690  
 CommandBuilder-Klasse 1019  
 Command-Klasse 994, 999  
 Common Language Infrastructure 198  
 Common Language Runtime 198  
 Common Type System 199, 201  
 Comparer-Klasse 553  
 benutzerdefiniert 556  
 Connection-Klasse 994, 998  
 Constraints 400  
 Container für Steuerelemente 44  
 ContainerControl 44  
 Continue-Anweisung 185  
 ControlCollection 826  
 ControlDesigner 45  
 ControlNativeWindow 830  
 Convenience-Patterns 377  
 Convert.ToByte 446  
 Convert.ToDecimal 451  
 Convert.ToDouble 451  
 Convert.ToInt16 447  
 Convert.ToInt32 448  
 Convert.ToInt64 449  
 Convert.ToSByte 446  
 Convert.ToSingle 450  
 Convert.ToInt32 448  
 Convert.ToInt64 449  
 Convert-Klasse 344  
 Cooperative Multitasking 939  
 Covers  
 Begleitdateien 41  
 Covers – kennen lernen der Entwicklungsumgebung, mit 38  
 Covers: 39  
 CreateParams-Eigenschaft 918  
 CSByte 446  
 CShort 447  
 CSng 450  
 CTS 199, 201  
 CType 344  
 für eigene Klassen/Strukturen 386  
 CUInt 448  
 CULng 449  
 CultureInfo 498  
 CultureInfo-Klasse 457  
 Currency (Zeitnähe bei Bindungen) 784  
 CurrencyManager-Klasse 784  
 CurrentCellChanged-Ereignis (DataGridView) 796  
 CurrentCell-Eigenschaft (DataGridView) 796  
 CustomFormatProvider 517

## D

DataAdapter-Klasse 1002  
 DataGridViewCellCollection-Auflistung 794  
 DataGridViewCell-Objekt 796  
 DataGridViewColumnCollection-Auflistung 791  
 DataGridView-Steuerelement 782  
 Aktuelle Zelle ermitteln 796  
 Bearbeitungsmodus, prüfen auf 801  
 CellTemplate-Eigenschaft 795  
 CellValidating-Ereignis 801  
 Columns-Eigenschaft 791  
 ColumnType-Eigenschaft 794  
 Curorsprungverhalten ändern nach Eingabetaste 803  
 CurrentCellChanged-Ereignis 796  
 CurrentCell-Eigenschaft 796  
 DataGridViewCellCollection-Auflistung 794  
 DataGridViewCell-Objekt 796  
 DataGridViewColumnCollection-Auflistung 791  
 Datenquellen konfigurieren 787  
 Designer-Unterstützung 786  
 Editieren von Spalten verhindern 799  
 Eingaben verarbeiten 797  
 Formatieren von Inhalten 797  
 Häufige Aufgaben (Designer) 792  
 IsInEditMode-Eigenschaft 801

DataGridView-Steuerelement ( <i>Fortsetzung</i> )	Daten ( <i>Fortsetzung</i> )
IsNewRow-Eigenschaft 800	erfassen, Anwendungsbeispiel 761
Neue Zeile, testen auf 800	Listendarstellung 777
Nur-Lesen-Spalten 799	Strukturieren in Anwendungen 743
Parse von Eingaben 797	Datenbanken
ReadOnly-Spalte 799	Abfragen generieren 1027
Spalten konfigurieren 791, 793	Änderungen übermitteln 1012
Spalten programmtechnisch ansprechen 791	Beispieldatenbank 992
Spaltenindex aus Spaltennamen ermitteln 794	Grundsätzliches 992
Spaltenkopfnamen bestimmen 794	SELECT-Anweisung <i>siehe</i> SELECT-Anweisung (SQL)
Spaltentypen bestimmen 794	Tabellen 997
Unterscheiden bearbeiten/neu eingeben 800	Datenbindung 784
Validieren von Eingaben 801	Anwendungsbeispiel 790
Verhindern falscher Eingaben 801	BindingSource-Klasse 785
Vorgehensweisen, grundsätzliche 783	Blättern in Datenquelle 785
Vorlagen 795	CurrencyManager-Klasse 784
Zeilenvalidierungen 801	komplexe 784
Zelle ermitteln 794	Objekte, an 788
Zelle, aktuelle ermitteln 796	PropertyManager-Klasse 784
Zellen formatieren 797	Datenerfassung, mit Formularen 754
Zellencursorverhalten ändern 803	Datenquellen konfigurieren 787
Zellenspeicherung 796	Datenrückgabe aus Formularen 759
Zellenvalidierungen 801	Datenschicht 743
Zellenvorlagen 795	Datentypen
Zellenwechsel mit Eingabetaste 803	.NET-Synonyme 440
Zelleselektionsänderung feststellen 796	Berechnungen durch Prozessor 442
DataReader-Klasse 999	Boolean <i>siehe</i> Boolean-Datentyp
HasRow-Eigenschaft 1000	Byte <i>siehe</i> Byte-Datentyp
Read-Methode 1000	Char <i>siehe</i> Char-Datentyp
Zeilen lesen 1000	CLS-Compliance 444
Zeilen, auf vorhandene prüfen 1000	Date <i>siehe</i> Date-Datentyp
DataRelation-Klasse 1021	Decimal <i>siehe</i> Decimal-Datentyp
DataRow-Klasse 1004	deklarieren und definieren 442
DataSet-Klasse 1021	Double <i>siehe</i> Double-Datentyp
typisierte erstellen 1021	Einführung in 440
typisierte, Grundsätzliches 1025	Ermitteln des Typs 693
typisierte, Vorteil von 1022	generisch, benutzerdefiniert <i>siehe</i> Generische Datentypen
DataTable-Klasse 1001	generische 595
DataRow-Klasse 1002	Integer <i>siehe</i> Integer-Datentyp
Daten auffüllen 1003	Long <i>siehe</i> Long-Datentyp
Datenzeile 1002	null-bar (Wertetypen) 429
Füllen mit Daten 1003	nullbare 595
Funktionsweise 1002	numerische 442
Rows-Auflistung 1002	numerische in String wandeln 454
Zeilen abrufen 1004	numerische, Rundungsfehler 451
Zeilenauflistung 1002	numerische, Überblick über 445
Date-Datentyp 488	primitive 440, <i>siehe</i> Primitive Datentypen
formatieren, für Textausgabe 497	Rundungsfehler 451
Formatzeichenfolgen 501	SByte <i>siehe</i> SByte-Datentyp
ParseExact-Methode 493	Short <i>siehe</i> Short-Datentyp
Parse-Methode 493	Single <i>siehe</i> Single-Datentyp
rechnen mit 489	String <i>siehe</i> String-Datentyp
String, umwandeln in 493	TimeSpan 489
Dateinamen standardisieren 472	ToString 456
Dateioperationen (My-Namespace) 724	Typermittlung 693
Dateioperationen programmieren 709	Typparameter 398
Dateitypen von Projekten 22	UInteger <i>siehe</i> UInteger-Datentyp
Daten	ULong <i>siehe</i> ULong-Datentyp
bearbeiten, Anwendungsbeispiel 761	umwandeln 454
Binden an Komponenten 784	Umwandlungsversuch 458
Darstellen in Steuerelementen 777	
Eingeben 768	

Datentypen (*Fortsetzung*)  
 UShort *siehe* UShort-Datentyp  
 Datentypengrößen 145  
 DateTime 346, *siehe* Date-Datentyp  
 DateTimeFormatInfo-Klasse 512  
 DateTimeStyle 494  
 DateTimeStyles.AdjustToUniversal 494  
 DateTimeStyles.AllowInnerWhite 494  
 DateTimeStyles.allowLeadingWhite 494  
 DateTimeStyles.AllowTrailingWhite 494  
 DateTimeStyles.AllowWhiteSpaces 495  
 DateTimeStyles.NoCurrentDateDefault 495  
 DateTimeStyles.None 495  
 Datum  
     berechnen 489  
     Darstellungsstile 494  
     formatieren 497  
     Umwandlungsoptionen 494  
 Debuggen 244  
 Debug-Klasse 363  
 Debug-Konfigurationseinstellungen 100  
 Decimal 145  
 Decimal-Datentyp 451  
     Add 459  
     Floor-Funktion 459  
     formatieren, für Textausgabe 497  
     Formatzeichenfolgen 501  
     Negate-Funktion 459  
     Remainder-Funktion 459  
     Round-Funktion 459  
     spezielle Funktionen 459  
     Truncate-Funktion 459  
 Delegaten 432  
     AddressOf 432  
     asynchron aufrufen 436  
     Asynchron aufrufen 981  
     Beispielanwendung 639  
     Kommunikation in Formularen mit 763  
     Methoden als neuer Thread 436  
     Prozeduren als neuer Thread 436  
     Thread, als 981  
 Dequeue-Methode 588  
 Deserialisierung 660  
 Designer  
     aufrufen aus Projektmappen-Explorer 23  
     automatische Codegenerierung 822  
     Benutzersteuerelemente entwerfen 907  
     Codegenerierung 817  
     ControIDDesigner 45  
     DataGridView-Unterstützung 786  
     Datenquellen konfigurieren 787  
     Funktionen von Steuerelementen 45  
     für Formulare, Funktionsweise 45  
     Gestaltung, mit, am Fallbeispiel 43  
     Größe von Steuerelementen 45  
     Guidelines 44  
     Häufige Aufgaben 792  
     Kennenslernen am Beispiel 38  
     Layout-Funktionen für Formulare 68  
     Layout-Modus für Formulare 45  
     Position von Steuerelementen 45  
     Raster in Formularen 45  
 Designer (*Fortsetzung*)  
 Referenzsteuerelement beim Anordnen 46  
 Smarttags für Steuerelemente 47  
 Steuerelemente dynamisch Anordnen 48  
 Steuerelemente positionieren 43  
 Steuerelemente selektieren, unsichtbare 809  
 Steuerelemente, Aufgaben, häufige 47  
 Steuerelemente, Randabstände 44  
 Tastenkürzel 70  
 Designer Host 45  
 Designer-Code 817  
 Designer-Komponenten 769  
 Dialog *siehe* Modale Formulare  
 Dialogergebnisse 756  
 DialogResult-Eigenschaft 756  
 Dim 442  
 DirectoryInfo-Klasse 709  
 Disassembler 238  
 Dispose 355  
 Dispose 364  
 Dispose 839  
 Dispose 848  
 Dispose-Methode  
     Formulare 775  
 DllImport-Attribut 691  
 Dock-Eigenschaft 908  
 Dokumentationskommentare, benutzerdefiniert 80  
 Dokumentfenster 18  
     wechseln zwischen 20  
 Dokumentklassen 805  
 Doppelpufferung 879  
 DotNetCopy 705  
     /StartTime-Option 709  
     /Silent-Option 709  
     Bedienung 706  
     Funktionsweise, Abstrakt 711  
     Kopierverhalten einstellen 707  
     Optionen 709  
 Double Buffering 879  
 Double-Datentyp 450  
     formatieren, für Textausgabe 497  
     Formatzeichenfolgen 501  
     Parse-Methode 454  
     Vergleichen 454  
 DrawString-Methode 876  
 Dual Core-Prozessoren 933  
 Duotrigesimalsystems 330  
 Dynamische Hilfe 32  
     Inhalt anpassen 33

## E

Edit & Continue 239  
 Editieren, Code *siehe* Codeeditor  
 Editor *siehe auch* Codeeditor  
 Eigenschaften 246  
     Accessor 247, 267  
     Accessor, unterschiedliche für Lesen/Schreiben 259  
     auslesen 247

Eigenschaften ( <i>Fortsetzung</i> )	Entwicklungsumgebung ( <i>Fortsetzung</i> )
Default 252	Dynamische Hilfe <i>siehe</i> Dynamische Hilfe
definieren 247	Eigenschaftenfenster <i>siehe</i> Eigenschaftenfenster
definieren (Wertezuweisung) 247	Einführung 13
ermitteln 247	Einsatz mehrere Monitore 109
Get-Accessor 247	Einstellungen sichern 112
Leselogik definieren 247	Fehlerliste <i>siehe</i> Fehlerliste
Let-Anweisung 251	Fenstereinstellungen zurücksetzen 112
Margin 44	Fensterlayout zurücksetzen 16
MustOverride 296	Hilfe 122
nur lesen 249	Klassenansicht <i>siehe</i> Klassenansicht
nur schreiben 249	Konfigurationseinstellungen 100
Padding 44	MDI (Multi Document Interface), anzeigen als 18
Parameter überladen 251	Nachrichten-Channel 14
Parametern, mit 250	Originalzustand 112
ReadOnly 249	Originalzustand wiederherstellen 115
schreiben, Werte 247	Registerkartengruppen 18
Schreiblogik definieren 247	Server-Explorer 995
Set-Accessor 247	Startseite 13
Set-Anweisung 251	Tastenkombinationen 34
Standardeigenschaften 252	Toolfenster <i>siehe</i> Toolfenster
statische 252	typisierte DataSets erstellen 1021
überladen 251	Umgang mit Fenstern 18
Überschatten 318	
überschreiben 272	Enum
Überschreiben (Polymorphie) <i>siehe</i> Polymorphie	Praxisbeispiel, Einsatz im 288
Überschreibungszwang 296	Zahlen-Datentyp, konvertieren aus 536
Variablen, Vergleich zu öffentlichen 255	Enumeratoren 558
veraltete Markieren 230	benutzerdefiniert 559
virtuelle 294	Enums 533
Werte auslesen 247	direkte Wertebestimmung 534
Werte, zuweisen an 247	Dubletten 535
Wertzuweisung 247	Elementtyp ermitteln 535
WriteOnly 249	Flags 537
Zugriffsmodifizierer 249	Flags, Abfrage von 538
Eigenschaften eines Projektes 23	Kombinationen, Abfrage von 538
Eigenschaftenfenster 25	Kombinieren von Werten 537
Ereignisse 25	konvertieren 536
Selektieren von Steuerelementen 60	String-Datentyp, konvertieren in 536
Einblenden von Projektdateien 22	Equals-Methode 315, 318, 582
Einführung 3	Hinweis zum Überschreiben 582
Eingabefehler anzeigen	Ereignisbehandlungsroutinen erstellen 26
Formulare, in 768	Ereignisparameter 420
Tabellenzeilen, in 801	EventArgs 422
Tabellenzellen, in 801	EventArgs.Empty 422
Eingaben überprüfen <i>siehe</i> Validieren	leere 422
ElseIf-Anweisung 485	Sender 420
Elternfenster 806	Ereignisse 413, 827
Empty (EventArgs) 422	AddHandler 423
End Select-Anweisung 487	Anwendungen, für <i>siehe</i> Anwendungsereignisse
EndUpdate-Methode (ListView) 779	Auslösen 418
Enqueue-Methode 588	Auslösen, durch Überschreiben von Onxxx 419
Entwicklerbuch	Behandeln (Handles) 415
IntelliLinks 9	Behandlungscode erstellen 25
Entwicklungsumgebung 11	Benutzerdefinierte, Beispiel 414
alle Projektdateien anzeigen 22	Code, gleicher für mehrere Ereignisse 26
als Multi Document Interface (MDI) 18	Delegaten 432
auf einen Blick 16	dynamische Einbinden 423
Aufgabenliste <i>siehe</i> Aufgabenliste	Ereignisbehandlungsroutinen 415
Ausgabefenster <i>siehe</i> Ausgabefenster	Feuern (ugs.) 418
beste Monitorauflösung 16	Formulare, von <i>siehe</i> Formular- und
Dokumentfenster 18	Steuerelementereignisse

Ereignisse (*Fortsetzung*)  
 Handles 415  
 Objekte zum Konsumieren deklarieren 415  
 Onxxx überschreiben 419  
 PaintEventArgs-Klasse 865  
 Parameter *siehe* Ereignisparameter  
 RaiseEvent 419  
 RemoveHandler 423  
 Rumpf für Behandlungscode einfügen 25  
 Steuerelementen, von *siehe* Formular- und Steuerelementereignisse  
 Verdrahten 415  
 verwalten durch Eigenschaftenfenster 25  
 WithEvents 415  
 Erl (VB6) 167  
 ErrorProvider-Komponente 769  
 Erzwungene Typsicherheit 80  
 EventInfo-Klasse 696  
 Events *siehe* Ereignisse  
     fire (col.) *siehe* Ereignisse, Auslösen  
     raise *siehe* Ereignisse, Auslösen  
 Exception  
     OutOfRange 446  
 Exceptions 169  
     Codeausführung sicherstellen 172  
     typabhängig behandeln 170  
 Execution Engine 198

## F

False 482  
 Family (CTS) 257, 258  
 FamilyOrAssembly (CTS) 257, 258  
 Fehler 167  
     Auffangen mit Try/Catch 169  
     Behandeln 167  
     Codeausführung sicherstellen 172  
     kulturspezifisch, bei Konvertierungen 500  
     typabhängig behandeln 170  
 Fehler abfangen  
     global 736  
 Fehlerliste 27  
     Background Compiler 27  
     Meldungstypen 27  
     Navigieren im Code, mit 30  
     Tipps 30  
 Fehlertypen im Codeeditor 27  
 Fehlerüberprüfung im Hintergrund 27  
 Fenster *siehe* Formulare  
 Feuern, Ereignisse (ugs.) 418  
 FieldInfo-Klasse 696  
 FIFO-Prinzip 587  
 FileInfo-Klasse 709  
 FileStream 370  
 Finalize 355, 359  
 Finalize-Methode 318  
 Finally 172  
 Flächen 867  
 Flächen füllen 868

flaches Klonen 666  
 Flags-Attribut (Enums) 537  
 Flags-Enum (Flags-Aufzählungen) 537  
 Flimmerfreie Darstellung (Formulare, Steuerelemente) 878  
 FlowLayoutPanel 49  
 Fokussierung, Reihenfolge *siehe*  
     Aktivierungsreihenfolge  
 fomatieren  
     Zahlen und Datumswerte 497  
 For/Each-Anweisung  
     Auflistungen, für benutzerdefinierte 558  
 For/Next-Schleifen  
     Fließkommazahlen 453  
 Format Provider 456  
     Ausrichtungen 514  
     benutzerdefiniert 517  
     CultureInfo 498  
     DateTimeFormatInfo-Klasse 512  
     Einführung 497  
     Fehler vermeiden, kulturabhängig 500  
     Formatierung, gezielte 511  
     Formatzeichenfolgen 501  
     ICustomFormatter-Schnittstelle 527  
     IFormatProvider-Schnittstelle 527  
     Länderkürzel 499  
     NumberFormatInfo-Klasse 511  
 formatieren  
     Format Provider, mit 511  
     IFormattable-Schnittstelle 530  
     kombiniert 513  
     kulturabhängig 498  
 Formatieren, Zahlenwerte 287  
 Formatvorlagen 504  
 Formatzeichenfolge 219  
 Formatzeichenfolgen 414, 501  
     Formatvorlagen 504  
     vereinfacht 510  
     Zeitformatierung, für 510  
 Formel Parser 639  
 Formeln berechnen 634  
 Formular  
     Code statt Designer öffnen 22  
 Formular- und Steuerelementereignisse  
     Activated 838  
     auslösende Instanz 834  
     Click 839, 848  
     Closed 838  
     Closing 838  
     CreateControl 837, 847  
     Deactivate 838  
     DoubleClick 840, 849  
     Enter 845, 854  
     GotFocus 845, 854  
     HandleCreated 837, 847  
     HandleDestroyed 848  
     Invalidated 843, 853  
     KeyDown 841, 850  
     KeyPress 842, 851  
     KeyUp 842, 851  
     Layout 844, 853  
     Leave 845, 855

Formular- und Steuerelementereignisse (*Fortsetzung*)  
Load 837  
LocationChanged 842, 851  
LostFocus 845, 855  
MouseDown 839, 848  
MouseEnter 840, 849  
MouseHover 840, 849  
MouseLeave 841, 850  
MouseMove 841, 850  
MouseUp 840, 849  
MouseWheel 841, 850  
Move 842, 851  
OnHandleDestroyed 839  
Paint 844, 854  
PaintBackground 844, 854  
Resize 843, 852  
SetVisibleCore 838  
SizeChanged 843, 852  
WndProc 833, 847

Formulare 741  
Abbrechen (Tastatur) 774  
AcceptButton 64  
Aktivierreihenfolge von Steuerelementen 60  
Aktivierungsreihenfolge 774  
Anwendungsframework, Hilfe durch 731  
automatisches Scrollen 57  
AutoScroll-Eigenschaft 58  
CancelButton 64  
ClientArea-Eigenschaft 869  
Close-Methode 775  
Closing-Ereignis 758  
ControlCollection 826  
Datenbankgebundene 1033  
Datenerfassung mit 754  
Designer-Code 817  
Dispose-Methode 775  
Eigenschaften in Settings speichern 97  
Eingabefehler anzeigen 768  
Eingaben überprüfen *siehe* Validieren,  
    Formulareingaben  
    entsorgen 775  
Ereignisse *siehe* Formular- und  
    Steuerelementereignisse  
erstellen 823  
Fensterausschnitt, sichtbarer 869  
Fensterinnenbereich 869  
flimmerfreie Darstellung 878  
gestalten *siehe* Designer  
Größe anpassen, programmtechnisch 882  
Größe von Steuerelementen 45  
Grundverhalten 880  
Hide-Methode 775  
Hinzufügen zu Projekten 65  
Inhalte zeichnen in OnPaint 864  
InitializeComponent 824  
Instanziieren, ohne 715  
IsMdiContainer-Eigenschaft 806  
Klassennamen-Anpassung beim Umbenennen 88  
Kommunikation zwischen 763  
Layout-Funktionen 68  
Layout-Logik aussetzen 825  
Layout-Modus 45

Formulare (*Fortsetzung*)  
MDI-Anwendungen *siehe* MDI-Anwendungen  
MDI-Container, bestimmen als 806  
modal darstellen 757  
modale 755, *siehe* Modale Formulare  
Neuzeichnen auslösen 923  
OnPaint-Methode 864  
Rasterung im Designer 45  
Ressourcen freigeben 756  
ResumeLayout 825  
Schließen verhindern 758  
Schließen, richtiges 775  
Schnellzugriffstasten 64  
ShowDialog 757  
Splash-Dialoge 734  
Steuerelemente positionieren *siehe* Designer,  
    Steuerelemente positionieren  
SuspendLayout 825, 829  
Tabulatorreihenfolge von Steuerelementen 60  
Tastenkürzel für Layout von Steuerelementen 70  
thread-sicher 975  
unsichtbar machen 775  
Using 757  
Vererben 813  
WMClose-Methode 776  
WndProc-Methode 776  
Zeichnen von Inhalten 864  
zerstören 823  
Z-Reihenfolge 908  
Zugriffsmodifizierer 820

Formulare: 45  
Framework *siehe* .NET-Framework  
    Aufbau 200  
    Betriebssystemvoraussetzungen 5  
Framework Class Library 198  
Friend 257, 258  
FullName-Eigenschaft 694  
FullRowSelect-Eigenschaft (ListView) 777  
Funktionen  
    optionale Parameter 233  
    Signatur 232  
    statische 223  
    überladen 231  
    veraltete markieren 230  
Funktionen auswerten 634  
Funktionszeiger 640, *siehe* Delegaten

## G

Garbage Collector 198, 357  
    Generationen 358  
GC 357  
GDI 863  
GDI+ 863  
    Breite Linienzüge 884  
    Brush-Objekt 867  
    ClientArea-Eigenschaft 869  
    Demo 869  
    Doppelpufferung 879

GDI+ (*Fortsetzung*)  
 Double Buffering 879  
 DrawString-Methode 876  
 DrawXXX-Methoden 868  
 Fensterausschnitt, sichtbarer 869  
 Fensterinnenbereich 869  
 FillXXX 868  
 Flächen füllen 868  
 GraphicsPath-Objekt 886  
 HatchBrush-Objekt 868  
 Inch 869  
 Koordinaten 868  
 Koordinatengenauigkeit 869  
 LinearGradientBrush-Objekt 868  
 Linienzüge zeichnen 868  
 Linienzüge, benutzerdefiniert 886  
 Millimeter 869  
 PageUnit-Eigenschaft 869  
 PathGradientBrush-Objekt 868  
 Pen-Objekt 867  
 Pixel 869  
 PointF-Struktur 868  
 Point-Struktur 868  
 Polygone 886  
 RectangleF-Struktur 868  
 Rectangle-Struktur 868  
 Seiteneinteilung 869  
 Seitenkalierung 869  
 SizeF-Struktur 868  
 Size-Struktur 868  
 SolidBrush-Objekt 868  
 Text einpassen 876  
 TextureBrush-Objekt 868  
 VisibleClipBounds-Eigenschaft 869  
 Geerbte Formulare 813  
 Gehe zu 106  
 Generationen 358  
 Generische Auflistungen 599  
 Collection(Of) 600  
 Dictionary(Of) 601  
 KeyedCollection(Of) *siehe* KeyedCollection(Of)-Auflistung  
 LinkedList(Of) \t 605  
 List(Of) 601  
 Queue(Of) 601  
 ReadOnlyCollection(Of) 601  
 SortedDictionary(Of) 601  
 SortedList(Of) 602  
 Stack(Of) 602  
 Generische Datentypen 393, 595  
 Beschränken auf Wertetypen 409  
 Beschränkungen 400  
 Beschränkungen kombinieren 409  
 Constraints 400  
 JITter, Umsetzung durch 398  
 Konstruktorbeschränkungen 408  
 Notwendigkeit, Beispiel 394  
 Schnittstellenbeschränkungen 405  
 Typparameter 398  
 vererben 410  
 Geschäftslogik 743  
 Gestalten von Formularen *siehe* Designer

Get-Accessor (Eigenschaften) 247  
 GetCustomAttributes-Methode 694  
 GetEvent-Methode 694  
 GetEvents-Methode 694  
 GetField-Methode 694  
 GetFields-Methode 694  
 GetHashCode-Methode 582  
 GetHashCode-Methode 318  
 GetKeyForItem-Methode 602  
 GetMember-Methode 694  
 GetMembers-Methode 694  
 GetProperties-Methode 694  
 GetProperty-Methode 694  
 GetType  
 Methode, als 693  
 GetType-Methode 318  
 Gleichzeitigkeit 933  
 Gliederungsansicht 103  
 Global-Anweisung 188  
 Grafiken  
 volatile 864  
 Graphics Device Interface *siehe* GDI  
 Graphics-Klasse 865  
 GraphicsPath-Objekt 886  
 GridLines-Eigenschaft (ListView) 777  
 Groups-Auflistung (Reguläre Ausdrücke) 632  
 Gruppen (Reguläre Ausdrücke) 622  
 Guidelines 44  
 Gültigkeitsbereiche 161  
 Block-Gültigkeitsbereiche 162  
 Formular-Ebene 162  
 Klassen-Ebene 162  
 lokale Variable 162  
 Member-Variablen 162  
 Modul-Ebene 162  
 öffentliche Variablen 161  
 public 161

## H

Haltepunkte 244  
 Haltepunkte setzen 101  
 Handles-Anweisung 415  
 Hashtable-Auflistung 571  
 Anwenden 572  
 Aufzählungen in 585  
 Bucket 585  
 DictionaryEntry 585  
 enumerieren 585  
 For/Each für 585  
 Funktionsweise 578  
 KeyedCollection (als Alternative) 587  
 Load-Faktor 578  
 Schlüssel-Klassen, benutzerdefiniert 581  
 typsichere 585  
 Verarbeitungsgeschwindigkeit 575  
 Zugriffszeiten 577  
 HasRow-Eigenschaft 1000  
 HatchBrush-Objekt 868

HeckTick (Anwendungsbeispiel, ADO.NET) 1037  
 Hide-Methode 775  
 HideSelection-Eigenschaft (ListView) 777  
 Hilfe 122  
 Hintergrunderkennung von Fehlern 76  
 Hintergrundkomplilierung 27  
 http-Referenzen:  
     http://activedevelop.de 9  
     http://entwicklerbuch.net 9  
     http://links.entwicklerbuch.net 9  
 HyperThreading 933

**I**

IComparable-Schnittstelle 553  
 IComparer-Schnittstelle 556  
 ICustomFormatter-Schnittstelle 527  
 IDE (Integrated Development Environment) *siehe*  
     Entwicklungsumgebung  
 IDisposable 369  
 IEnumerable-Schnittstelle 559  
 If/Then/Else-Anweisung 485  
 IFormatProvider-Schnittstelle 527  
 IFormattable-Schnittstelle 530  
 ILDASM 238  
 IList-Schnittstelle 570  
 IML 199, 237  
     Code untersuchen 238  
     Codeanalyse 240  
     Disassembler 238  
     ILDASM 238  
     Konstruktoren 241  
     Umsetzung in nativen Code 237  
 Implementierungsvorschriften *siehe* Schnittstellen  
 Implements 300  
 Implizite Konvertierung 300  
 Imports 179, 456  
 Inch 869  
 Indikatorrand 101  
 Infinity 457  
 Infodialog (beim Starten einer Anwendung) *siehe*  
     Splash-Dialoge  
 Informationsbeschaffung 122  
 Inheritance 261  
 Inherits 288  
 INI-Dateien, Ersatz für 727  
 InitializeComponent 824  
 inkrementelles Suchen 106  
 INSERT-Anweisung (SQL) 1009  
 Instanziieren  
     Wertetypen 336  
 Instanzieren von Klassen, Erklärung 216  
 Instanzieren von Objekten 227  
 Instanzierung  
     Objektspeicher 275, 327  
     Speicher für Objekte 275, 327  
 Instr-Funktion 467  
 InstrRev-Funktion 467  
 Int16 *siehe* Short-Datentyp

**J**

JITter 198, 199, 237  
 Generische Datentypen, Umsetzung von 398  
 Joker 613  
 Just in time-Komplilierung 198

# K

Key (Auflistungen) *siehe* Schlüssel (Auflistungen)  
KeyedCollection(Of )-Auflistung  
    GetKeyForItem-Methode 602  
    Schlüssel ermitteln 602  
    Serialisierungsfehler 602  
    Serialisierungsprobleme 680  
    Workaround bei Serialisierungsproblemen 684  
Kindfenster 805  
Klassen  
    .cctor 239  
    .ctor 239, 267  
    Abstrakt, als solche definieren 295  
    abstrakte 294  
    Analysieren 694  
    Basisklasse erreichen 236  
    BindingSource 785  
    CLS-Compliance 444  
    Code über mehrere Codedateien aufteilen 190  
    Console 217  
    Convert 344  
    CurrencyManager 784  
    DirectCast 348  
    Dispose 355, 364  
    eigene, einfaches Beispiel 220  
    Eigenschaften 246  
    Einführung 193  
    Ereignisse *siehe* Ereignisse  
    Ereignisse verknüpfen 415  
    Ereignisse, Instanzieren zum Konsumieren von 415  
    Ereignisse, Parameter für 422  
    Ermitteln des Typs 693  
    EventArgs 422  
    explizite Konvertierung, implementieren 386  
    Finalize 355  
    Formel Parser 639  
    generisch, benutzerdefiniert *siehe* Generische  
        Datentypen  
    geteilte (Partial) 190  
    Hierarchie unterbrechen 320  
    IML, Umsetzung 267  
    Implementierungsvorschriften *siehe* Schnittstellen  
    Implements 300  
    implizite Konvertierung, implementieren 386  
    Instanz verweist auf Nothing 328  
    Instanzen, prüfen auf Gleichheit 315  
    Instanzieren 216, 227  
    Interfaces *siehe* Schnittstellen  
    Kommunikation zwischen *siehe* Ereignisse  
    Konstruktor 227  
    Konstruktoren überladen 231  
    Konstruktoren, parametrisierte 269  
    Konstruktorzwang 237  
    Managed Heap 275, 327  
    Me 236, 293  
    Member 222  
    Member über Schnittstellen erreichen 571  
    Methoden überschreiben 271  
    Modul 324  
    MustInherit 295

Klassen (*Fortsetzung*)  
     MyBase 236, 293  
    MyClass 293  
    New 236  
    Object 311  
    Operatoren, benutzerdefinierte *siehe*  
        Operatorenprozeduren  
    Overridable 272  
    Overrides 271  
    partielle (Designercode) 819  
    Polymorphie 261, *siehe* Polymorphie  
    Prinzip als Analogie 216  
    Prinzip am anfachen Beispiel 218  
    private Member, indirekt zugreifen auf 571  
    PropertyManager 784  
    Random 350  
    Schnittstellen *siehe* Schnittstellen  
    Schnittstellen einbinden, mehrere 310  
    Schnittstellen implementieren 300  
    serialisieren 657  
    Singleton 324  
    Speicher für Objekte 275, 327  
    Speichern von Inhalten 657  
    Standardkonstruktor 237, 268  
    statische Konstruktoren 241  
    Sub New 227, 231  
    Type 692  
    Typermittlung 693  
    Vererbung 261  
    Wiederverwendung durch Vererbung 261  
    Wrapper (Betriebssystemaufrufe) 480  
Klassenansicht 33  
Klassendesigner 23  
Klonen 666  
Kombinierte Formatierungen 513  
Komplizieren  
    Just in time 198  
Komponenten 769  
    BackgroundWorker 978  
    ErrorProvider 769  
Komponentenfach 769  
Konfigurationsdateien 727  
Konsolenanwendungen 216  
Konstanten 150  
Konstituierende Steuerelemente 904  
Konstruktoren  
    .cctor 239  
    .ctor 239  
    aus Konstruktoren aufrufen 236  
    automatisch durch VB 268  
    Erzwingen in Generischen Datentypen 408  
    IML, in 241  
    Notwendigkeit 237  
    parametrisierte 269  
    Standardkonstruktor 268  
    statische 241  
    Strukturen, bei 336  
    Syntaxfehler 236  
    überladen 231  
    überladene aufrufen 236  
    Überschreiben (Polymorphie) *siehe* Polymorphie

Konvertieren  
 Arrays zu ArrayList 566  
 Boolean in String 483  
 Byte-Datentyp, in 446  
 Date in String 493  
 Datum in Zeichenkette 493  
 Decimal-Datentyp, in 451  
 DirectCast 348  
 Double-Datentyp, in 451  
 Enums 536  
 Erweitern von Typen 387  
 Fehler bei numerischen Typen 452  
 Fehler vermeiden, kulturabhängig 500  
 Formatvorlagen 504  
 Integer-Datentyp, in 448  
 kulturabhängige Fehler vermeiden 500  
 Long-Datentyp, in 449  
 Parse 347  
 ParseExact 347  
 SByte-Datentyp, in 446  
 Short-Datentyp, in 447  
 Single-Datentyp, in 450  
 String in Boolean 483  
 String in Date 493  
 UInteger-Datentyp, in 448  
 ULong-Datentyp, in 449  
 UShort-Datentyp, in 447  
 Verkleinern von Typen 387  
 versuchen 458  
 versuchen (TryParse) 347  
 Wert in Zeichenkette 454  
 Zeichenkette in Datum 493  
 Zeichenkette in Datum (aus Formatvorlagen) 493  
 Zeichenkette in Wert 454  
 Konvertierung, von Datentypen 343  
 Konvertierungen  
     explizite, implementieren benutzerdefiniert 386  
     implizite, implementieren benutzerdefiniert 386  
 Koordinaten 868  
     Skalierung 869  
 Koordinatengenauigkeit 869  
 Kultur, nicht bestimmt 457  
 Kulturbabhängige Formate 456  
 Kulturinformationen 498  
 Kulturkürzel 499  
 Kurzschlussauswertung 174  
 Kurzschlussauswertungen 174

## L

Länderkürzel 499  
 Layer *siehe* Anwendungsschichten  
 Left-Funktion 466  
 Leistungsmesser 481  
 Len-Funktion 465  
 Length-Methode (Arrays) 549  
 Lesezeichen 104  
 Lesezeichen (Programmcode) 107  
 Let-Anweisung (Variablenzuweisung) 251

LIFO-Prinzip 589  
 Like-Operator 484  
 LinearGradientBrush-Objekt 868  
 Linien 867  
 Linienzüge zeichnen 868  
 LinkedList(Of )-Auflistung 605  
     AddAddFirst-Methode 605  
     AddAfter-Methode 605  
     AddBefore-Methode 605  
     AddLast-Methode 605  
     FindLast-Methode 605  
     Find-Methode 605  
     First-Eigenschaft 605  
     Last-Eigenschaft 605  
     RemoveFirst -Methode 606  
     RemoveLast -Methode 606  
     Remove-Methode 605  
 Links zu Aktualisierungen 9  
 List(Of )-Auflistung  
     Aktionen (Actions) 608  
     Aussageprüfer (Predicates) 610  
     For/Each-Ersatz 608  
     Sortierungssteuerung-Ersatz 609  
     Suchsteuerung 610  
     Vergleicher (Comparisons) 609  
 ListBoxItem 317  
 ListBox-Steuerelement 312  
     ausgewählter Eintrag 317  
     Eintrag entfernen 317  
     Elemente identifizieren 317  
     Items-Auflistung 313  
     Remove 317  
     SelectedItem 317  
     selektiertes Element 317  
     Texteinträge aus Objekten 312  
 Listen 540  
 Listendarstellung 777  
 ListViewItem-Objekt  
     Selected-Eigenschaft 782  
     Tag-Eigenschaft 780  
 ListViewItem-Objekt (ListView) 778  
 ListView-Steuerelement 777  
     BeginUpdate-Methode 779  
     Beschleunigen 779  
     Columns-Eigenschaft 778  
     ColumnsHeaderCollection-Auflistung 778  
     Daten für Spalten 778  
     Daten hinzufügen 778  
     Detaillierte Listendarstellung 777  
     Einträge selektieren 782  
     EndUpdate-Methode 779  
     FullRowSelect-Eigenschaft 777  
     Gitterlinien anzeigen 777  
     GridLines-Eigenschaft 777  
     HideSelection-Eigenschaft 777  
     Items-Auflistung 778  
     ItemSelectionChanged-Ereignis 781  
     Korrelation Objekt/Eintrag 780  
     ListViewItem-Objekt 778  
     Mehrere Einträge selektieren 781  
     MultiSelect-Eigenschaft 781  
     Objekinstanzen zuordnen 780

ListView-Steuerelement (*Fortsetzung*)  
     SelectedIndexChanged-Ereignis 781  
     Selektierte Elemente immer anzeigen 777  
     Selektion feststellen 781  
     Spaltenkopfbreite 778  
     Spaltenkopfbreiten anpassen, automatisch 779  
     Spaltenköpfe 778  
     SubItems-Auflistung 779  
     View-Eigenschaft 777  
     Zeile hinzufügen 778  
     Zeile selektieren, komplette 777  
 Load-Faktor 578  
 Lokalisieren *siehe* auch Ressourcen  
 Lokalisieren von Anwendungen 517  
 Long 145, 344  
 Long-Datentyp 448  
     Enum, konvertieren in 536  
     formatieren, für Textausgabe 497  
     Formatzeichenfolgen 501  
     Parse-Methode 454  
 Look and Feel von Windows XP 733  
 LSet-Funktion 466  
 LTrim-Funktion 469

**M**

Main, Sub 227  
 Managed Heap 275, 327  
 ManualResetEvent-Klasse (Threading) 952  
 Margin-Eigenschaft 44  
 MarshalAs-Attribute 691  
 Matches-Auflistung (Reguläre Ausdrücke) 631  
 Match-Klasse (Reguläre Ausdrücke) 630  
 Maximalwerte 457  
 MaxValue 457  
 MDI *siehe* Multi Document Interface  
 MDI-Anwendungen 805  
     Aufteilung von Funktionalitäten 806  
     Child Window 805  
     Dokumentklassen 805  
     Elternfenster 806  
     Formular als Container 806  
     Funktionalitäten aufteilen in Parent/Child 806  
     Kindfenster 805  
     Menüs verschmelzen 807  
     Parent Window 806  
 Me 236, 293  
 Meldungsfelder *siehe* Modale Dialoge  
 MemberInfo-Klasse 696  
 Member-Variablen 222  
     Speicherfolge in Strukturen 340  
 MemberwiseClone-Methode 318  
 Menüs *siehe* MenuStrip-Steuerelement  
 MenuStrip-Steuerelement  
     MDI-Anwendungen, konfigurieren 806  
     Schnellzugriffstasten 772  
     ShortcutKeyDisplayString-Eigenschaft 772  
     ShortcutKeyes-Eigenschaft 772  
     Tastenkürzel 772

Message Queue *siehe* Nachrichtenwarteschlange  
 Message-Klasse 830  
 MethodBase-Klasse 696  
 Methoden  
     AddressOf 432  
     Basisklassenmethoden überschreiben 271  
     ersetzen 272  
     MustOverride 296  
     Objektvariablen, speichern in 432  
     optionale Parameter 233  
     Signatur 232  
     Speichern in Objektvariablen 432  
     statische 221, 223  
     überladen 231  
     Überschatten 318  
     Überschreiben (Polymorphie) *siehe* Polymorphie  
     überschreiben in Basisklasse 271  
     Überschreibungzwang 296  
     veraltete Markieren 230  
     virtuelle 294  
     Zeiger auf 640  
 MethodImpl-Attribut 950  
 Mid-Funktion 466  
 Millimeter 869  
 Minimalwerte 457  
 MinValue 457  
 Modale Formulare  
     Abbrechen (Tastatur) 774  
     Aktivierungsreihenfolge 774  
     Anwendungsbeispiel 760  
     Bestätigen (Tastatur) 774  
     Closing-Ereignis 758  
     Darstellen 757  
     Daten erfassen/bearbeiten 761  
     Datenaustausch in 756  
     Dateneingabe 768  
     Datenausgabe 759  
     Dialogergebnis 756  
     DialogResult-Eigenschaft 756  
     Eingabefehler anzeigen 768  
     Instanzieren, ohne 715  
     Kommunikation zwischen 763  
     MessageBox 755  
     Muster für 756  
     Nachrichtenwarteschlange 755  
     Ressourcen freigeben 756  
     Schließen verhindern 758  
     ShowDialog 757  
     Umgang mit 755  
     Using 757  
     Validierung *siehe* Validieren, Formulareingaben  
 Modifizierer *siehe* Zugriffsmodifizierer  
     Narrowing 387  
     Widening 387  
     WithEvents 415  
 Module 324  
 Monitor-Klasse (Threading) 943  
 MRU-Liste der Projekte 15  
 mscoree.dll 198  
 Müllabfuhr 357  
 Multi Core-Systeme 933  
 Multi Document Interface 805

Multiprozessor-Systeme 933  
MultiSelect-Eigenschaft (ListView) 781  
Multitasking  
    Cooperative Multitasking 939  
    preemptive 939, 941  
MustInherit 295  
MustOverride 296  
Mutex  
    Einsatz bei Singleton-Klassen 326  
Mutex-Klasse 946  
    Array 949  
Mutual Exclusion *siehe* Mutex  
My.Application 713, 716  
My.Computer 713  
    Dateioperationen 724  
    FileSystem 724  
My.Forms 713  
My.Resources 714  
My.Settings 714  
    Anwendungseinstellungen 727  
My.User 714  
My.WebServices 714  
MyBase 236, 293  
MyClass 293  
My-Namespace  
    Anwendungsbeispiel 705  
    Anwendungseinstellungen 727  
    Dateioperationen 724  
    Formulare, ohne Instanzen 715  
    Hintergrund 713  
    Ressourcen 718  
    Übersetzen in andere Sprachen 721  
MyProject-Klasse 716

## N

Nachrichten-Channel 14  
    einstellen 15  
Nachrichtenfelder *siehe* Modale Dialoge  
Nachrichtenwarteschlange  
    Ereignisse auslösen 833  
    Modale Formulare 755  
    Schließen von Formularen 776  
Namespaces 178  
    Bestimmen für Projekte 181  
    Erklärung 197  
    Global 188  
    Importieren 179  
    Importieren, global für Projekt 180  
    verschiedene in einer Assembly 182  
    Zugriff auf System-Namespace (Global) 188  
Narrowing-Modifizierer 387  
NativeWindow 829  
Navigieren 102  
Navigieren, im Code mit Ausgabefenster 31  
Navigieren, im Code mit Fehlerliste, Aufgabenliste 30  
NetworkAvailabilityChanged -Ereignis  
    (Anwendungsframework) 736  
Neue Codedateien anlegen 84

New 236  
New, Sub  
    Konstruktoren aufrufen, weitere 236  
     MyBase 236  
    Überladen 231  
New, Sub in Klassen 227  
Nothing 328  
Null 328  
Nullable(Of 429  
Nullable(Of ) 595  
null-bare Wertetypen 429  
Nullbare-Datentypen 595  
NumberFormatInfo-Klasse 511  
Numerale 207

## 0

Object 146, 311  
    Eigenschaften 318  
    Equals 315, 318  
    Equals, Praktische Tipps 317  
    Finalize 318  
    GetHashCode 318  
    GetType 318, 693  
    MemberwiseClone 318  
    Methoden 318  
    ReferenceEquals 318  
    Sender bei Ereignissen, Träger der Quelle 420  
    ToString 318  
Objekte  
    Analysieren 694  
    Arrays, als 544  
    Binden an Steuerelemente 784  
    BindingSource-Klasse 785  
    CurrencyManager 784  
    Darstellung in ListBox 312  
    DeepCopy 669  
    Deklarieren mit Ereignissen 415  
    Delegaten *siehe* Delegaten  
    Einführung 193  
    entsorgen 355  
    Ermitteln des Typs 693  
    finalisieren 355  
    freigeben, automatisch 186  
    Gleichheit, prüfen auf 315  
    Instanzieren 227  
    Klassenmember 222  
    Klonen 666  
    Konstruktoren überladen 231  
    Managed Heap 275, 327  
    Member, als 222  
    Methoden von Object 311  
    Nothing 328  
    Polymorphie *siehe* Polymorphie  
    PropertyManager 784  
    Referenztypen 327  
    serialisieren 657  
    Speichern von Inhalten 657  
    Speicherung 275, 327

Objekte ( <i>Fortsetzung</i> )	Operatorenprozeduren ( <i>Fortsetzung</i> )
statische 221	Problembehandlung 389
Type 692	Rechenoperatoren, implementieren 383
Typermittlung 693	Signaturenmehrdeutigkeiten 390
Typsicherheit 201	Strukturen vorbereiten für 379
vergleichen 553	Typkonvertierungen, implementieren 386
Wertetypen 327	Überladen 384
XML, speichern als 674	Übersicht möglicher Operatoren 391
Zeiger 275, 327	Vergleichsoperatoren, implementieren 385
Objektorientierte Programmierung 206	Wahr/Falsch-Auswerter, implementieren 387
Obsolete-Attribut 230, 688	Option Strict 80, 234
On Error GoTo, Ersatz für 168	optionale Parameter 233
OnActivated-Methode 838	OrElse 174
OnClick-Methode 839, 848	OutOfRangeException 446, 447
OnClosed-Methode 838	Overridable-Anweisung 272
OnClosing-Methode 758, 838	Overrides-Anweisung 271
OnCreateControl-Methode 837, 847	
OnDeactivate-Methode 838	
OnDoubleClick-Methode 840, 849	
OnEnter-Methode 845, 854	<b>P</b>
OnGotFocus-Methode 845, 854	
OnHandleCreated-Methode 837, 847	Padding-Eigenschaft 44
OnHandleDestroyed-Methode 848	PageUnit-Eigenschaft 869
OnInvalidate-Methode 843, 853	PaintEventArgs-Klasse 865
OnKeyDown-Methode 841, 850	Parameter
OnKeyPress-Methode 842, 851	Arrays, als 541
OnKeyUp-Methode 842, 851	Befehlszeile, auslesen 716
OnLayout-Methode 844, 853	ByRef 336
OnLeave-Methode 845, 855	ByVal 336
OnLoad-Methode 837	Eigenschaften, für 250
OnLocationChanged-Methode 842, 851	Ereignisse, für 422, <i>siehe auch</i> Ereignisparameter
OnLostFocus-Methode 845, 855	ermitteln, in Formularen 756
OnMouseDown-Methode 839, 848	EventArgs 422
OnMouseEnter-Methode 840, 849	Konstruktoren, in 269
OnMouseHover-Methode 840, 849	Operatorenprozeduren überladen 384
OnMouseLeave-Methode 841, 850	PaintEventArgs-Klasse 865
OnMouseMove-Methode 841, 850	SELECT-Abfragen (SQL), in 1031
OnMouseUp-Methode 840, 849	überladen (Eigenschaften) 251
OnMouseWheel-Methode 841, 850	Parent Window 806
OnMove-Methode 842, 851	Parse 333, 346, 347
OnPaintBackground-Methode 844, 854	ParseExact 346, 347
OnPaint-Methode 844, 854	ParseExact-Methode (Date-Datentyp) 493
OnResize-Methode 843, 852	Parse-Methode (Date-Datentyp) 493
OnSizeChanged-Methode 843, 852	Parse-Methode (numerische Werte) 454
Onxxx-Methoden <i>siehe</i> Ereignisse, Onxxx	Partial 190
überschreiben	PathGradientBrush-Objekt 868
OOP 206	Pen-Objekt 867
wichtiger Tipp zu 712	Performace-Counter 481
Operatoren	PictureBox-Steuerelement, Ausschnitte einstellen in 58
AndAlso 174	Pinsel 867
benutzerdefinierte <i>siehe</i> Operatorenprozeduren	Pixel 869
Bitverschiebungen 166	PointF-Struktur 868
Boolesche 484	Point-Struktur 868
GetType 693	Polygone 886
Kurzschriftweise 165	Polymorphie 261, 279
OrElse 174	am Beispiel 282
Type Of 693	Beispiel als Analogie 279
Überladen von Operatorenprozeduren 384	Beispiel, Praxisbeispiel 280
Vergleiche, für 484	Beispiel, ToString 312
Operatorenprozeduren 377	Einführung 279
Einführung 378	ListBox 312
Klassen vorbereiten für 379	

**Polymorphie (Fortsetzung)**  
 Me 293  
 MyBase 293  
 MyClass 293  
 ToString 312  
 Wiederverwenden von Code 290  
**Preemptive Multitasking** 939  
**PreProcessMesage** 833  
**Preserve-Anweisung** 544  
**Primitive Datentypen** 440  
 Boolean 147  
 Byte 146  
 Char 147  
 Currency (VB6) 145  
 Date 147  
 Decimal 147  
 Decimal, Unterschied zu VB6 145  
 Deklaration 148  
 Double 147  
 Integer 146  
 Integer, Unterschied zu VB6 145  
 Konstanten 150  
 Long 147  
 Long, Unterschied zu VB6 145  
 nullbare 595  
 Object 147  
 Object, Unterschied zu VB6 146  
 Operatoren 165  
 SByte 146  
 Short 146  
 Short, Unterschied zu VB6 145  
 Single 147  
 String 147  
 Typliterale 150  
 Typsicherheit 150  
 UInteger 146  
 ULong 147  
 UShort 146  
 Variablendeklaration in For-Schleifen 159  
 Variablenverwendung in For/Each-Schleifen 161  
 Variant (VB6) 146  
 Werteneicht darstellbare Werte 156  
 Zahlenüberläufe 156  
**Private** 257, 258  
**ProcessCmdKey-Methode** 846, 855  
**ProcessDialogChar-Methode** 846, 856  
**ProcessDialogKey-Methode** 846, 856  
**ProcessKeyPreview-Methode** 847  
 Programmcode bearbeiten *siehe* Codeeditor  
**Programme**  
 internationalisieren 517  
 lokalisieren 517  
**Programmfehler**  
 kulturspezifisch 500  
**Programmieren**  
 Regular Expressions 630  
**Programmstart** 227  
**Projekt/Projektmappe öffnen** 16  
**Projekte** 21  
 alle Dateien anzeigen 22  
 als Startprojekt 21  
 Assemblyname, resultierender 181  
**Projekte (Fortsetzung)**  
 Dateianzeige aktualisieren 23  
 Dateitypen 22  
 Eigenschaften anzeigen 23  
 Formular hinzufügen 65  
 Konfigurationseinstellungen 100  
 Liste der letzten 15  
 MRU-Liste 15  
 MyProject 716  
 Namespace zuweisen 181  
 Neu anlegen 41  
 versteckte Dateien anzeigen 22  
 verwalten 21  
**Projektmappe** 21  
 ausführen 21  
 Build 21  
 Eigenschaften abrufen 21  
 Eigenschaften festlegen 21  
 erstellen (komplizieren) 21  
 neu erstellen 21  
 neu erstellen (komplizieren) 21  
 speichern 21  
 umbenennen 21  
 verwalten 21  
 zur Quellcodewaltung hinzufügen 21  
**Projektmappen**  
 Any-Zieltyp 200  
 Konfigurationseinstellungen 100  
 öffnen 16  
 Startobjekt festlegen 227  
**Projektmappen-Explorer** 21  
 Aktualisieren der Ansicht 23  
 alle Projektdateien anzeigen 22  
 Codedateien organisieren 23  
 Codeeditor für Projektdatei aufrufen 23  
 Dateioperationen 24  
 Designer für Projektdatei aufrufen 23  
 Funktionen der Werkzeugleiste 23  
 Klassendesigner aufrufen 23  
 Neue Codedateien anlegen 84  
 Startobjekt festlegen 227  
 Symbole 23  
**Property** 246  
**Property Extender** 56  
**PropertyInfo-Klasse** 696  
**PropertyManager-Klasse** 784  
**Protected** 257, 258  
**Protected Friend** 257, 258  
**Prozedurale Programmierung** 206  
 Kontroverse 207  
**Prozeduren**  
 AddressOf 432  
 Property 246  
 Speichern in Objektvariablen 432  
**Public** 257, 258  
**Pulldown-Menüs** *siehe* MenuStrip-Steuerelement  
**Pull-Methode** 589  
**Push-Methode** 589

## Q

Quantifizierer (Reguläre Ausdrücke) 620  
Queue-Auflistung 587  
  ausreihen (Dequeue-Methode) 588  
  einreihen (Enqueue-Methode) 588

## R

RaiseEvent-Anweisung 419  
Random-Klasse 350  
ReaderWriterLock-Klasse (Threading) 951  
Read-Methode 1000  
ReadOnly (Eigenschaften) 249  
Recherchieren 122  
RectangleF-Struktur 868  
Rectangle-Struktur 868  
ReDim-Anweisung 542  
Refactoring 86  
  Begriffsdefinition 745  
ReferenceEquals-Methode 318  
Referenztypen 327  
  Null 328  
  Performance-Unterschied zu Wertetypen 341  
Reflection 687  
  Assembly-Klasse 694  
  AssemblyQualifiedName-Eigenschaft 694  
  Attribute ermitteln 697  
  Attribute ermitteln, benutzerdefiniert 700  
  Attributes-Eigenschaft 694  
  BaseType-Eigenschaft 694  
  Eigenschafteninformationen 696  
  Einführung 691  
  FullName-Eigenschaft 694  
  GetCustomAttributes-Methode 694  
  GetEvent-Methode 694  
  GetEvents-Methode 694  
  GetField-Methode 694  
  GetFields-Methode 694  
  GetMember-Methode 694  
  GetMembers-Methode 694  
  GetProperties-Methode 694  
  GetProperty-Methode 694  
Refresh-Methode (Steuerelemente) 923  
Region 104  
  Designer-Code 822  
Regionen (geographisch) 499  
Registerkartengruppen 18  
Regular Expressions *siehe* Reguläre Ausdrücke  
Reguläre Ausdrücke 613  
  austesten 614  
  Beispiel 634  
  Captures-Auflistung 625, 632  
  Einführung 616  
  Groups-Auflistung 632  
  Gruppen 622  
  Gruppensteuerzeichen 629  
  Joker 613

## Reguläre Ausdrücke (*Fortsetzung*)

  Matches-Auflistung 631  
  Match-Klasse 630  
  Namensbereich 613  
  Programmieren mit 630  
  Quantifizierer 620  
  Sonderzeichen, suchen nach 617  
  Suche, komplex 618  
  Suchen und Ersetzen 624  
  Suchoptionen 627  
  Suchvorgänge, einfache 616  
  Wildcard 613  
Rekursion *siehe* Rekursion  
Release-Konfigurationseinstellungen 100  
RemoveHandler 423  
Remove-Methode (ListBox) 317  
Replace-Funktion 467  
Ressourcen  
  Abrufen 719  
  Anlegen und Verwalten 718  
  Internationalisieren 721  
Ressourcen aufteilen (Threading) 946  
Resultsets 994  
  Auslesen, zeilenweise *siehe* DataReader-Klasse  
ResumeLayout 825  
Reverse-Methode 550  
Right-Funktion 466  
römische Numerale 207  
Römische Numerale 220  
RSet-Funktion 466  
RTrim-Funktion 469  
Rundungsfehler 451

## S

SByte-Datentyp 446  
Schleifen  
  Continue 185  
Schleifenabbruchbedingungen 487  
Schlüssel  
  benutzerdefinierter 581  
  Equals-Methode 582  
  GetHashCode-Methode 582  
  Unveränderlichkeit 585  
Schlüssel (Auflistungen) 571  
Schnellzugriffstasten 63, 771  
Schnittstellen 297  
  Anwendungsbeispiel 298, 299  
  Benennungskonventionen 299  
  Beschränkungen, auf (Generische Datentypen) 405  
  Boxing, Besonderheiten 354  
  explizite Member-Definition 571  
  Implementieren 300  
  Implementieren in Klassen 300  
  Implementieren von Schnittstellen 309  
  private Member öffentlich machen 571  
Schnittstellen, IntelliSense-Unterstützung 301  
Scroll-Funktionalität für Formulare 57  
Seiteneinteilung (beim Zeichnen) 869

Seitenskalierung (beim Zeichnen) 869  
 Select-Anweisung 487  
 SELECT-Anweisung (SQL)  
     Abfragen generieren 1027  
     Abfragetipps 1009  
     Beispiele, typische 1006  
     Datumswerte abfragen 1006  
     Felder gezielt auswählen 1007  
     Tabellen verknüpfen 1008  
 Selected-Eigenschaft (ListViewItem-Objekt) 782  
 SelectedIndexChanged-Ereignis (ListView) 781  
 SelectedItem-Eigenschaft (ListBox) 317  
 Serialisierung 657  
     BinaryFormatter 660  
     Probleme bei KeyedCollection(Of ) 680  
     SOAPFormatter 660  
     Versionen 674  
     Versionsunabhängigkeit 679  
     Workaround bei KeyedCollection-Auflistung 684  
     XML 674  
     Zirkelverweise 671  
 Serializable-Attribute 690  
 Server-Explorer 995  
 Set 159  
 Set-Accessor (Eigenschaften) 247  
 SET-Anweisung 1014  
 Set-Anweisung (Objektzuweisung) 251  
 SetBoundsCore 852  
 SetClientSize 853  
 SetStyle-Methode 880  
 Settings 93  
     im Code verwenden 95  
     Speicherort 98  
     Variablen einrichten 94  
     Verknüpfen von Formularen, mit 97  
     Verknüpfen von Steuerelementen, mit 97  
 SetVisibleCore 838  
 Shadows 318  
 Shared 223  
 Short 145  
 ShortcutKeyDisplayString-Eigenschaft 772  
 ShortcutKeyes-Eigenschaft 772  
 Short-Datentyp 446  
     Enum, konvertieren in 536  
     Parse-Methode 454  
 ShowDialog-Methode 757  
 ShutDown-Ereignis (Anwendungsframework) 735  
 Signaturen 232  
 Single-Datentyp 449  
     formatieren, für Textausgabe 497  
     Formatzeichenfolgen 501  
     Parse-Methode 454  
     Vergleichen 454  
 Singleton-Klassen 324  
 SizeF-Struktur 868  
 Size-Struktur 868  
 Sleep 954  
 SmartClient-Entwicklung 739  
 SmartClients  
     Anwendungsframework 731  
     Anwendungsschichten 743  
     Beenden der Anwendung, Benachrichtigung 735  
     SmartClients (*Fortsetzung*)  
         Daten strukturieren in 743  
         Datenbankgebundene 1033  
         Dokumenttypen 742  
         Einführung 742  
         Ereignisse, globale *siehe* Anwendungsereignisse  
         Fehlerbehandlung, globale 736  
         Haupt-Thread 953  
         Herunterfahren, Modus 734  
         Internationalisieren 721  
         lokalisieren 718  
         MDI-Anwendungen *siehe* MDI-Anwendungen  
         Mehrgefachstart verhindern 733  
         Netzwerkzustandsänderung, Benachrichtigung 736  
         Schichten für Anwendungen *siehe*  
             Anwendungsschichten  
         Splash-Dialoge 734  
         Sprachen, andere 718  
         SQL Server 2005 Express 985  
         Starten der Anwendung, Benachrichtigung 735  
         Tastaturbedienung 771  
         Thread, erster 953  
         übersetzen in andere Sprachen 718  
         UI-Thread 953  
         Unterstützung durch VB2005 731  
         Weiteres Starten der Anwendung,  
             Benachrichtigung 736  
 Smarttags  
     Codeeditor, für 79  
     Codeeditor, Verwendung im 225  
     Steuerelemente, für 47  
 SoapFormatter (Serialisierung) 661  
 Softwarevoraussetzungen 3  
 SolidBrush-Objekt 868  
 Solution-Explorer 21  
 SortedList-Auflistung 590  
     Indexer, benutzerdefiniert 591  
     Sortierkriterien, benutzerdefiniert 591  
 Sortieren (Arrays) 549  
 Sort-Methode (Arrays) 549  
 Space Invaders 217  
 Speicherverwaltung 275, 327  
 Splash-Dialoge 734  
 SplitContainer 48  
 Split-Funktion 470  
 SQL 983  
     DELETE-Anweisung 1009  
     INSERT-Anweisung 1009  
     SELECT-Anweisung *siehe* SELECT-Anweisung  
         (SQL)  
     SET-Anweisung 1014  
     UPDATE-Anweisung 1009  
 SQL Server 2005 Express 984  
     Ausliefern, mit eigenen Anwendungen 984  
     Authentifizierungsmodus 990  
     Deployment 984  
     Dienstkonten 989  
     Einschränkungen 984  
     Einsetzen in eigenen Anwendungen 985  
     Firewall-Konfiguration 991  
     Gemischter Modus, Installation 986  
     Installation 986

SQL Server 2005 Express ( <i>Fortsetzung</i> )	Steuerelemente ( <i>Fortsetzung</i> )
Instanzen 989	ListBox <i>siehe</i> ListBox-Steuerelement
Netzwerkkonfiguration 991	ListView <i>siehe</i> ListView-Steuerelement
SELECT-Anweisung	Menüs <i>siehe</i> ToolStrip-Steuerelement
<i>siehe</i> SELECT-Anweisung (SQL)	Name-Eigenschaft 62
Server-Explorer, Verbindung herstellen mit 995	Namenskonventionen im Buch 67
Verbindung herstellen mit Server-Explorer 995	Neuziehen auslösen 923
SqlCommandBuilder-Klasse 1019	OnPaint-Methode 864
SqlCommand-Klasse 999	PictureBox 58
SqlConnection-Klasse 994, 998	Positionieren, genaues 45
SqlDataAdapter-Klasse 1002	Property Extender 56
SqlDataReader-Klasse <i>siehe</i> DataReader-Klasse	proportionales Vergrößern 50
Stack-Auflistung 589	Pulldown-Menüs <i>siehe</i> ToolStrip-Steuerelement
Abladen (Pull-Methode) 589	Referenzsteuerelement beim Anordnen 46
Aufladen (Push-Methode) 589	Schnellzugriffstasten 63
Standardeigenschaften 252	Scrollen in Formularen/Containern 57
Standardkonstruktor 237, 268	Seiten docken 48
Standardschaltflächen 64	Selektieren von unsichtbaren 809
Start eines Programms 227	selektieren, erweitertes 60
Start Page 13	SetStyle-Methode 880
Startobjekt 21	Smarttags 47
Startobjekt festlegen 227	SplitContainer 48
Startprojekt 21	Tab Order <i>siehe</i> Aktivierungsreihenfolge
Startseite 13	TableLayoutPanel 48, 50
Neues Projekt anlegen 14	Tabulatorreihenfolge 60
Projekt öffnen 14	Tastenkürzel für Layout 70
überflüssige Einträge entfernen 15	Text statt Caption 62
Startup-Ereignis (Anwendungsframework) 735	Text-Eigenschaft 62
StartUpNextInstance-Ereignis (Anwendungsframework)	Threading in 953
736	thread-sicher 953
Static 223	verankern 48, 49
statische Elemente 221	Verankern in TableLayoutPanel 55
Steuerelemente	Zeichnen von Inhalten 864
Aktivierungsreihenfolge 60	Z-Reihenfolge 908
als Container 44	zur Laufzeit nicht sichtbare 769
Anchor-Eigenschaft 48, 49	Steuerelemente positionieren 43
Anordnen in Tabellen 48	Stifte 867
Anordnen in TableLayoutPanel 53	Strg (Tastatur) 772
Anordnen in veränderbaren Bereichen 48	strikte Typbindung 234
Anordnen mit Umbruchlogik 49	String\$-Funktion 462
Arrays 814	String.Intern-Methode 465
Aufgaben, häufige 47	Stringbuilder-Klasse 476
benutzerdefiniert <i>siehe</i> Benutzersteuerelemente	String-Datentyp 460
Binden an Daten 784	Ähnlichkeit, prüfen auf 484
Caption-Eigenschaft 62	Boolean, aus 483
ControlCollection 826	Carriage Return 463
Darstellen von Daten in 777	Date, umwandeln in 493
DataGridView <i>siehe</i> DataGridView-Steuerelement	deklarieren und definieren 461
Dock-Eigenschaft 48	Enum, konvertieren aus 536
dynamisch Anordnen 48	Funktionen, neue in .NET 461
Eigenschaften in Settings speichern 97	Geschwindigkeit beim Zusammenbauen 476
Ereignisse <i>siehe</i> Formular- und	IndexOfAny-Methode 467
Steuerelementereignisse	Intern-Methode 465
erweitern vorhandener Eigenschaften 56	Iterieren durch Zeichen 475
flimmerfreie Darstellung 878	Konstruktor 462
FlowLayoutPanel 49	Länge ermitteln 465
Fokussierungsreihenfolge <i>siehe</i>	Längen angleichen 466
Aktivierungsreihenfolge	Length-Methode 465
Größe anpassen 45	Like-Operator 484
Größe anpassen, programmtechnisch 882	numerische Werte wandeln, in 454
Grundverhalten 880	PadLeft-Methode 466
Layout-Funktionen 68	PadRight-Methode 466

**String-Datentyp (Fortsetzung)**  
 Remove-Methode 467  
 Replace-Methode 467  
 Sonderzeichen 463  
 Speicherbedarf 464  
 Speicheroptimierung 464  
 Split-Methode 470  
 String.IndexOf-Methode 467  
 StringBuilder, Vergleich 476  
 Stringpool 464  
 SubString-Methode 466  
 Suchen und Ersetzen 467  
 Teile ermitteln 466  
 TrimEnd-Methode 469  
 Trimmern 469  
 Trim-Methode 469  
 TrimStart-Methode 469  
 umfangreiches Beispiel 472  
 Unveränderlichkeit 464  
 vbBack 463  
 vbCr 463  
 vbCrLf 463  
 vbLf 463  
 vbnewline 463  
 vbNullChar 463  
 vbNullString 463  
 vbTab 463  
 Verketten, Performance-Tipps 462  
 Wagenrücklauf 463  
 Zeilenvorschub 463  
 zerlegen in Teile 470  
**Strings**  
 Konvertieren 345  
 Parsen 346  
**StructLayout-Attribut** 340  
**Structure** 329  
**Structured Query Language** *siehe* SQL  
**Strukturen** 329  
 CLS-Compliance 444  
 Ereignisse *siehe* Ereignisse  
 Ereignisse verknüpfen 415  
 Ereignisse, Instanziieren zum Konsumieren von 415  
 explizite Konvertierung, implementieren 386  
 generisch, benutzerdefiniert *siehe* Generische  
     Datentypen  
 implizite Konvertierung, implementieren 386  
 Kommunikation zwischen *siehe* Ereignisse  
 Konstruktoren 336  
 Operatoren, benutzerdefinierte *siehe*  
     Operatorenprozeduren  
**Sub Main** 227  
**Sub New** 227  
 Konstruktoren aufrufen, weitere 236  
 MyBase 236  
 Überladen 231  
**SubItems-Auflistung (ListView)** 779  
**Suchen und Ersetzen** 104  
 Reguläre Ausdrücke 624  
**Suchergebnisse**  
 durch Lesezeichen markieren 107  
**Suspend** 954  
**SuspendLayout** 825

**Symbole**  
 Projektmappen-Explorer 23  
 Synchronization-Attribute (Threading) 950  
 SyncLock-Methode 939  
 Syntaxfehler 236  
 System.Boolean (Datentyp) 482  
 System.Byte (Datentyp) 445  
 System.Char (Datentyp) 459  
 System.DateTime (Datentyp) 488  
 System.Decimal (Datentyp) 451  
 System.Double (Datentyp) 450  
 System.Globalization 456  
 System.Int16 (Datentyp) 446  
 System.Int32 (Datentyp) 447  
 System.Int64 (Datentyp) 448  
 System.SByte (Datentyp) 446  
 System.Single (Datentyp) 449  
 System.UInt16 (Datentyp) 447  
 System.UInt32 (Datentyp) 448  
 System.UInt64 (Datentyp) 449

**T**

**Tab Order** *siehe* Aktivierungsreihenfolge  
**Tabellen (Daten)** 997  
**TableAdapter-Klasse** 1027  
 Abfragen, zusätzliche als Funktion 1027  
 DataAdapter-Klasse, Basierung auf 1027  
 DataTable-Klasse, Zusammenspiel mit 1027  
 Datentabellen füllen 1027  
 Füllen von Tabellen 1027  
 Funktionsweise 1027  
**TableLayoutPanel** 48, 50  
 Prozentuale und fixe Zellengrößen 53  
 Spalten einstellen 51  
 Steuerelemente anordnen, in 53  
 Steuerelemente verankern 55  
 Zeilen einstellen 51  
 Zellen verbinden 56  
 Tabulatorreihenfolge 60, *siehe* Aktivierungsreihenfolge  
 Tag-Eigenschaft (ListViewItem-Objekt) 780  
 Tag-Eigenschaft (Steuerelemente) 815  
**Task-Manager** 935  
**Tastaturbedienung** 771  
**Tasturbefehle** 34  
**Tastenkombinationen** 34  
**Testbild (Demo für GDI+)** 869  
**Testen von Software** 117  
**Texte (Grafik)**  
 ausmessen 877  
 einpassen 876  
 zeichnen 876  
**TextureBrush-Objekt** 868  
**Then-Anweisung** 485  
**Threading** 933  
**Thread-Pool** 970  
**Threads**  
 Abbrechen 955  
 aus Pool 970

Threads ( <i>Fortsetzung</i> )	Toolfenster ( <i>Fortsetzung</i> )
Auslastung 935	zusätzliche öffnen 20
Aussetzen 954	ToString 346, 456
AutoResetEvent-Klasse 952	ListBox-Eintrag, Überschreiben für 312
BackgroundWorker-Komponente 978	Object, von 318
Beenden 955	Polymorphie 312
Dateizugriffe synchronisieren 951	ToString-Methode 287
Daten austausch 959	Trace-Listener 363
Delegaten, durch 981	Trim-Funktion 469
einfachste Vorgehensweise 978	True 482
Formulare 975	Try/Catch/Finally 169
Fortschriftsanzeigen realisieren 978	TryParse 347
Grundlagen 936	TryParse-Anweisung 458
Grundsätzliches 939	Type
HyperThreading 933	Assembly-Eigenschaft 694
Interlocked-Klasse 950	AssemblyQualifiedName-Eigenschaft 694
Managen 954	Attributes-Eigenschaft 694
ManualResetEvent-Klasse 952	BaseType-Eigenschaft 694
MethodImpl-Attribut 950	MethodInfo-Klasse 696
Monitor-Klasse 943	FieldInfo-Klasse 696
Multiprozessor-Systeme 933	FullName-Eigenschaft 694
Mutex-Klasse 946	GetCustomAttributes-Methode 694
Notwendigkeit für 935	GetEvent-Methode 694
Praxiseinsatz 962	GetEvents-Methode 694
Prozessorauslastung 935	GetField-Methode 694
ReaderWriterLock-Klasse 951	GetFields-Methode 694
Ressourcen aufteilen zwischen 946	GetMember-Methode 694
Sleep 954	GetMembers-Methode 694
Starten 938, 954	GetProperties-Methode 694
Steuerelemente, in 953	GetProperty-Methode 694
Synchronisieren 939	MethodInfo-Klasse 696
Synchronisieren, mehrere gegenseitig 952	MethodBase-Klasse 696
Synchronisierungstechniken 949	PropertyInfo-Klasse 696
Synchronization-Attribut 950	Type Of-Operator 693
SyncLock 939	Type-Klasse 692
UI-Thread 953	Typen 692
vorhandene verwenden 970	casten 343
wechseln zwischen 978	Ereignisse, 415
tiefes Klonen 666	generisch, benutzerdefiniert <i>siehe</i> Generische
Tiers <i>siehe</i> Anwendungsschichten	Datentypen
Timer-Klasse 918	null-bar 429
TimeSpan-Struktur 489	Parameter für 398
Toolfenster	umwandeln 343
als Dokumentfenster 19	Typkonvertierung
andocken 19	DirectCast 348
Anordnung 20	Typliterale 150
arbeiten mit 19	Typparameter 398
Aufgabenliste <i>siehe</i> Aufgabenliste	Typsicherheit 150, 201, 234
Ausgabefenster <i>siehe</i> Ausgabefenster	Typsicherheit erzwingen 80
automatisch im Hintergrund 19	Typumwandlungen 343
Dokumentfenster, als 19	
Dynamische Hilfe <i>siehe</i> Dynamische Hilfe	
Eigenschaften <i>siehe</i> Eigenschaftenfenster	
Einführung 18	
Fehlerliste <i>siehe</i> Fehlerliste	<b>U</b>
Fenstereinstellungen zurücksetzen 112	Überladen 231
Klassenansicht <i>siehe</i> Klassenansicht	Überschatten von Prozeduren 318
Layout in der Entwicklungsumgebung 20	Überschreiben 271
Originalzustand 112	Überschreiben von Onxxx-Methoden <i>siehe</i> Ereignisse, Onxxx überschreiben
Projektmappen-Explorer 21	UInt16 <i>siehe</i> UShort-Datentyp
wichtige 20	
zur Laufzeit/Entwurfszeit 20	

UInt32 *siehe* UInteger-Datentyp  
 UInt64 *siehe* ULong-Datentyp  
 UInteger-Datentyp 448  
 UI-Thread 953  
 ULong-Datentyp 449  
 Umgestalten von Code (Refactoring) 86  
 Umwandeln, von Datentypen 343  
 Umwandlungsversuch 458  
 Unboxing 352  
 Unendlich 457  
 UnhandledException-Ereignis (Anwendungsframework) 736  
 Unicode-Zeichen 460  
 unverbundene Daten 1001  
 UPDATE-Anweisung (SQL) 1009  
 Update-Methode (Steuerelemente) 923  
 User Controls *siehe* Benutzersteuerelemente  
 UShort-Datentyp 447  
 Using-Anweisung 186, 757

**V**

Validieren  
 Benutzereingaben (DataGridView) 800  
 Eingaben in Zellen 801  
 Formulareingaben 766

Variablen  
 Arrays, Unterschiede zu VB6 164  
 Deklaration 148  
 Deklaration in For-Schleifen 159  
 Deklaration und Definition 155  
 Delegaten 433  
 Eigenschaften, Vergleich zu öffentlichen 255  
 Eliminierung von Set 159  
 globale, Initialisierungszeitpunkt 270  
 Gültigkeitsbereiche *siehe* Gültigkeitsbereiche  
 Initialisierungszeitpunkt bei globalen 270  
 Managed Heap 275, 327  
 Member-Variablen 222  
 Member-Variablen (Gültigkeitsbereich) 162  
 Namensgebung 204  
 öffentliche, Vergleich zu Eigenschaften 255  
 Operatoren 165  
 Option Strict 234  
 Referenztypen 327, 329  
 Speicherfolge in Strukturen 340  
 statische 221  
 strikte Typbindung 234  
 Structure 329  
 Typliterale 150  
 typsicher 234  
 Typsicherheit 150, 201  
 Typzeichen 148  
 Übergabe an Subs 175  
 Verweise 275, 327  
 Verwendung in For/Each-Schleifen 161  
 Wertetypen 327  
 Werteverlust im Gültigkeitsbereich 223  
 Zeiger 275, 327

Variablen (*Fortsetzung*)  
 Zuweisung durch Konstante 150  
 VBFixedArray-Attribut 690  
 VBFfixedString-Attribut 690  
 Vererben  
 Generische Datentypen 410  
 Vererbung 261  
 Vergleichen  
 Double- oder Single-Datentyp 454  
 Vergleichen von Objekten 553  
 Vergleicher (List (Of)) 609  
 Vergleichsoperatoren 484  
 Verhindern falscher Eingaben (DataGridView) 801  
 Versionsunabhängigkeit (Serialisierung) 679  
 Versteckte Projektdateien anzeigen 22  
 Versuchte Typkonvertierung 458  
 Vervollständigungsliste *siehe* IntelliSense,  
 Vervollständigungsliste  
 Verwaltung von Projekten 21  
 Verweisvariablen 275, 327  
 Vielgestaltungkeit *siehe* Polymorphie  
 View-Eigenschaft (ListView) 777  
 Virtual PC und Virtual Server 117  
 Virtuelle Prozeduren 294  
 VisibleClipBounds-Eigenschaft 869  
 Visual Basic  
 Express Edition 3  
 Visual Basic 2005  
 Anwendungsframework 731  
 Betriebssystemvoraussetzungen 5  
 Entwicklungsumgebung *siehe*  
 Entwicklungsumgebung  
 Express Edition auf Buch CD 4  
 IDE (Integrated Development Environment) *siehe*  
 Entwicklungsumgebung  
 Installation 4  
 Neuerungen, Übersicht 190  
 Umsteigen von VB6 143  
 Upgrade-Assistent 143  
 Visual Basic 6.0  
 Programme upgraden 143  
 Visual Basic 6.0-Unterschiede  
 Arrays 164  
 Currency 145  
 Datentypen, weggefallene 145  
 Datentypengrößen 145  
 Decimal 145  
 Eigenschaften, Default 252  
 Eigenschaften, Standard 252  
 Fehlerbehandlung 167  
 Gültigkeitsbereiche von Variablen *siehe*  
 Gültigkeitsbereiche  
 Instanzierung von Objekten 155  
 Integer 145  
 Kurzschlussauswertungen (AndAlso, OrElse) 174  
 Let 158  
 Long 145  
 New 155  
 Object 146  
 Objekte, Umgang mit 158  
 On Error GoTo, Ersatz für 168  
 Operatoren 165

Visual Basic 6.0-Unterschiede (*Fortsetzung*)  
Parameterübergabe an Sub/Function 175  
Set 158  
Short 145  
Variablen in For/Each-Schleifen, verwendbare 161  
Variablen in For-Schleifen, Deklaration 159  
Variablen, Gültigkeitsbereiche von *siehe*  
    Gültigkeitsbereiche  
Variablenbehandlung 144  
Variablendeklaration und -definition 155  
Variant 146  
Zahlenüberläufe 156  
Visual Studio  
    Versionen 3  
Visual Studio 2005  
    Arbeitsspeicherbedarf 116  
    Beta Release-Hinweise 5  
    Betriebssystemvoraussetzungen 5  
    Einstellungen sichern/wiederherstellen 112  
    Entwicklungsumgebung *siehe*  
        Entwicklungsumgebung  
    IDE (Integrated Development Environment) *siehe*  
        Entwicklungsumgebung  
    Installation 4  
    Mindestanforderungen 4  
    Originalzustand wiederherstellen 115  
    Versionsübersicht 6  
    zusätzliche Werkzeuge 238  
Visualisieren von Daten 777  
Visuelle XP-Stile 733  
Vorschriften zur Klassenimplementierung *siehe*  
    Schnittstellen

## W

Warnungen konfigurieren 28  
Web-Links zu Aktualisierungen 9  
WebMethod-Attribut 690  
Werte  
    Zeichen umwandeln in 460  
Wertetypen 327  
    Beschränkung, auf (Generische Datentypen) 409  
    Konstruktoren, bei 336  
    null-bar 429  
    Performance-Unterschied zu Referenztypen 341  
Widening-Modifizierer 387  
Wildcard 613  
Window *siehe* Formulare  
WindowClass 829  
Windows Forms-Anwendungen *siehe* SmartClients  
Windows XP  
    Firewall-Konfiguration (SQL Server 2005 Express)  
        991  
    Windows XP-Stile verwenden 733  
    Windows-Programmierung 185, 497

WithEvents-Anweisung 415  
WMClose-Methode 776  
WndProc-Methode 776, 830, 847  
Wrapper-Klassen 480  
Write 363  
WriteOnly (Eigenschaften) 249

## X

XML 674  
XML-Kommentare, benutzerdefiniert 80

## Z

Zahlen  
    formatieren 497  
    Zeichenketten wandeln, in 454  
Zahlenkonvertierung 207  
Zahlenkonvertierungsfehler 452  
Zahlensysteme 452  
Zahlenwerte  
    formatieren 287  
    Text, umwandeln in 287  
Zeichen  
    Carriage Return 463  
    Datentyp für 459  
    Sonderzeichen 463  
    Unicode 460  
    verkettete 460  
    Wagenrücklauf 463  
    Werte umwandeln in 460  
    Zeilenvorschub 463  
Zeichenketten *siehe* String-Datentyp  
Zeichenparameter *siehe* PaintEventArgs-Klasse  
Zeigervariablen 275, 327  
Zeitdifferenzen 489  
Zeitnähe der Datenbindung 784  
Zirkelverweise 671  
Z-Reihenfolge 908  
Zufallszahlen 350  
Zugriffsebenen *siehe* Zugriffsmodifizierer  
Zugriffsmodifizierer 256  
    Ändern durch Schnittstellen 571  
    Assembly (CTS) 257, 258  
    Eigenschaften, für 249  
    Family (CTS) 257, 258  
    FamilyOrAssembly (CTS) 257, 258  
    Friend 257, 258  
    Private 257, 258  
    Protected 257, 258  
    Protected Friend 257, 258  
    Public 257, 258

