

Leseprobe

Walter Doberenz, Thomas Gewinnus

Visual Basic 2012 - Kochbuch

ISBN (Buch): 978-3-446-43430-1

ISBN (E-Book): 978-3-446-43563-6

Weitere Informationen oder Bestellungen unter

<http://www.hanser-fachbuch.de/978-3-446-43430-1>

sowie im Buchhandel.

Kleiner VB-Anfängerkurs

Vor Beginn des Windows-Zeitalters wurden Programme geschrieben, die für jede Ein- und Ausgabe von Daten eine neue Textzeile in einem Bildschirmfenster (Konsole) erzeugten. Dieses Verhalten war ein Erbe aus jener Zeit, als lediglich Fernschreiber für die Ein- und Ausgabefunktionen von Computern zur Verfügung standen.

Aber auch heute noch kann es für den Newcomer durchaus sinnvoll sein, wenn er für seine ersten Schritte die gute alte Konsolentechnik verwendet. So kann er sich doch auf das zunächst Wesentliche, nämlich die Logik von VB-Befehlen, konzentrieren, ohne von der erdrückenden Vielfalt der Windows-Oberfläche abgelenkt zu werden.

Die folgende Serie von sechs absoluten Anfängerbeispielen benutzt zunächst Konsolenanwendungen, um einige grundlegende VB-Sprachelemente zu demonstrieren. Die letzten beiden Beispiele zeigen dann den Übergang zur zeitgemäßen Windows-Programmierung.

HINWEIS: Der "Kleine VB-Crashkurs" kann keinesfalls das Studium einführender Visual Basic-Literatur ersetzen (siehe z.B. unser Buch [Visual Basic 2012 – Grundlagen und Profiwissen]), sondern ist lediglich als Ergänzung zu verstehen.

R1 Das EVA-Prinzip anwenden

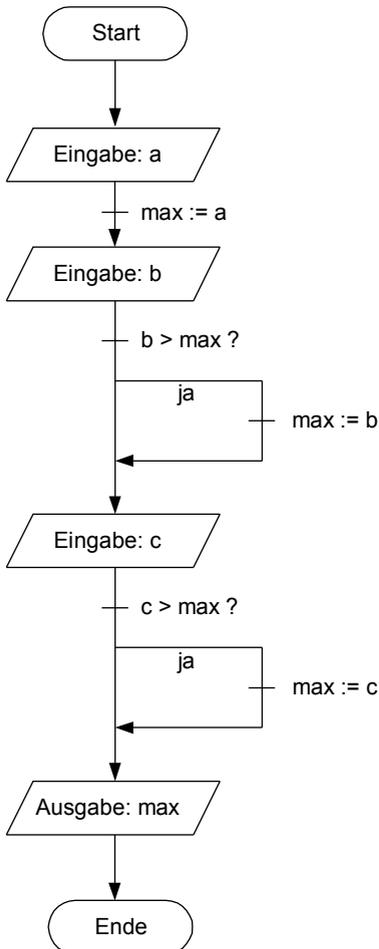
Jeder Weg, und ist er noch so weit, beginnt mit dem ersten Schritt! Für den Anfang soll uns deshalb ein ganz einfaches Beispiel genügen.

Aufgabenstellung

Nacheinander sind drei positive ganze Zahlen einzugeben. Das Programm soll die größte der drei Zahlen ermitteln und das Ergebnis anzeigen.

Lösungsvorschlag

Einen Vorschlag (Algorithmus) für den Programmablauf zeigt der nachfolgende Plan.



Sie erkennen daran das altbekannte EVA-Prinzip eines Programms, wobei die Anweisungen in der Reihenfolge

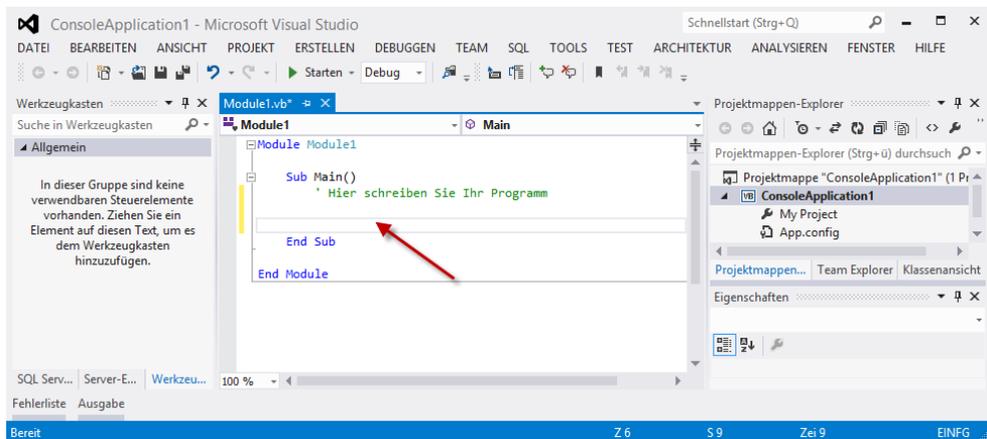
- Eingabe,
- Verarbeitung,
- Ausgabe

ausgeführt werden.

Ein Programmablaufplan (PAP), wie ihn die obige Abbildung zeigt, ist zwar heute etwas aus der Mode gekommen, für den Anfänger kann er aber ganz nützlich sein, um die Logik eines VB-Programms besser zu verstehen.

Programmierung

Wählen Sie auf der Startseite von Visual Studio den Link *Neu/Projekt...* Es öffnet sich das Dialogfenster "Neues Projekt". Wählen Sie den Projekttyp *Visual Basic* und die Vorlage *Konsolenanwendung*. *Name* und *Projektmappenname* können auf den Standardeinstellungen *ConsoleApplication1* verbleiben. Nachdem Sie den *Speicherort* (das Verzeichnis, in welchem Ihr Projekt abgespeichert werden soll) festgelegt haben, erscheint das Fenster des Quellcode-Editors, in welchem bereits ein Codegerüst "vorgefertigt" ist. Ihren eigenen Code fügen Sie innerhalb der *Main*-Methode hinzu:



Ergänzen Sie, ohne länger darüber nachzudenken, den Code entsprechend folgendem VB-Listing. Dabei können Sie die mit Apostroph (') eingeleiteten Kommentare bedenkenlos weglassen, da sie eh vom Compiler überlesen werden:

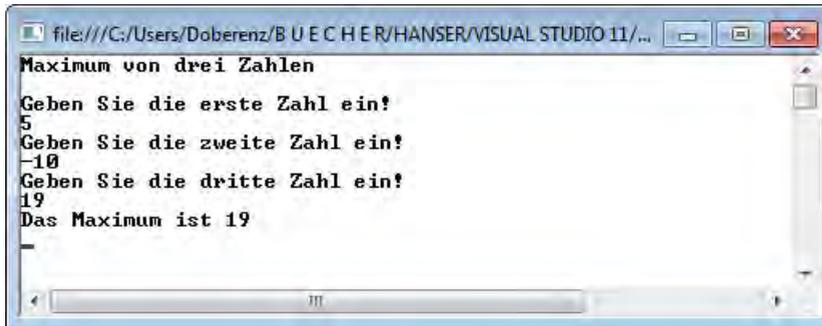
```

Module Module1
    Sub Main()
        Console.WriteLine("Maximum von drei Zahlen") ' Überschrift
        Console.WriteLine() ' Leerzeile
        Dim a, b, c, max As Integer ' Variablendeklaration
        Console.WriteLine("Geben Sie die erste Zahl ein!")
        a = CInt(Console.ReadLine()) ' Eingabe a
        max = a ' Initialisieren von max
        Console.WriteLine("Geben Sie die zweite Zahl ein!")
        b = CInt(Console.ReadLine()) ' Eingabe b
        If b > max Then max = b ' Bedingung
        Console.WriteLine("Geben Sie die dritte Zahl ein!")
        c = CInt(Console.ReadLine()) ' Eingabe c
        If c > max Then max = c ' Bedingung
        Console.WriteLine("Das Maximum ist " + max.ToString()) ' Ergebnisausgabe
        Console.ReadLine() ' Programm wartet auf <Enter>, um zu beenden
    End Sub

```

Test

Kompilieren Sie das Programm (F5-Taste) und überzeugen Sie sich von seiner Funktionsfähigkeit.



```
file:///C:/Users/Doberenz/BUECHER/HANSER/VISUAL STUDIO 11/...
Maximum von drei Zahlen
Geben Sie die erste Zahl ein!
5
Geben Sie die zweite Zahl ein!
-10
Geben Sie die dritte Zahl ein!
19
Das Maximum ist 19
```

HINWEIS: Durch Drücken der *Enter*-Taste beenden Sie Ihre erste VB-Anwendung.

Bemerkungen

Im Vorgriff auf die weiteren Kapitel erfolgt für den Anfänger hier die Zusammenstellung einiger wichtiger Visual Basic-Grundlagen, wie sie in diesem Beispiel zur Anwendung gekommen sind:

- Unser VB-Programm beginnt mit der *Main*-Methode, die gewissermaßen den Einstiegspunkt in das Programm darstellt.
- Jedes VB-Programm besteht aus einer Folge von Anweisungen. Es besteht kein Unterschied zwischen Groß- und Kleinschreibung!
- Jede VB-Anweisung wird mit einem Zeilenumbruch abgeschlossen, eine Zeile ist also eine kompletter VB-Befehl.
- Die mit einem Apostroph (!) eingeleiteten Anweisungen sind lediglich Kommentare und für den Programmablauf bedeutungslos.
- Vor Beginn eines Programms (bzw. eines in sich abgeschlossenen Programmteils) sind alle benötigten Variablen zu deklarieren, d.h., ihr Name und ihr Datentyp müssen festgelegt werden.
- Unter "Initialisierung einer Variablen" versteht man das Zuweisen eines Anfangswertes.
- *ReadLine* und *WriteLine* sind die einfachsten Ein-/Ausgabeanweisungen, wie Sie sie allerdings nur bei einer Konsolenanwendung verwenden sollten. Die Endung *...Line* der Befehlsörter bewirkt einen Zeilenvorschub.
- Die *If... Then...-Anweisung* (wenn ... dann ...) ist ein Verzweigungsbefehl (gewissermaßen eine "Programm-Weiche") und führt eine Anweisung in Abhängigkeit von einer Bedingung aus.

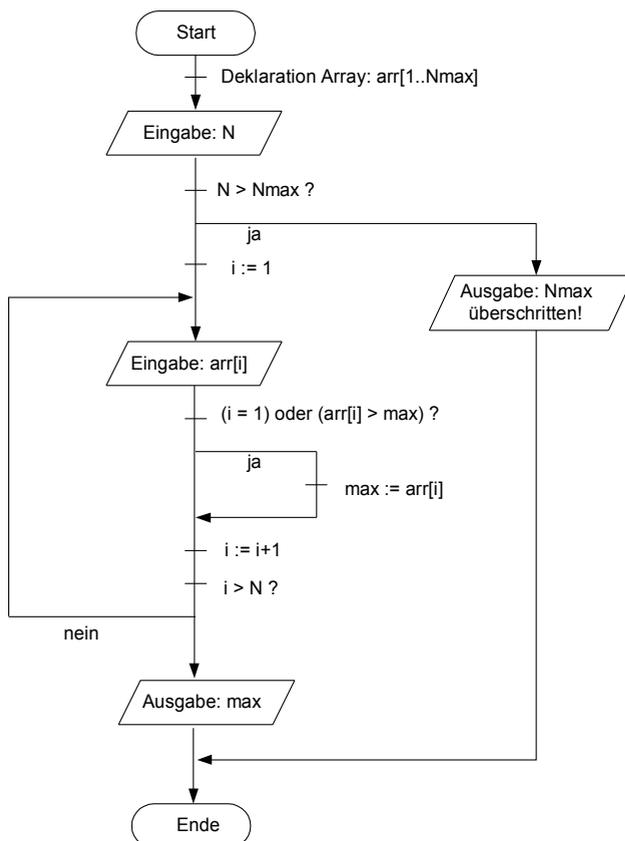
R2 Ein Array definieren und initialisieren

Aufgabenstellung

Erweitern Sie das Programm so, dass es die Eingabe einer flexiblen Anzahl von Werten ermöglicht!

Lösungsvorschlag

Der im nachfolgenden PAP dargestellte Algorithmus geht davon aus, dass die Zahlenwerte nicht mehr in Einzelvariablen a , b , c , sondern in einem Array gespeichert werden. Stellen Sie sich ein solches Array wie ein Regal mit einzelnen Fächern vor, in denen beliebig große Zahlenwerte abgelegt werden. Die Fächer sind beschriftet mit 1, 2, 3 ... Die Anzahl der Regalfächer beträgt N_{max} . Diese Konstante ist ausreichend groß zu wählen, damit genügend Reserven für die maximale Anzahl N von Zahlenwerten vorhanden sind.



Programmierung

Die Umsetzung des PAP in ein VB-Programm wird Ihnen umso leichter fallen, je besser Sie das Vorgängerbeispiel verstanden haben:

```
Module Module1
    Sub Main()
        Const Nmax As Integer = 10          ' maximale Anzahl von Werten
        Console.WriteLine("Maximumbestimmung einer Zahlenfolge") ' Überschrift
        Console.WriteLine()                 ' Leerzeile
        Dim arr(10) As Integer              ' Array dimensionieren
        Dim n, max As Integer               ' zwei Variablen deklarieren
        Console.Write("Geben Sie die gewünschte Anzahl von Werten ein: ")
        n = Convert.ToInt32(Console.ReadLine()) ' Eingabe n
        If (n > Nmax) Then                  ' Bedingung trifft zu
            Console.WriteLine("Es dürfen maximal nur " &
                Nmax.ToString() & " Werte eingegeben werden!")
        Else                                ' Bedingung trifft nicht zu
            For i As Integer = 1 To n      ' Beginn der Schleife
                Console.Write("Geben Sie die " & i.ToString() & ".Zahl ein: ")
                arr(i - 1) = Convert.ToInt32(Console.ReadLine()) ' Eingabe i-te Zahl in Array
                If (i = 1) Or arr(i - 1) > max Then max = arr(i - 1)
            Next i                          ' Ende der Schleife
            Console.WriteLine()
            Console.WriteLine("Das Maximum ist " & max.ToString() & " !") ' Ergebnis
        End If
        Console.ReadLine()
    End Sub
End Module
```

Test

Gleich nach Programmstart werden Sie zur Eingabe der Anzahl von Werten aufgefordert, die beliebig groß sein kann.

Bemerkungen

- Die Kommentare im Quelltext beschränken sich nur auf die Neuigkeiten gegenüber dem Vorgängerbeispiel, vor allem auf das Array *arr(10)* und die *For ... To ... Next*-Schleifenanweisung, die genau *n* mal durchlaufen wird.
- Ein Array-Index steht immer in runden Klammern hinter der Arrayvariablen. Der untere Index eines Arrays beginnt mit null, deshalb wird im Feld *arr(0)* die erste Zahl gespeichert.
- Die *ToString*-Funktion, über die jedes .NET-Objekt verfügt, verwandelt den Array-Index *i* (Datentyp *Integer*) in eine Zeichenkette (Datentyp *String*), damit eine Ausgabe über *Write* ermöglicht wird.

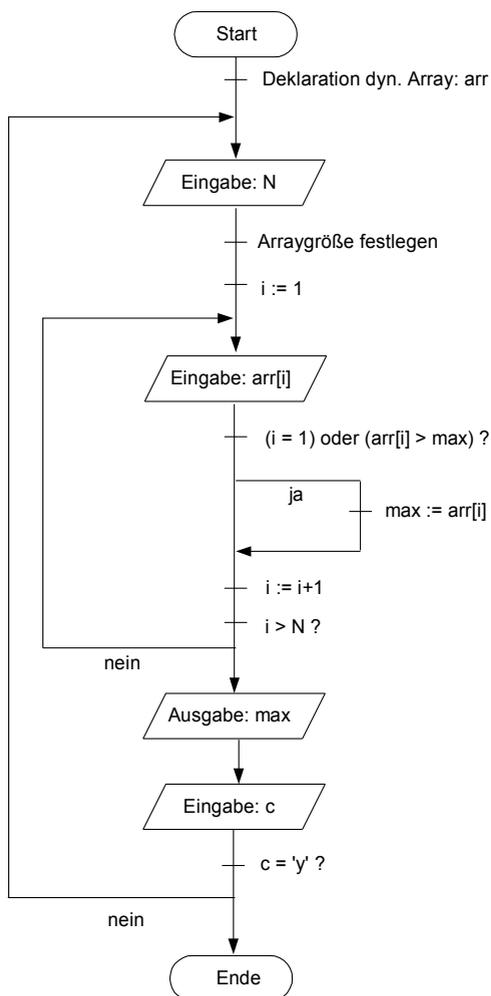
R3 Die Arraygröße zur Laufzeit ändern

Im Vorgängerbeispiel wurde das Programm nach einem einzigen Durchlauf beendet, danach war ein Neustart erforderlich.

Aufgabenstellung

Ergänzen Sie das Programm so, dass es nach Ermittlung des Maximums entweder wieder von vorn beginnt oder aber beendet werden kann!

Lösungsvorschlag



Programmierung

Öffnen Sie den Code-Editor und ergänzen Sie das Programm. Der Programmablauf weist viele Analogien zum Vorgängerbeispiel auf, deshalb wird nur auf die Neuigkeiten per Kommentar hingewiesen.

```

Module Module1
    Sub Main()
        Console.WriteLine("Maximum einer Zahlenfolge")      ' Überschrift
        Console.WriteLine()                                ' Leerzeile

        Dim arr() As Integer
        Dim n, max As Integer
        Dim c As String

        Do
            Console.Write("Geben Sie die gewünschte Anzahl von Werten ein: ")
            n = Convert.ToInt32(Console.ReadLine())          ' Eingabe n
            ReDim arr(n)
            For i As Integer = 1 To n
                Console.Write("Geben Sie die " & i.ToString() & ".Zahl ein: ")
                arr(i - 1) = Convert.ToInt32(Console.ReadLine()) ' Eingabe i-te Zahl
                If i = 1 Or arr(i - 1) > max Then max = arr(i - 1)
            Next i
            Console.WriteLine()
            Console.WriteLine("Das Maximum ist " & max.ToString() & " !") ' Ergebnis
            arr = Nothing
            Console.WriteLine()
            Console.Write("Wollen Sie das Programm beenden? (j/n) ")
            c = Console.ReadLine()
        Loop While c <> "j"

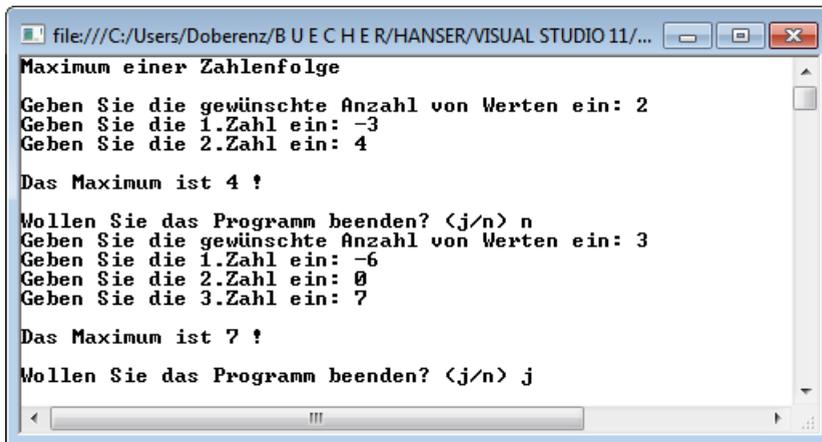
    End Sub
End Module

```

Test

Sie können den Berechnungszyklus jetzt beliebig oft wiederholen und dabei die Länge der einzugebenden Zahlenreihe neu festlegen (siehe folgende Abbildung).

HINWEIS: Nach Eingabe von "j" bzw. "n" müssen Sie die *Enter*-Taste drücken!



```
file:///C:/Users/Doberenz/BUECHER/HANSER/VISUAL STUDIO 11/...
Maximum einer Zahlenfolge
Geben Sie die gewünschte Anzahl von Werten ein: 2
Geben Sie die 1.Zahl ein: -3
Geben Sie die 2.Zahl ein: 4
Das Maximum ist 4 !
Wollen Sie das Programm beenden? <j/n> n
Geben Sie die gewünschte Anzahl von Werten ein: 3
Geben Sie die 1.Zahl ein: -6
Geben Sie die 2.Zahl ein: 0
Geben Sie die 3.Zahl ein: 7
Das Maximum ist 7 !
Wollen Sie das Programm beenden? <j/n> j
```

Bemerkungen

- Mit *Nothing* (eine Zeigervariable auf "nichts") wird der vom Array belegte Speicherplatz wieder freigegeben.
- Die *Do ... Loop While...*-Schleifenanweisung verlangt am Ende eine Abbruchbedingung.
- In unserem Beispiel dient die *String*-Variable *c* der Entgegennahme einer Benutzereingabe ("j" bzw. "n").

R4 Berechnungen in eine Methode auslagern

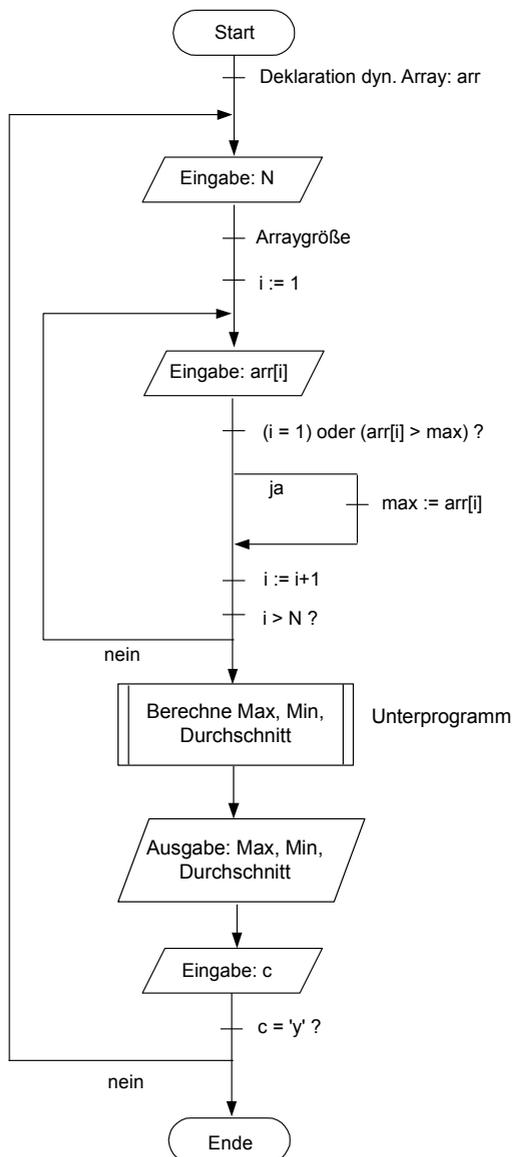
Es gibt kein Programm, das man nicht noch weiter verbessern könnte. Da sich von einer Zahlenreihe weitaus mehr Informationen gewinnen lassen als nur der Maximalwert, sollen Sie noch ein weiteres Problem lösen.

Aufgabenstellung

Ergänzen Sie das Programm, damit auch Minimum und Durchschnitt ausgegeben werden!

Lösungsvorschlag

Trotz der erweiterten Funktionalität erscheint der nachfolgend abgebildete PAP keinesfalls komplizierter als sein Vorgänger zu sein. Dies wurde vor allem durch Auslagern der Berechnungsfunktionen für Maximum, Minimum und Durchschnitt in ein Unterprogramm (eine Methode) erreicht.



Programmierung

Aufgrund der vielen Analogien zum Vorgängerbeispiel wird auch hier nur auf die Besonderheiten per Kommentar hingewiesen:

```

Module Module1
    ' globale Variablen deklarieren:
    Private arr() As Integer
    Private max, min As Integer
    Private av As String
  
```

```

Private Sub berechne()                                ' hier beginnt die Methode
    Dim sum As Double = arr(0)                        ' lokale Variable
    min = arr(0)
    max = arr(0) ' globale Variablen initialisieren
    Dim n As Integer = arr.Length - 1
    For i As Integer = 1 To n
        If arr(i) > max Then
            max = arr(i)
        Else
            If arr(i) < min Then min = arr(i)
        End If
        sum += arr(i)
    Next i
    Dim d As Double = sum / (n + 1)                   ' Durchschnitt berechnen
    av = d.ToString("#0.00")                          ' globale Variable zuweisen
End Sub                                               ' hier endet die Methode

```

```

Sub Main()
    Console.WriteLine("Auswerten einer Zahlenfolge") ' Überschrift
    Console.WriteLine()                               ' Leerzeile

    Dim n As Integer
    Dim c As String

    Do
        Console.Write("Geben Sie die gewünschte Anzahl von Werten ein: ")
        n = Convert.ToInt32(Console.ReadLine())      ' Eingabe n
        ReDim arr(n - 1)
        For i As Integer = 1 To n
            Console.Write("Geben Sie die " & i.ToString() & ".Zahl ein: ")
            arr(i - 1) = Convert.ToInt32(Console.ReadLine()) ' Eingabe i-te Zahl
        Next i
        Console.WriteLine()
        berechne() ' Zahlenfolge auswerten (Methodenaufruf)
        ' Globale Variablen anzeigen:
        Console.WriteLine("Das Maximum ist " & max.ToString() & " !")
        Console.WriteLine("Das Minimum ist " & min.ToString() & " !")
        Console.WriteLine("Der Durchschnitt ist " & av & " !")
        arr = Nothing
        Console.WriteLine()
        Console.Write("Wollen Sie das Programm beenden? (j/n) ")
        c = Console.ReadLine()
        Console.WriteLine()
    Loop While c <> "j"
End Sub

```

```
End Module
```

Test

Vergleichen Sie die Leistungsfähigkeit dieses Programms mit der ersten Version, so ist doch bereits ein deutlicher Unterschied festzustellen:

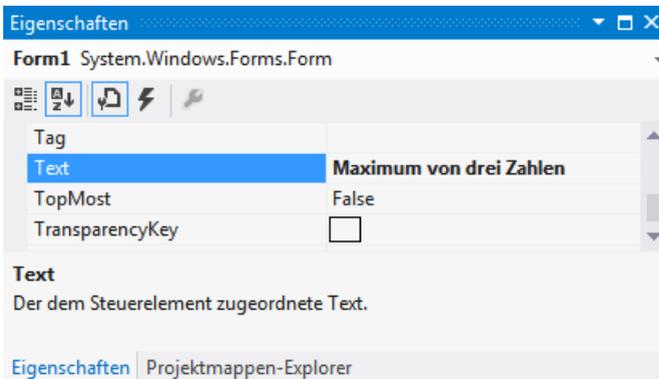
```

file:///C:/Users/Doberenz/BUECHER/HANSER/VISUAL STUDIO 11/...
Auswerten einer Zahlenfolge
Geben Sie die gewünschte Anzahl von Werten ein: 2
Geben Sie die 1.Zahl ein: -5
Geben Sie die 2.Zahl ein: 2
Das Maximum ist 2 !
Das Minimum ist -5 !
Der Durchschnitt ist -1,50 !
Wollen Sie das Programm beenden? <j/n> n
Geben Sie die gewünschte Anzahl von Werten ein: 3
Geben Sie die 1.Zahl ein: 6
Geben Sie die 2.Zahl ein: 0
Geben Sie die 3.Zahl ein: -1
Das Maximum ist 6 !
Das Minimum ist -1 !
Der Durchschnitt ist 1,67 !
Wollen Sie das Programm beenden? <j/n> j_

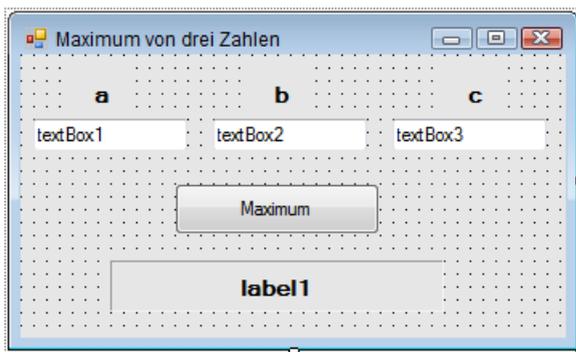
```

Bemerkungen

- In unserem Quellcode wurden Variablen auf globaler Ebene (gültig innerhalb des gesamten Programms) und auf lokaler Ebene (gültig innerhalb der Methode *berechne*) benutzt. Eine lokale Variable hat immer Vorrang vor einer gleichnamigen globalen Variablen.
- Die *ToString("#0.00")*-Methode besitzt hier einen so genannten Formatierungsstring als Argument, damit der Durchschnitt (eine Gleitkommazahl!) so in eine Zeichenkette umgeformt wird, dass immer zwei Nachkommastellen angezeigt werden.
- Die *Length*-Eigenschaft der *arr*-Variablen liefert die Anzahl der Array-Elemente. Da die Indizierung eines dynamischen Arrays stets mit null beginnt, ist der höchste Index immer um Eins niedriger.
- Ein Unterprogramm (hier die benutzerdefinierte Methode *berechne*) ist immer dann zweckmäßig, wenn die Übersichtlichkeit des Programms erhöht werden soll, oder aber wenn gleiche Codeabschnitte mehrfach ausgeführt werden sollen. Außerdem wird ein Wiederverwendbarkeit des Quellcodes erleichtert.
- Nicht nur der trostlose Textbildschirm, auch die mühselige Bedienung, bei der der Rechner die Reihenfolge der Benutzereingaben zwangsweise vorgibt, wird Ihnen missfallen haben. Dies sollte ein weiterer wichtiger Grund dafür sein, von der Konsolen- zur Windows-Anwendung überzugehen (siehe folgende Beispiele).



Vom Werkzeugkasten (*Strg+Alt+X*) ziehen Sie die Steuerelemente (3 mal *TextBox*, 1 mal *Button*, 4 mal *Label*) gemäß folgender Abbildung auf *Form1* und stellen auch hier bestimmte *Text*-Eigenschaften neu ein:



Programmierung

Durch einen Doppelklick auf *Button1* wird automatisch das Codefenster der Klasse *Form1* mit dem bereits vorbereiteten Rahmencode des *Click*-Eventhandlers geöffnet. In diesem Zusammenhang ist ein für den Einsteiger wichtiger Hinweis zu beachten, der auch für die Zukunft gilt:

HINWEIS: In der Regel sollten Sie den Rahmencode der Eventhandler nicht selbst eintippen, sondern immer auf die oben genannte Weise durch Visual Studio automatisch erzeugen lassen!

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        ' Hier müssen Ihre VB-Anweisungen eingefügt werden!
    End Sub
End Class
```

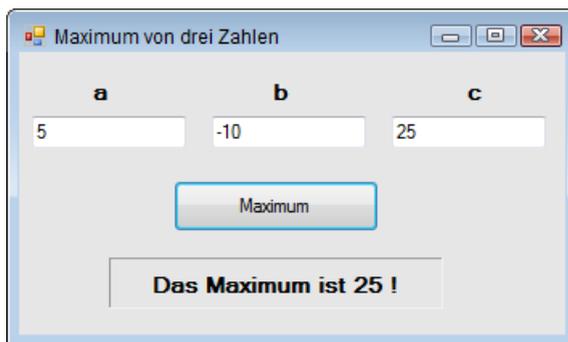
Füllen Sie den Körper des obigen Eventhandlers mit den erforderlichen Anweisungen aus, sodass der komplette Eventhandler schließlich folgendermaßen aussieht:

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    Try
        Dim a As Integer = Convert.ToInt32(TextBox1.Text) ' Eingabe a
        Dim max As Integer = a ' Initialisieren von max
        Dim b As Integer = Convert.ToInt32(TextBox2.Text) ' Eingabe b
        If b > max Then max = b ' Bedingung
        Dim c As Integer = Convert.ToInt32(TextBox3.Text) ' Eingabe c
        If c > max Then max = c ' Bedingung
        Label1.Text = "Das Maximum ist " & max.ToString & " !" ' Ergebnisausgabe
    Catch ex As Exception
        MessageBox.Show(ex.Message, "Fehler") ' Fehlermeldung
    End Try
End Sub
```

HINWEIS: Beim Vergleich mit der Konsolenanwendung erkennen Sie, dass Ein- und Ausgabe deutlich einfacher und übersichtlicher geworden sind!

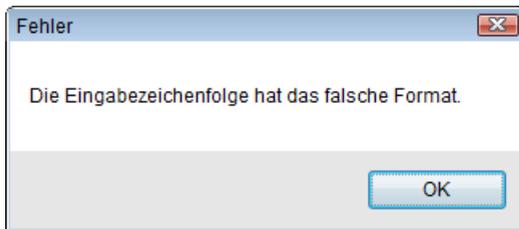
Test

Nachdem Sie das Projekt abgespeichert haben, kompilieren und starten Sie das Programm mit der *F5*-Taste (oder durch Klick auf die entsprechende kleine Schaltfläche mit dem grünen Dreieck auf der Symbolleiste von Visual Studio):



HINWEIS: Sie können mit der *Tab*-Taste den Eingabefokus zwischen den einzelnen Komponenten der Bedienoberfläche verschieben!

In unserem Beispiel darf man nur Ganzzahlen eingeben. Dank Fehlerbehandlung mittels *Try-Catch*-Block stürzt das Programm aber bei falschen Eingaben nicht ab, sondern liefert eine höfliche Meldung:



Bemerkungen

- Wenn Sie die Programmbedienung mit der bei einer Konsolenanwendung vergleichen, so stoßen Sie auf ein Hauptmerkmal der Windows-Programmierung: Eine fest vorgeschriebene Reihenfolge für die Benutzereingaben gibt es nicht mehr!
- Genauso wie im Konsolen-Beispiel haben wir es auch hier mit VB-Anweisungen zu tun, die sich jetzt allerdings innerhalb einer Ereignisbehandlungsmethode (Event-Handler) befinden.
- Im Unterschied zur Konsolenanwendung (*ReadLine/WriteLine*) ist die Programmierung der Ein-/Ausgabe der Zahlen etwas umständlicher. Deren Werte sind zunächst in der *Text*-Eigenschaft der drei Textboxen enthalten und müssen mit Hilfe der *Convert*-Klasse vom Datentyp *String* in den *Integer*-Datentyp umgewandelt werden. Die Ergebnisausgabe erfolgt umgekehrt mittels der bereits bekannten *ToString*-Methode, deren Ergebnis der *Text*-Eigenschaft von *Label* zugewiesen wird.

R6 Werte in einer ListBox anzeigen

Mittlerweile sind Sie auf den Geschmack gekommen und wollen sich an eine etwas anspruchsvollere Windows-Anwendung heranwagen. Schließlich verfügt Visual Studio nicht nur über solche einfache Steuerelemente wie *Label*, *TextBox* und *Button*, sondern über ein ganzes Arsenal attraktiver und leistungsfähiger Controls.

Aufgabenstellung

Verwandeln Sie das Konsolenprogramm aus

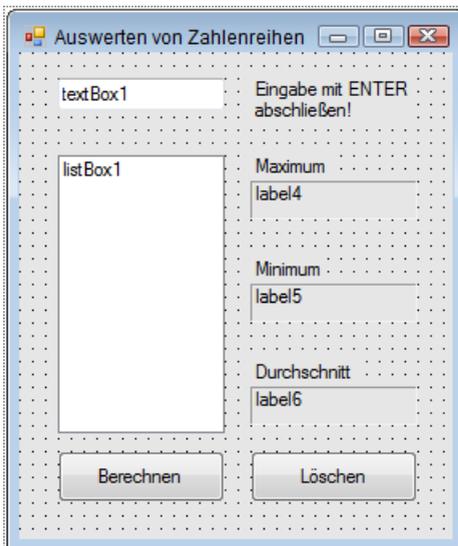
- R4 Berechnungen in eine Methode auslagern

in eine Windows Forms-Applikation und zeigen Sie die Zahlenreihe in einer *ListBox* an. Für die Eingabe sollen nicht nur ganze Zahlen, sondern auch Gleitkommazahlen zulässig sein.

Lösungsvorschlag

Da Sie die allgemeine Vorgehensweise bereits am Vorgängerbeispiel geübt haben, werden sich die folgenden Erläuterungen nur auf das Spezifische beschränken.

Die folgende Abbildung zur Gestaltung der Benutzerschnittstelle ist lediglich als Anregung zu verstehen. Eine *TextBox* dient zur Eingabe der Zahlen, die per *Enter*-Taste in eine *ListBox* übernommen werden sollen. Für die bequeme Bedienung und Anzeige sind weiterhin zwei *Button*- und drei *Label*-Steuerelemente vorgesehen.



Programmierung

In die Klasse *Form1* tragen Sie zunächst folgenden Code ein, wie er Ihnen bereits in ähnlicher Gestalt aus der entsprechenden Konsolenanwendung bekannt ist:

```
Public Class Form1
    Private arr() As Double           ' globale Variable deklarieren
    Private max, min As Double
    Private av As String

    Private Sub berechne()
        Dim sum As Double = arr(0)   ' lokale Variable
        min = arr(0)
        max = arr(0)                 ' globale Variable initialisieren
        Dim n As Integer = arr.Length - 1
        For i As Integer = 1 To n
            If arr(i) > max Then
                max = arr(i)
            Else

```

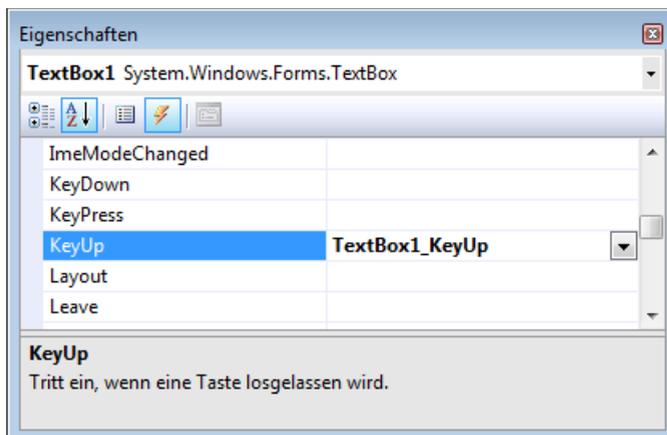
```

        If arr(i) < min Then min = arr(i)
    End If
    sum += arr(i)
Next i
sum = sum / (n + 1)      ' Durchschnitt berechnen
av = sum.ToString("#0.00") ' globale Variable zuweisen
End Sub

```

Der folgende Event-Handler für das *KeyUp*-Ereignis von *TextBox1* überträgt dann den Eintrag nach *ListBox1*, wenn die Enter-Taste gedrückt wird. Anschließend wird der Eintrag in der Textbox gelöscht, um für neue Zahleneingaben bereit zu sein.

HINWEIS: Tippen Sie den Rahmencode dieser Event-Handler nicht per Hand ein, sondern erzeugen Sie ihn über die *Ereignisse*-Seite des Eigenschaftfensters (*F4*)!



```

Private Sub TextBox1_KeyUp(sender As Object, e As KeyEventArgs) Handles TextBox1.KeyUp
    If (e.KeyCode = Keys.Enter) And (TextBox1.Text <> String.Empty) Then      ' Enter-Taste
        ListBox1.Items.Add(TextBox1.Text)      ' TextBox => ListBox
        TextBox1.Text = String.Empty          ' TextBox-Inhalt löschen
    End If
End Sub

```

Ein weiterer Event-Handler wertet das *Click*-Ereignis von *Button1* aus. Der Inhalt der *ListBox* wird ausgelesen und in das Array kopiert. Da der erste Eintrag einer *ListBox* (genauso wie das erste Feld des Arrays) immer den Index null hat, kann man das Kopieren elegant in einer *For...Next*-Schleife erledigen. Nach Aufruf der Methode *berechne* erfolgt dann die Ergebnisanzeige:

```

Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    Dim n As Integer = ListBox1.Items.Count      ' Anzahl der ListBox-Einträge
    If n > 0 Then                                ' ListBox hat mindestens einen Eintrag
        ReDim arr(n - 1)                        ' Array dimensionieren
        For i As Integer = 0 To n - 1

```

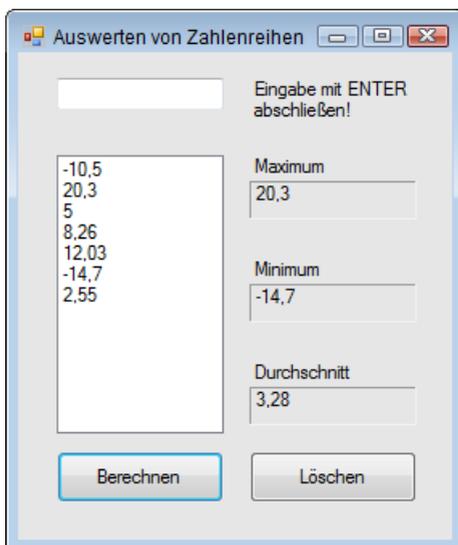
```
Try
    arr(i) = Convert.ToDouble(ListBox1.Items(i))      ' ListBox => Array
Catch ex As Exception                               ' Fehlerbehandlung
    MessageBox.Show(ex.Message, "Fehler")
End Try
Next i
berechne()                                         ' Methodenaufruf
Label1.Text = max.ToString()                       ' Ergebnisanzeige
Label2.Text = min.ToString()
Label3.Text = av                                   ' Gleitkommazahl formatieren
End If
End Sub
```

Der letzte Event-Handler ermöglicht das Löschen der gesamten Zahlenreihe, um wieder von vorn beginnen zu können:

```
Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
    ListBox1.Items.Clear()
    arr = Nothing                                  ' Array freigeben
    TextBox1.Text = String.Empty
    Label1.Text = String.Empty
    Label2.Text = String.Empty
    Label3.Text = String.Empty
End Sub
End Class
```

Test

Geben Sie eine beliebig lange Zahlenkolonne ein (jede Zahl mit *Enter*-Taste abschließen) und lassen Sie sich dann das Ergebnis der Auswertung anzeigen:



Bemerkungen

- Vergleichen Sie den Quellcode dieses Programms mit dem der Konsolenanwendungen aus den Vorgängerbeispielen. Nach einiger Praxis dürfte es Ihnen nicht schwer fallen, weitere "alte" Konsolenprogramme auf ähnliche Weise unter Visual Studio "aufzumöbeln". Dabei wird das "Zusammenschieben" der Benutzerschnittstelle nicht das Problem sein, da die Vorgehensweise starke Ähnlichkeiten mit einem Zeichenprogramm hat. Dreh- und Angelpunkt ist vielmehr das Aufbrechen des linearen Programmablaufs (PAP) und seine zweckmäßige Verteilung auf verschiedene Event-Handler.
- In einer Windows Forms-Anwendung hat der klassische Programmablaufplan (PAP) – zumindest außerhalb von Methoden bzw. Event-Handlern – seine Bedeutung weitestgehend verloren, da es keine festgelegte Reihenfolge der Benutzereingaben mehr gibt.
- Wenn Sie auch dieses letzte Beispiel ohne größere Schwierigkeiten gemeistert haben, sind Sie auf dem besten Weg zu einem hoffnungsvollen Visual Basic-Programmierer, denn Sie haben bereits ein Gefühl für die wichtigsten sprachlichen Grundlagen entwickelt. Die Zeit ist also reif, um die sprachlichen Grundlagen zu vertiefen und sich an anspruchsvollere Projekte zu wagen.

Schnittstellen

R189 Eine einfache E-Mail versenden

Mit Hilfe der Anweisung "mailto:" ist es problemlos möglich, eine E-Mail mit Adressangabe, Betreffzeile und E-Mail-Text zu generieren, lediglich auf Dateianhänge müssen Sie verzichten. Aufgerufen wird die Anweisung mit Hilfe der *Start*-Methode eines *Process*-Objekts, siehe dazu

- ▶ R240 Ein externes Programm starten

HINWEIS: Das in R241 vorgestellte Programm können Sie für die folgenden Beispiele verwenden, Sie müssen jedoch die Zeile *"proc.WaitForExit()"* auskommentieren.

Folgende Varianten bieten sich an:

BEISPIEL: Eine einfache E-Mail ohne Betreffzeile oder Body-Text.

```
mailto:xyz@abc.com
```

BEISPIEL: Eine E-Mail mit einer Betreffzeile ("Preis-anfrage").

```
mailto:xyz@abc.com?subject=Preis-anfrage
```

BEISPIEL: Eine E-Mail mit Adresse, Betreffzeile und zusätzlicher Kopie an die Adresse "hans@glueck.com".

```
mailto:xyz@abc.com?subject=Preis-anfrage&CC=hans@glueck.com
```

BEISPIEL: Eine E-Mail mit Adresse, Betreffzeile sowie einem E-Mail-Text.

```
mailto:abc@xyz.de?subject=Anfrage Preisliste&Body=Bitte senden Sie mir %0A die aktuellen Preislisten zu!
```

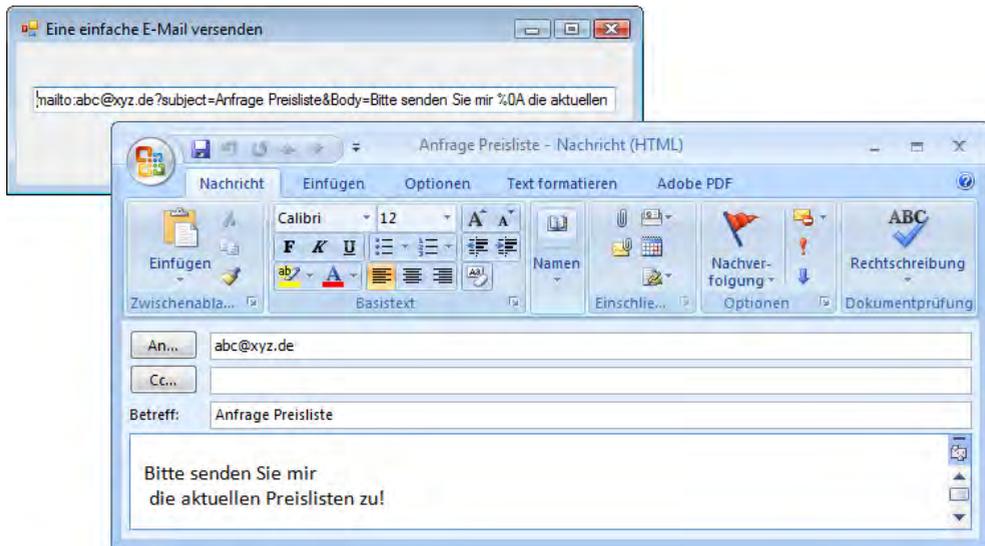
Wie Sie sehen, ist die Verwendung recht einfach. Es sind lediglich einige Grundregeln zu beachten:

- Die Betreffzeile (*subject*) ist mit einem Fragezeichen "?" von der Adressangabe zu trennen.
- Alle weiteren Optionen sind mit einem "&" voneinander zu trennen.

- Zeilenumbrüche in der Textangabe können Sie mit der Kombination "%0A" realisieren.
- Leerzeichen in der Adressangabe können Sie mit "%20" einfügen.

Test

Tragen Sie obige Anweisungen in das Programm ein und klicken Sie auf den *Start*-Button:



R190 E-Mails mit dem integrierten Mail-Client versenden

Der Ablauf für das Senden einer E-Mail in Windows Forms-Anwendungen entspricht der Vorgehensweise bei ASP.NET-Anwendungen, wie es in Rezept

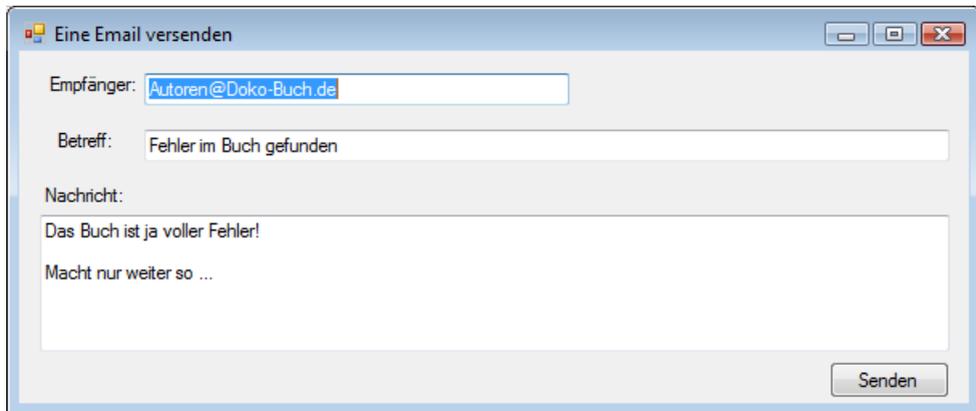
- R458 EMail-Versand in ASP.NET realisieren

ausführlich erläutert wurde.

An dieser Stelle wollen wir deshalb die theoretischen Grundlagen nicht erneut auswalzen, sondern mit einem kleinen Beispielprogramm die Umsetzung für Windows Forms-Anwendungen demonstrieren.

Oberfläche

Ein Windows *Form*, drei *TextBoxen* (Empfänger, Betreff, Nachricht) einige *Labels* für die Beschriftung sowie ein *Button* genügen:



Quelltext

```
Imports System.Net
```

```
Public Class Form1
```

```
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
```

Instanz eines E-Mail-Clients erzeugen:

```
        Dim mail As New System.Net.Mail.SmtpClient()
```

Die eigentliche Message:

```
        Dim msg As New System.Net.Mail.MailMessage("leser@deutschland.de", TextBox1.Text)
        msg.Subject = TextBox2.Text
        msg.Body = TextBox3.Text
        mail.Credentials = New NetworkCredential("Hans", "Wurst")
        mail.Host = "server"
```

E-Mail versenden:

```
        mail.Send(msg)
    End Sub
```

```
End Class
```

Test

Nach dem Klick sollte die Mail schon unterwegs sein ...

R191 Die Zwischenablage verwenden

Das *Clipboard*-Objekt kann mit einer ganzen Reihe von Methoden aufwarten, zu jedem der Datentypen *Data*, *Text*, *Audio*, *Image*, *FileDropList* gibt es eine *Contains*-, eine *Get*- und eine *Set*-Methode, z.B. *ContainsText*, *GetText* und *SetText*.

Eine *Contains*-Methode (*True/False*) überprüft, ob in der Zwischenablage eine Information im gewünschten Format vorliegt. Die *Get*- und *Set*-Methoden übernehmen das Kopieren bzw. Einfügen.

Ein kleines Beispiel zeigt wie es funktioniert.

Oberfläche

In ein Windows *Form* fügen Sie bitte eine *PictureBox*, eine *TextBox* (*MultiLine=True*) sowie vier Schaltflächen ein (siehe Laufzeitansicht).

Quelltext

```
Public Class Form1
```

Text in die Zwischenablage kopieren:

```
Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
    If TextBox1.SelectedText <> String.Empty Then
        Clipboard.SetText((TextBox1.SelectedText))
    Else
        MessageBox.Show("Kein Text selektiert!")
    End If
End Sub
```

Text aus der Zwischenablage einfügen:

```
Private Sub Button4_Click(sender As Object, e As EventArgs) Handles Button4.Click
    If Clipboard.ContainsText() Then
        TextBox1.Text = Clipboard.GetText()
    Else
        MessageBox.Show("Keine geeigneten Daten im Clipboard!")
    End If
End Sub
```

Grafik in die Zwischenablage kopieren:

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    If PictureBox1.Image IsNot Nothing Then
        Clipboard.SetImage(PictureBox1.Image)
    Else
        MessageBox.Show("Keine Grafik enthalten!")
    End If
End Sub
```

Grafik aus der Zwischenablage einfügen:

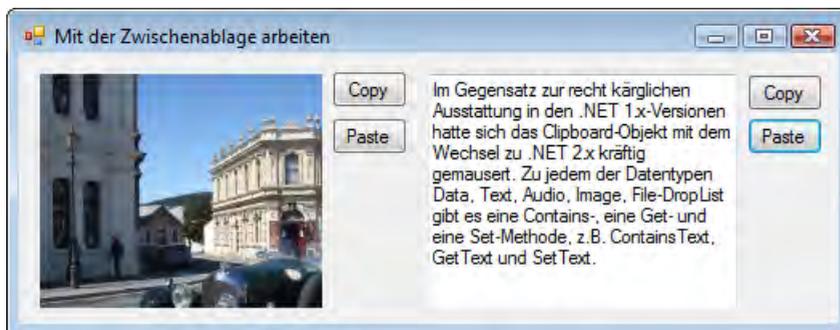
```
Private Sub Button3_Click(sender As Object, e As EventArgs) Handles Button3.Click
    If Clipboard.ContainsImage() Then
        Dim pic As System.Drawing.Image = Clipboard.GetImage()
        PictureBox1.Image = pic
    Else
        MessageBox.Show("Keine Bitmap im Clipboard!")
    End If
End Sub
```

```
End Class
```

Test

Kopieren Sie mit einem Zeichenprogramm (z.B. Paint) eine Grafik in die Zwischenablage und versuchen Sie, diese in das Programm einzufügen.

HINWEIS: Testen Sie auch was passiert, wenn keine Grafik/kein Text vorhanden ist.



R192 Die WIA-Library kennenlernen

Geht es um den Zugriff auf externe Gerate, wie Scanner, Digitalkameras, Webcams etc., bietet sich dem Programmierer mit der Microsoft WIA-Library (*Windows Image Acquisition Automation Layer*) eine recht komfortable Variante an, deren Vorteil unter anderem darin besteht, ohne viel VBCode auszukommen.

Doch worum geht es eigentlich? Bei WIA handelt es sich um eine komplexe Windows-COM-API, die

- das Auslesen statischer Bilder von Scannern, digitalen Kameras, WebCams,
- das Nachbearbeiten (Groe, Farbe, Drehen ...),

- das Kommentieren (Exif-Informationen)
- und das Konvertieren in diverse Dateiformate (PNG, JPG, GIF, BMP, TIFF)

ermöglicht.

HINWEIS: WIA steht dem Programmierer als COM-Komponente in der Datei *wiaaut.dll* zur Verfügung.

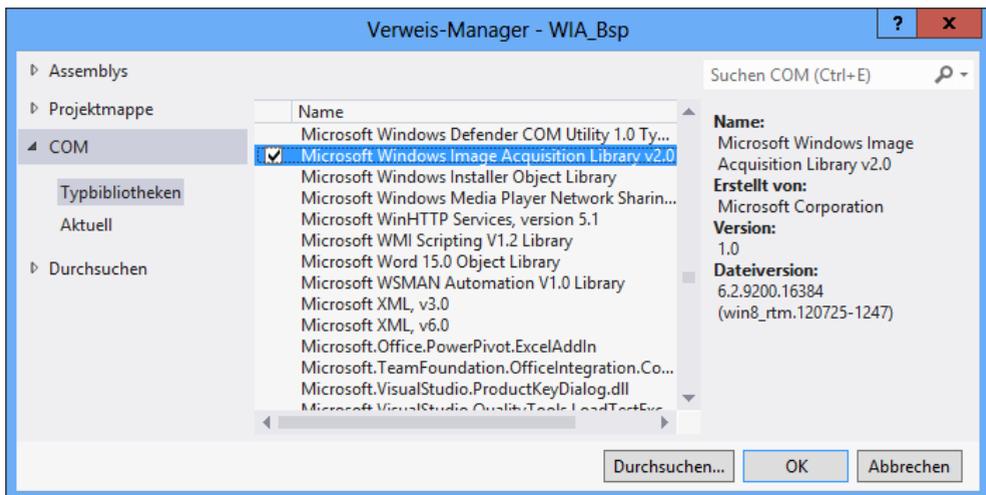
Installation/Vorbereitung

Die Unterstützung beschränkt sich auf die Windows-Version XP ab SP1 und alle später erschienenen Windows Versionen (Vista, 7, 8, Server ...).

Windows Vista/Windows 7/Windows 8

Die komplette und aktuelle WIA-Library ist bereits auf Ihrem System installiert, Sie können sofort loslegen, indem Sie die *Microsoft Windows Image Acquisition Library v2.0* in Ihr VB-Projekt einbinden. Erstellen Sie dazu ein neues Projekt und fügen Sie die o.g. Library über *Projekt|Verweis hinzufügen* hinzu (siehe folgende Abbildung).

HINWEIS: Ab Visual Studio 2010 wird die PIA¹ (bzw. die nötigen Datentypen) automatisch in die Anwendung eingebunden, eine Weitergabe der extra Assembly ist also nicht mehr nötig.



¹ Primary Interop Assembly

Windows XP

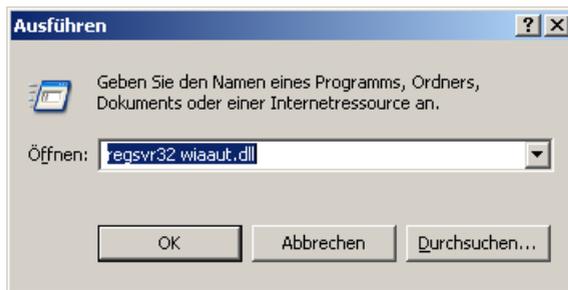
Hier ist noch etwas Nacharbeit angesagt, da sich auf Ihrem System derzeit nur die Version 1.x befindet. Laden Sie zunächst die aktuelle Version der WIA von der Microsoft-Webseite:

LINK: <http://www.microsoft.com/downloads/details.aspx?FamilyID=a332a77a-01b8-4de6-91c2-b7ea32537e29>

Entpacken Sie die ZIP-Datei und kopieren Sie enthaltene Datei *wiaaut.dll* in das `\\Windows\System32`-Verzeichnis.

HINWEIS: Ganz nebenbei finden Sie in dieser Datei auch einige VB6-Beispiele und eine umfangreiche Hilfedatei.

Die neue Library registrieren Sie über *Start/Ausführen*:



Binden Sie anschließend die Library, bzw. eine Referenz auf diese (siehe vorhergehender Abschnitt), in Ihr Projekt ein.

Einstieg mit kleiner Beispielanwendung

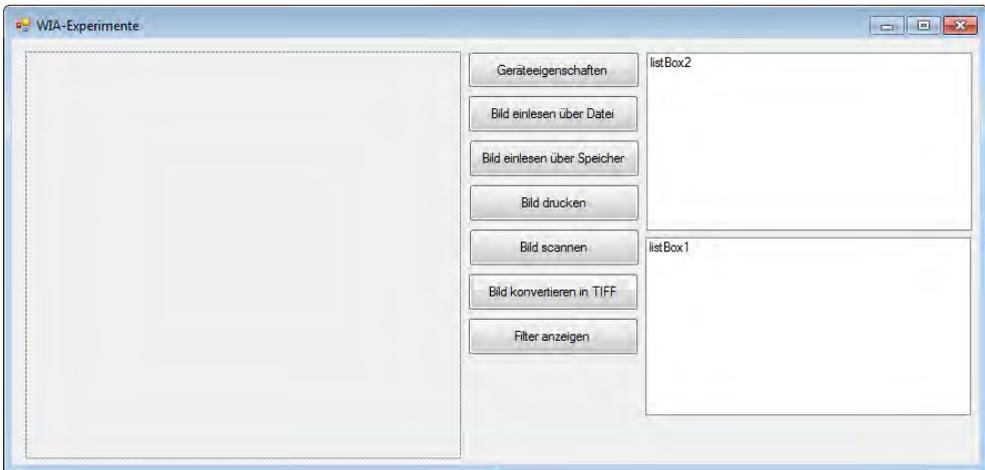
Auf Grund der umfangreichen Möglichkeiten dieser Library möchten wir Ihnen anhand einer kleinen Beispielanwendung einige wesentliche Features, die für das Einlesen und Weiterverarbeiten von Bildern unabdingbar sind, vorstellen:

- Erkennen der angeschlossenen Geräte/Gerätetypen
- Auswerten von Ereignissen (Anschließen/Trennen)
- Anzeige der Geräteeigenschaften
- Einlesen von Bildern
- Verwendung des Scan-Assistenten
- Einsatz des Druckassistenten

Oberfläche

Entwerfen Sie zunächst eine Oberfläche entsprechend der folgenden Abbildung. Fügen Sie dazu zwei *ListBoxen* (Anzeige von Statusmeldungen, Anzeige der aktiven Geräte) sowie ein *PictureBox*-Steuerelement in das Formular ein. Die Bedeutung der Schaltflächen dürfte aus der Beschriftung ersichtlich sein.

Abschließend fügen Sie noch eine *OpenFileDialog*-Komponente ein, mit der wir Bilddateien auswählen wollen.



Quellcode

Ein wichtiger Hinweis vorweg:

HINWEIS: Erzeugen Sie so wenig globale Objekte wie möglich, es besteht immer die Möglichkeit, dass der Anwender die externen Geräte vom PC trennt bzw. diese ausschaltet. In diesem Fall ist die Referenz auf das entsprechende Objekt futsch.

```
Imports System.IO
Imports System.Diagnostics
Imports System.Drawing.Imaging
```

Das zentrale Objekt für den Zugriff auf alle Geräte ist der *DeviceManager*:

```
Public Class Form1
    Dim dm As WIA.DeviceManager
```

Um die PIA-Klassen einbinden zu können, definieren wir einige wichtige Konstanten direkt, andernfalls müssten wir die externe PIA mitgeben:

```
Const wiaEventDeviceConnected As String = "{A28BBADE-64B6-11D2-A231-00C04FA31809}"
Const wiaEventDeviceDisconnected As String = "{143E4E83-6497-11D2-A231-00C04FA31809}"
```

```

Const wiaFormatBMP As String = "{B96B3CAB-0728-11D3-9D7B-0000F81EF32E}"
Const wiaFormatPNG As String = "{B96B3CAF-0728-11D3-9D7B-0000F81EF32E}"
Const wiaFormatGIF As String = "{B96B3CB0-0728-11D3-9D7B-0000F81EF32E}"
Const wiaFormatJPEG As String = "{B96B3CAE-0728-11D3-9D7B-0000F81EF32E}"
Const wiaFormatTIFF As String = "{B96B3CB1-0728-11D3-9D7B-0000F81EF32E}"

```

Mit dem Laden des Formulars wird zunächst der *DeviceManager* instanziiert, nachfolgend können zwei wichtige Ereignisse angemeldet werden:

```

Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    dm = New WIA.DeviceManager()
    dm.RegisterEvent(wiaEventDeviceConnected, "*")
    dm.RegisterEvent(wiaEventDeviceDisconnected, "*")
    AddHandler dm.OnEvent, AddressOf dm_OnEvent
    anzeige()
End Sub

```

HINWEIS: Die Verwendung von *RegisterEvent* ist unbedingt erforderlich, andernfalls wird das betreffende Ereignis nicht an die Ereignismethode weitergeleitet.

Reagieren auf das Verbinden/Trennen von Geräten

Nutzen Sie die folgende Ereignisbehandlung, um mit Ihrer Anwendung auf das Hinzufügen bzw. Entfernen von WIA-Geräten zu reagieren:

```

Private Sub dm_OnEvent(EventID As String, DeviceID As String, ItemID As String)
    If EventID = wiaEventDeviceDisconnected Then
        ListBox1.Items.Add("Gerät getrennt")
    End If
    If EventID = wiaEventDeviceConnected Then
        ListBox1.Items.Add("Gerät angeschlossen")
    End If
    anzeige()
End Sub

```

HINWEIS: Nur angemeldete Ereignisse werden ausgelöst!

Ermitteln der verfügbaren Geräte

Die folgende Routine aktualisiert die Anzeige der vorhandenen Geräte. Dazu werden alle vom *DeviceManager* in der Collection *DeviceInfos* aufgelisteten Geräte abgefragt und mit *Connect* instanziiert:

```

Private Sub anzeige()
    Dim dev As WIA.Device
    ListBox2.Items.Clear()
    For Each di As WIA.DeviceInfo In dm.DeviceInfos

```

```
dev = di.Connect()
```

Konnte eine Instanz gebildet werden, wird der Name des Geräts in die Liste eingetragen:

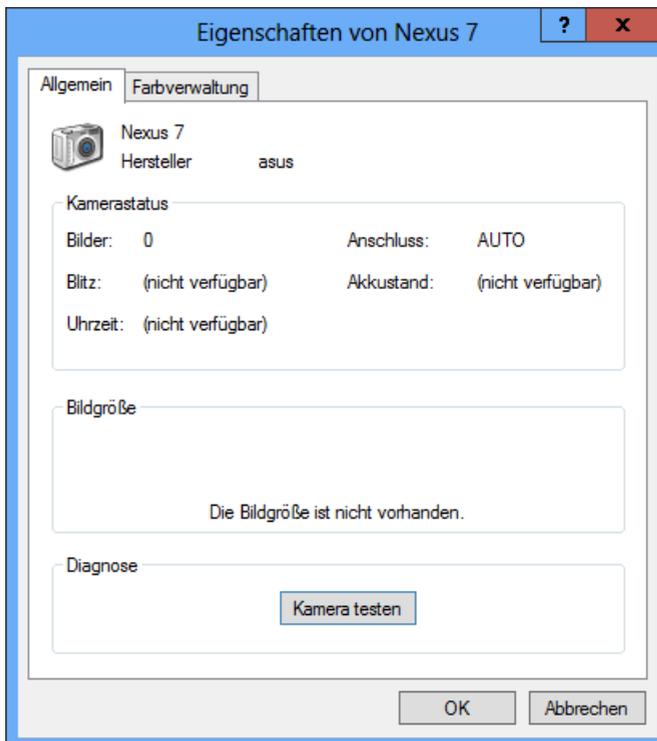
```
If dev IsNot Nothing Then ListBox2.Items.Add(di.Properties("Name").Value)
Next di
```

Die Schaltflächen unserer Anwendung werden nur freigegeben, wenn mindestens ein Gerät installiert ist:

```
Button1.Enabled = ListBox2.Items.Count > 0
Button2.Enabled = ListBox2.Items.Count > 0
Button4.Enabled = ListBox2.Items.Count > 0
Button5.Enabled = ListBox2.Items.Count > 0
If ListBox2.Items.Count > 0 Then ListBox2.SelectedIndex = 0
End Sub
```

Anzeige der Geräteeigenschaften

Erfreulicherweise stellt die WIA-Library bereits einen großen Fundus an Dialogen bereit. Unter anderem befindet sich darunter auch ein Dialog zur Anzeige der Geräteeigenschaften, den Sie mit einem *CommonDialog*-Objekt (nicht mit den entsprechenden Dateidialogen verwechseln) und der Methode *ShowDeviceProperties* anzeigen können:



```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    Dim dev As WIA.Device
    Dim dlg As New WIA.CommonDialog()
```

Zunächst wird eine *Device*-Instanz erzeugt (ist dem physischen Gerät zugeordnet):

```
    dev = dm.DeviceInfos(ListBox2.SelectedIndex + 1).Connect()
    dlg.ShowDeviceProperties(dev)
End Sub
```

Alternativ können Sie auch einzelne Eigenschaften über die *Properties*-Auflistung abrufen:

BEISPIEL: Ausgabe der Properties in einer *ListBox*

```
    dev = dm.DeviceInfos(ListBox2.SelectedIndex + 1).Connect()

    For Each prop As WIA.Property In dev.Properties
        ListBox1.Items.Add(prop.Name & ": " & prop.Value)
    Next prop
```

Die Ausgabe:

```
Item Name: Root
Full Item Name: 0018\Root
Item Flags: 76
Unique Device ID: {EEC5AD98-8080-425f-922A-DABF3C}
Manufacturer: MATSHITA
Description: E:\
Type: 131072
Port: AUTO
Name: E:\
Server: local
Remote Device ID:
UI Class ID: {00000000-0000-0000-0000-000000000000}
```

Ein Bild einlesen

Das eigentliche Ziel unserer Bemühungen war und ist immer noch das Einlesen eines Bildes, was wir jetzt in Angriff nehmen.

Wie schon beim vorhergehenden Anzeigen von Eigenschaften können wir auch hier auf einen integrierten Dialog zugreifen, mit dem der Anwender das eigentliche Bild auswählt bzw. die Eigenschaften einstellt (siehe folgende Abbildung).



```
Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
    Dim dlg As New WIA.CommonDialog()
    Dim img As WIA.ImageFile
    img = dlg.ShowAcquireImage()
```

Der Rückgabewert ist ein *ImageFile*-Objekt, dessen Dateiformat Sie über die *FileExtension*-Eigenschaft in Erfahrung bringen können.

Für die weitere Vorgehensweise bieten sich zwei Varianten an:

- Sie speichern das Bild als Datei und laden diese gegebenenfalls später.
- Sie übertragen direkt die binären Bilddaten an Ihre Anwendung und nutzen diese beispielsweise für eine *PictureBox*.

An dieser Stelle beschränken wir uns zunächst auf die erste Variante:

```
Dim TempFileName As String = Path.Combine(Path.GetTempPath(),
    Path.GetRandomFileName() & "." & img.FileExtension)
img.SaveFile(TempFileName)
Using fs As New FileStream(TempFileName, FileMode.Open, FileAccess.Read)
    pictureBox1.Image = Image.FromStream(fs)
    fs.Close()
End Using
File.Delete(TempFileName)
End Sub
```

HINWEIS: Die Verwendung der *Image.FromFile*-Methode wäre zwar mit weniger Schreibaufwand verbunden, allerdings haben Sie nachfolgend keinen Zugriff auf die Datei und somit können Sie diese temporäre Datei auch nicht löschen.

Alternativ hier der Weg über einen *MemoryStream*:

```
Dim dlg As New WIA.CommonDialog()
Dim img As WIA.ImageFile
img = dlg.ShowAcquireImage()
```

Hier ermitteln wir die Bilddaten,

```
Dim v As WIA.Vector = img.FileData
```

kopieren diese in ein Byte-Array

```
Dim bytes() As Byte = CType(v.BinaryData(), Byte())
```

und wandeln dieses in einen *MemoryStream* um:

```
Dim ms As New MemoryStream(bytes)
```

Letzter Schritt ist die Anzeige des Bildes:

```
pictureBox1.Image = Image.FromStream(ms)
```

Haben Sie statt einer Digitalkamera einen Scanner angeschlossen, erscheint der bekannte Scan-Dialog von Windows¹.



¹ Unsere treuen Stammler werden sich vermutlich noch an unser allererstes Buch (siehe Abbildung) von 1995 erinnern.

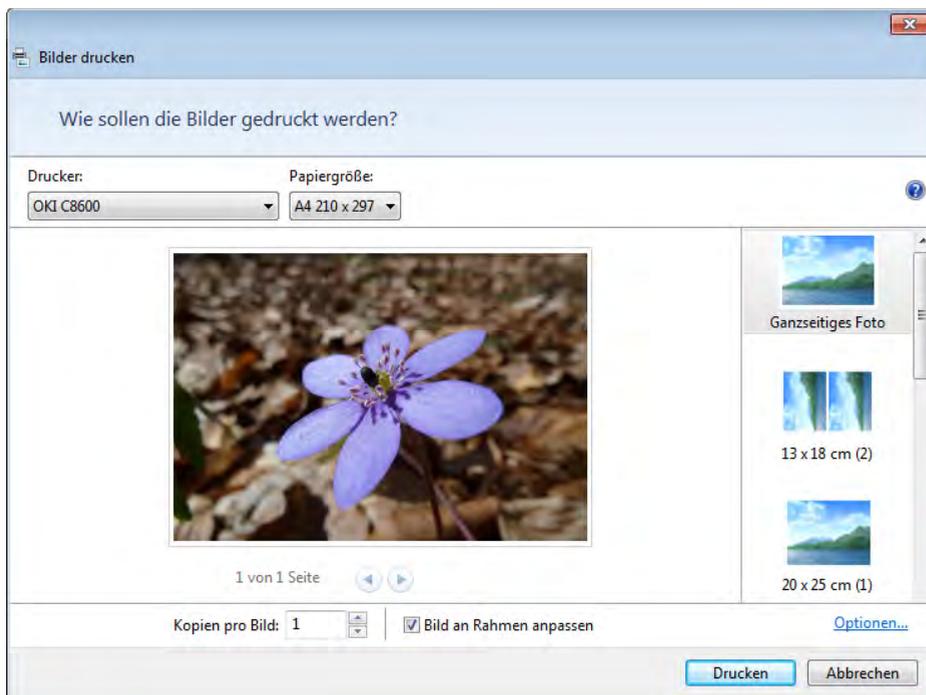
In diesem Dialog können Sie auch spezielle Einstellungen (Größe, DPI, Farbe) vornehmen.

Doch Vorsicht:

HINWEIS: Gerade hier sollten Sie auf den Datentransfer per Arbeitsspeicher verzichten, können doch die Bilder sehr groß werden..

Bild(er) drucken per Assistent

Neben der reinen Anzeige können Sie per WIA auch gleich die Ausgabe realisieren, ohne sich große Gedanken um ein ansprechendes Interface machen zu müssen. Ein entsprechender Assistent steht bereits "ab Werk" zur Verfügung:



```
Private Sub Button3_Click(sender As Object, e As EventArgs) Handles Button3.Click
    Dim dlg As New WIA.CommonDialog()

```

Wird ein Bild per *OpenFileDialog* ausgewählt:

```
If OpenFileDialog1.ShowDialog() = System.Windows.Forms.DialogResult.OK Then

```

... zeigen wir den Windows-Druckassistenten an:

```

        dlg.ShowPhotoPrintingWizard(OpenFileDialog1.FileName)
    Else
        MessageBox.Show("Kein Bild gewählt!")
    End If
End Sub

```

Den Scanner-Assistent aufrufen

Nicht genug der Assistenten, auch für das Scannen in eine Datei steht ein entsprechender Assistent bereit:

```

Private Sub Button4_Click(sender As Object, e As EventArgs) Handles Button4.Click
    Dim dev As WIA.Device
    Dim dlg As New WIA.CommonDialog()
    dev = dm.DeviceInfos(ListBox2.SelectedIndex + 1).Connect()

```

Nur wenn es sich um einen Scanner handelt, wird der Dialog angezeigt:

```

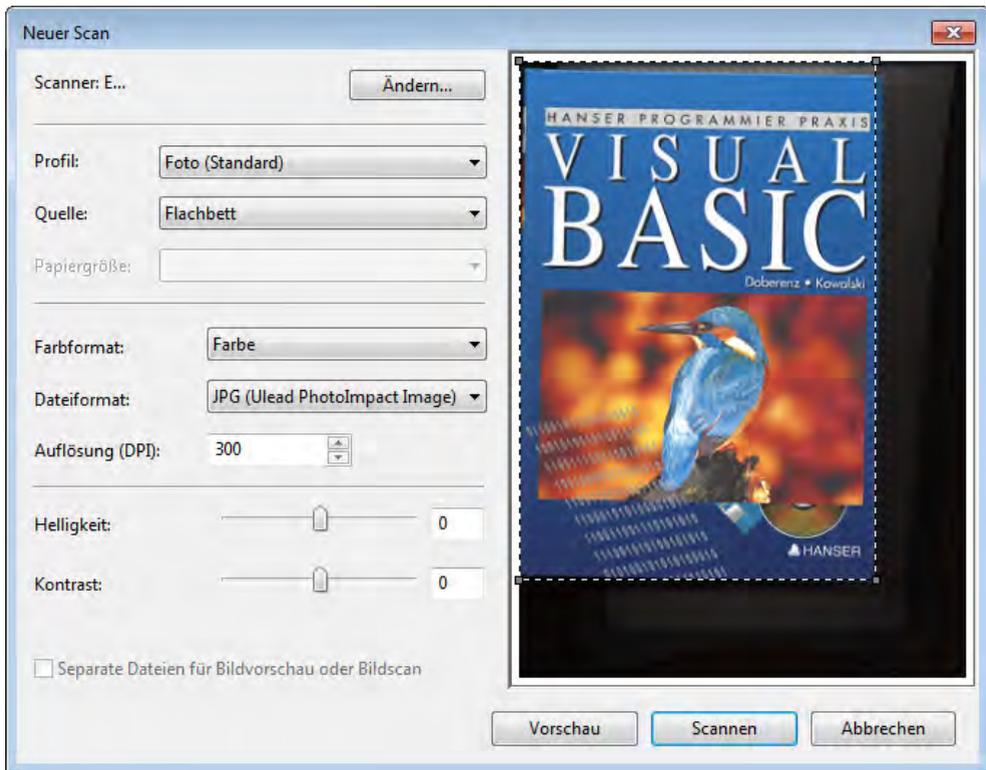
    If dev.Type = WIA.WiaDeviceType.ScannerDeviceType Then
        dlg.ShowAcquisitionWizard(dev)
    Else
        MessageBox.Show("Kein Scanner vorhanden!")
    End If
End Sub

```

Wie Sie sehen, können Sie per *Device.Type*-Eigenschaft den Gerätetyp in Erfahrung bringen:

Konstante	Bedeutung
<i>ScannerDeviceType</i>	Scanner, nur diese können Sie auch mit dem Scanner-Assistenten verwenden.
<i>CameraDeviceType</i>	Digitale Kameras die per WIA eingebunden sind. Achtung, hier haben Sie es meist mit Verzeichnisstrukturen zu tun!
<i>VideoDeviceType</i>	WebCam etc.
<i>UnspecifiedDeviceType</i>	Unbekannt, wird meist bei der Auswahl von Geräten angegeben, um alle Geräte abzufragen.

Der Scandialog unterscheidet sich in diesem Fall etwas vom *ShowAcquireImage*-Dialog (siehe folgende Abbildung).



Mehr dazu im Rezept

► R194 Auf den Scanner zugreifen

Grafikbearbeitung mit WIA

Neben dem reinen Import bietet sich WIA auch für die Bearbeitung von Bildern an. Nachdem diese in einem *ImageFile*-Objekt vorliegen, können diese mit dem *ImageProcess*-Objekt verarbeitet werden. Dazu erstellen Sie zunächst eine Instanz des *ImageProcess*-Objekts und weisen per *Filters.Add*-Methode spezielle Verarbeitungsfiler zu. Anschließend können Sie diese noch konfigurieren.

BEISPIEL: Konvertieren einer eingescannten Grafik (ins TIFF-Format)

```
Dim dlg As New WIA.CommonDialog()
Dim img As New WIA.ImageFile()
Dim proc As WIA.ImageProcess
```

```
If OpenFileDialog1.ShowDialog() = System.Windows.Forms.DialogResult.OK Then
```

Bild einlesen:

```
img.LoadFile(OpenFileDialog1.FileName)
```

Konverter erzeugen:

```
proc = New WIA.ImageProcess()
```

Filter zuweisen:

```
proc.Filters.Add(proc.FilterInfos("Convert").FilterID)
```

Filter konfigurieren

```
proc.Filters(1).Properties("FormatID").Value = wiaFormatTIFF
```

Filter anwenden:

```
img = proc.Apply(img)  
img.SaveFile("c:\test.tiff")  
End If  
End Sub
```

BEISPIEL: Verfügbare Filter anzeigen

```
Private Sub Button7_Click(sender As Object, e As EventArgs) _  
    Handles Button7.Click  
    Dim proc As New WIA.ImageProcess()  
    For Each fil As WIA.FilterInfo In proc.FilterInfos  
        ListBox1.Items.Add(fil.Name)  
    Next fil  
End Sub
```

BEISPIEL: Namen der verfügbaren Filter:



```
RotateFlip  
Crop  
Scale  
Stamp  
Exif  
Frame  
ARGB  
Convert
```

Natürlich können Sie mit den WIA-Objekten noch weit mehr anfangen, aber für einen ersten Einstieg dürften die bisherigen Ausführungen sicherlich genügen.

R193 Auf eine Webcam zugreifen

Haben Sie schon einmal daran gedacht, Bilder Ihrer Webcam per VB-Programm abzurufen und gegebenenfalls auf einem Server abzulegen? Wenn ja, finden Sie mit Hilfe der Microsoft WIA-Library eine recht einfache Lösung.

Zu den Grundlagen der WIA siehe Rezept

► R192 Die WIA-Library kennenlernen

Oberfläche

Ein einfaches Windows-Form mit einer *PictureBox* und einem *Timer*, den Sie bitte auf einen nicht zu kleinen Wert einstellen (z.B. 10000 für 10 Sekunden).

HINWEIS: Binden Sie zusätzlich die "Microsoft Windows Image Acquisition Library v2.0" in Ihr VB-Projekt ein (*Projekt|Verweis hinzufügen|COM*).

Quelltext

Binden Sie für die Stream-Unterstützung den folgenden Namespace ein:

```
Imports System.IO
```

```
Public Class Form1
```

Die Konstante für die Aufnahme von Bildern:

```
Private Const wiaCommandTakePicture As String = "{AF933CAC-ACAD-11D2-A093-00C04F72DC3C}"
```

Eine Referenz für das WIA-Gerät, in unserem Fall eine Webcam:

```
Private Camera As WIA.Device
```

```
...
```

Mit dem Laden des Formulars wählen wir zunächst das gewünschte Gerät aus. Sind mehrere Geräte vorhanden, wird ein Auswahldialog angezeigt, sonst wird automatisch das erste Gerät gewählt:

```
Private Sub Form1_Load(sender As Object, e As EventArgs) _
    Handles MyBase.Load
```

```
Try
    Camera = (New WIA.CommonDialog()).ShowSelectDevice()
```

Testen auf VideoDevice:

```
If Camera.Type = WIA.WiaDeviceType.VideoDeviceType Then
    timer1.Enabled = True
```

```
Else
    MessageBox.Show("Keine Webcam vorhanden!")
    Application.Exit()
End If
```

```
Catch e1 As Exception
    MessageBox.Show("Keine Webcam vorhanden!")
    Application.Exit()
End Try
```

```
End Sub
```

Ist die Zeit abgelaufen, wird der Schnappschuss ausgelöst:

```
Private Sub timer1_Tick(sender As Object, e As EventArgs) Handles timer1.Tick
    timer1.Enabled = False
    Me.Text = "ACHTUNG: AUFNAHME"
```

Ein Kommando an das Gerät senden:

```
Dim item As WIA.Item = Camera.ExecuteCommand(wiaCommandTakePicture)
```

Das erzeugte Bild abrufen und per Speicher an die *PictureBox* weiterleiten:

```
Dim img As WIA.ImageFile = item.Transfer()
If img IsNot Nothing Then
    Dim vector As WIA.Vector = img.FileData
    pictureBox1.Image = Image.FromStream(New MemoryStream(CType(
        vector.get_BinaryData(), Byte)))
End If
Me.Text = "-"
timer1.Enabled = True
End Sub
```

Test

Nach dem Start müssen Sie sich einige Sekunden gedulden, dann sollte auch bei Ihnen ein Bild im Fenster erscheinen:



R194 Auf den Scanner zugreifen

Nachdem wir uns im vorhergehenden Rezept schon etwas mit der WIA beschäftigt haben, wollen wir uns im Rahmen dieses Beispiels noch einmal intensiver mit dem Zugriff auf Scanner befassen.

Zwei grundsätzliche Varianten sind bei Verwendung der WIA möglich:

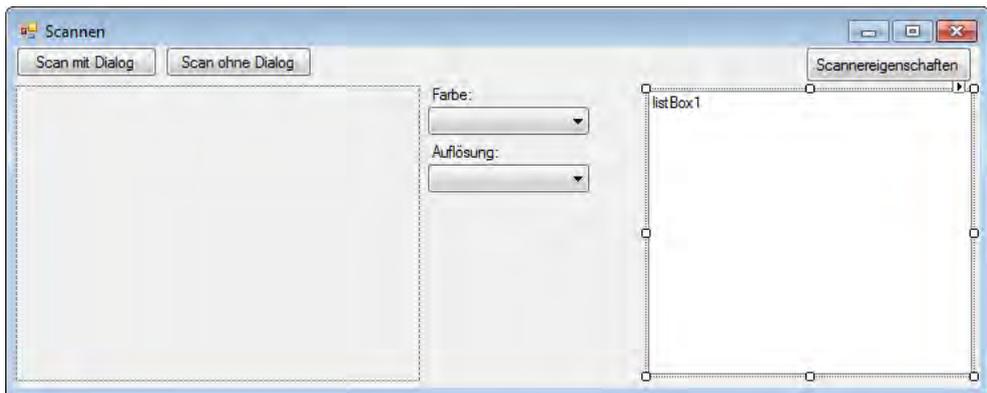
- Anzeige des Scanner-Assistenten und Übernahme des Bildes mit den dort getroffenen Einstellungen.
- Setzen der Scanner-Parameter mit eigenen Dialogen/Komponenten und direktem Abruf des Bildes.

Wie Sie es sich vermutlich denken können, ist die zweite Variante mit wesentlich mehr Programmieraufwand verbunden, müssen Sie sich doch recht intensiv mit den einzelnen Scanner-Parametern auseinandersetzen.

Im Weiteren wollen wir Ihnen beide Versionen vorstellen, zusätzlich beschäftigen wir uns noch mit dem Abruf der Standardeinstellungen.

Oberfläche

Wir beschränken uns auf eine *PictureBox* zur Anzeige, eine *ListBox* für die Scanner-Eigenschaften sowie zwei *ComboBox*en für die Auswahl der Farbeinstellungen und der DPI-Zahl. Last, but not least, sind auch noch drei Schaltflächen nötig:



Weisen Sie der Farb-*ComboBox* die Werte "Schwarz/Weiss", "Graustufe" und "Farbe" zu, die *ComboBox* für die DPI-Zahl konfigurieren Sie mit "75, 100, 150, 200, 300, 600".

HINWEIS: Binden Sie zusätzlich die "Microsoft Windows Image Acquisition Library v2.0" in Ihr VB-Projekt ein (*Projekt|Verweis hinzufügen|COM*).

Quellcode

```
Imports System.IO
Public Class Form1
```

Einige Konstanten für die Auswahl des Bildformats:

```
Const wiaFormatBMP As String = "{B96B3CAB-0728-11D3-9D7B-0000F81EF32E}"
Const wiaFormatPNG As String = "{B96B3CAF-0728-11D3-9D7B-0000F81EF32E}"
Const wiaFormatGIF As String = "{B96B3CB0-0728-11D3-9D7B-0000F81EF32E}"
Const wiaFormatJPEG As String = "{B96B3CAE-0728-11D3-9D7B-0000F81EF32E}"
Const wiaFormatTIFF As String = "{B96B3CB1-0728-11D3-9D7B-0000F81EF32E}"
...
```

Zunächst beschäftigen wir uns mit der Anzeige des Scannerdialogs:

```
Private Sub Button1_Click(sender As Object, e As EventArgs) _
    Handles Button1.Click
    Dim dlg As New WIA.CommonDialog()
    Dim img As WIA.ImageFile = Nothing
    img = dlg.ShowAcquireImage(WIA.WiaDeviceType.ScannerDeviceType,
        WIA.WiaImageIntent.GrayscaleIntent,
        WIA.WiaImageBias.MaximizeQuality, wiaFormatJPEG,
        False, False, False)
```

HINWEIS: Machen Sie sich keine Illusionen, es ist (fast) egal, was Sie dem Dialog übergeben, es wird immer die Standardansicht angezeigt.

Wurde ein Bild zurückgegeben, müssen wir dieses an unsere Anwendung weiterreichen:

```
If img IsNot Nothing Then
    Dim vector As WIA.Vector = img.FileData
    pictureBox1.Image = Image.FromStream(New MemoryStream(CType(vector.BinaryData(),
        Byte())))
End If
End Sub
```

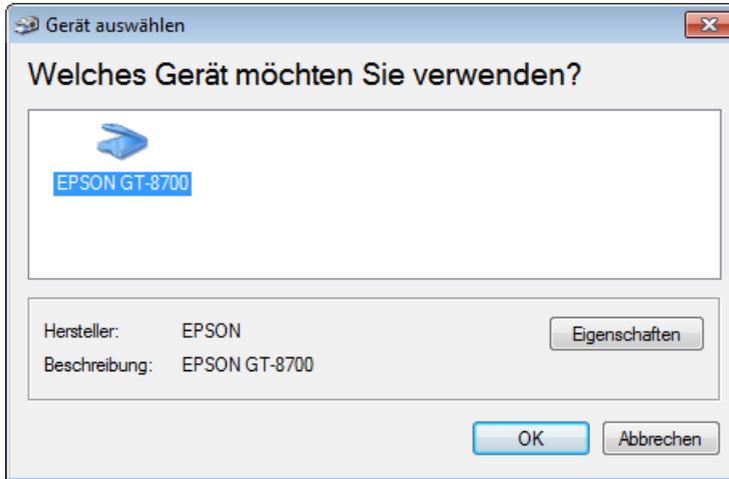
Die etwas flexiblere Variante für den Zugriff auf den Scanner nutzt die WIA-Dialoge lediglich zur Auswahl eines Gerätes. nachfolgend wird das Gerät konfiguriert und es wird das gewünschte Bild abgerufen. Doch der Reihe nach ...

```
Private Sub Button2_Click(sender As Object, e As EventArgs) _
    Handles Button2.Click
```

Ein Gerät abrufen:

```
Dim dlg As New WIA.CommonDialog()
Dim Scanner As WIA.Device = dlg.ShowSelectDevice(WIA.WiaDeviceType.ScannerDeviceType,
    False, False)
```

Der angezeigte Dialog:



Aus der zweiten *ComboBox* lesen wir die DPI-Zahl ab:

```
Dim dpi As Integer = Convert.ToInt16(comboBox2.SelectedItem)
```

Je nach Auswahl in der ersten *ComboBox* weisen wir die Farbauswahl zu:

```
Select Case comboBox1.SelectedIndex
    Case 0
        Scanner.Items(1).Properties("Current Intent").Value = 4
        ' 4 = SW, 2 = Graustufe, 1 = Color
    Case 1
        Scanner.Items(1).Properties("Current Intent").Value = 2
    Case Else
        Scanner.Items(1).Properties("Current Intent").Value = 1
End Select
```

Übergeben wird diese Information über die *Properties*-Collection des ersten Items. Gleiches trifft auch auf die anderen Scan-Optionen zu:

Dots per inch/Horizontal:

```
Scanner.Items(1).Properties("Horizontal Resolution").Value = dpi
```

Dots per inch/Vertikal:

```
Scanner.Items(1).Properties("Vertical Resolution").Value = dpi
```

Linke obere Ecke für den Scanbereich:

```
Scanner.Items(1).Properties("Horizontal Start Position").Value = 0 ' x links oben
Scanner.Items(1).Properties("Vertical Start Position").Value = 0 ' y links oben
```

Größe des Scanbereichs (Inch * dpi):

```
Scanner.Items(1).Properties("Horizontal Extent").Value = 8.5 * dpi
Scanner.Items(1).Properties("Vertical Extent").Value = 11 * dpi
```

Übertragen des Bildes in ein *ImageFile*-Objekt:

```
Dim img As WIA.ImageFile = Scanner.Items(1).Transfer()
```

War dies erfolgreich, müssen wir das *ImageFile*-Objekt "nur noch" in unserer *PictureBox* anzeigen:

```
If img IsNot Nothing Then
    Dim vector As WIA.Vector = img.FileData
    pictureBox1.Image = Image.FromStream(New MemoryStream(
        CType(vector.BinaryData(), Byte())))
End If
End Sub
```

Ganz nebenbei zeigen wir in der *ListBox* noch die möglichen Eigenschaften des Scanners an:

```
Private Sub Button4_Click(sender As Object, e As EventArgs) Handles Button4.Click
    Dim dlg As New WIA.CommonDialog()
    Dim Scanner As WIA.Device = dlg.ShowSelectDevice(WIA.WiaDeviceType.ScannerDeviceType,
        False, False)

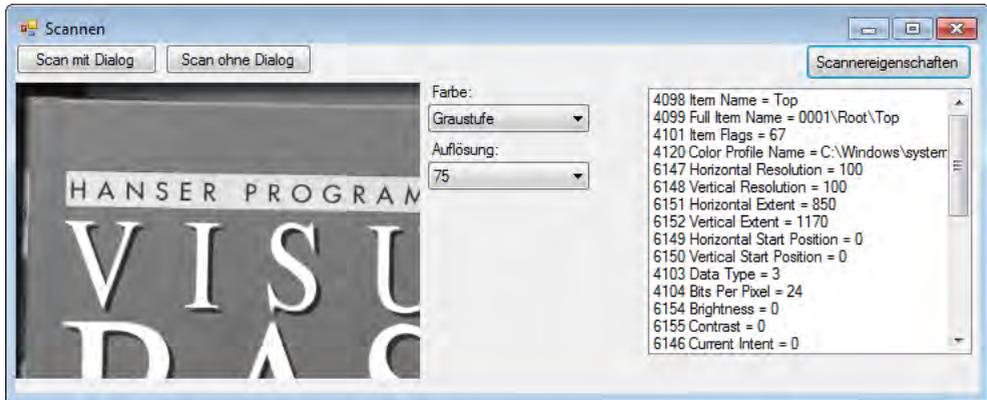
    For Each p As WIA.IProperty In Scanner.Items(1).Properties
        ListBox1.Items.Add(p.PropertyID & " " & p.Name & " = " & p.Value)
    Next p
End Sub
End Class
```

Das mögliche Ergebnis (geräteabhängig):

```
4101 Item Flags = 67
4120 Color Profile Name = C:\Windows\system
6147 Horizontal Resolution = 100
6148 Vertical Resolution = 100
6151 Horizontal Extent = 850
6152 Vertical Extent = 1170
6149 Horizontal Start Position = 0
6150 Vertical Start Position = 0
4103 Data Type = 3
4104 Bits Per Pixel = 24
6154 Brightness = 0
6155 Contrast = 0
6146 Current Intent = 0
4112 Pixels Per Line = 850
4114 Number of Lines = 1170
4105 Preferred Format = {B96B3CAA-0728-11C
```

Test

Nach dem Start können Sie beide Scan-Varianten ausprobieren. Experimentieren Sie ruhig auch einmal mit den Properties des *Item*-Objekts um andere Auflösungen oder Scanbereiche zu bestimmen.



Bemerkung

Die Parameter für *ShowAcquireImage*:

SYNTAX: ImageFile ShowAcquireImage([DeviceType], [Intent], [Bias], [FormatID],
[AlwaysSelectDevice], [UseCommonUI], [CancelError])

Die Bedeutung:

Parameter	Beschreibung
<i>DeviceType</i>	Werte der <i>WiaDeviceType</i> -Enumeration, Default = <i>UnspecifiedDeviceType</i> , für Scanner nutzen Sie bitte <i>ScannerDeviceType</i> .
<i>Intent</i>	Werte der <i>WiaImageIntent</i> -Enumeration (Wert egal)
<i>Bias</i>	Werte der <i>WiaImageBias</i> -Enumeration (Wert egal)
<i>FormatID</i>	Das zurückgegebene Bildformat: <i>wiaFormatBMP</i> = "{B96B3CAB-0728-11D3-9D7B-0000F81EF32E}" <i>wiaFormatPNG</i> = "{B96B3CAF-0728-11D3-9D7B-0000F81EF32E}" <i>wiaFormatGIF</i> = "{B96B3CB0-0728-11D3-9D7B-0000F81EF32E}" <i>wiaFormatJPEG</i> = "{B96B3CAE-0728-11D3-9D7B-0000F81EF32E}" <i>wiaFormatTIFF</i> = "{B96B3CB1-0728-11D3-9D7B-0000F81EF32E}"
<i>AlwaysSelectDevice</i>	Soll das Gerät jedes Mal ausgewählt werden?
<i>UseCommonUI</i>	Soll der Dialog angezeigt werden (das ist immer der Fall)?
<i>CancelError</i>	Soll ein Fehler ausgelöst werden, wenn der Nutzer Abbruch wählt?

R195 Auf die serielle Schnittstelle zugreifen

Zahlreiche Mess- und Elektrogeräte¹ sind auch heute noch mit der klassischen seriellen Schnittstelle ausgestattet. In Verbindung mit dem zu Visual Studio mitgelieferten *SerialPort*-Steuerelement können Sie beliebige Programme schreiben, mit denen sich diese Geräte steuern lassen.



Um eine sinnvolle praktische Anwendung des *SerialPort*-Controls mit geringstem Aufwand zu demonstrieren, verwenden wir hier als Peripheriegerät ein Digitalvoltmeter (z.B. DT9602R) mit serieller Schnittstelle². Es dürfte aber auch jedes andere DVM mit serieller Schnittstelle geeignet sein. Die Abbildung zeigt das DVM beim Messen einer Batteriespannung, oben sieht man das Kabel, welches zum seriellen COM-Port des PC führt.

Oberfläche

Auf das Startformular *Form1* setzen Sie ein mittels *BorderStyle*-, *Font*- und *Color*-Eigenschaften stattlich herausgeputztes *Label*, welches als Anzeigeelement dienen soll. Weiterhin benötigen Sie als Herzstück natürlich das *SerialPort*-Control sowie einen simplen *Button* zum Beenden.

Beim *SerialPort*-Control können Sie es meist bei den im Eigenschaftenfenster vorgegebenen Standardeigenschaften belassen (*DataBits* = 8; *Parity* = *None*; *StopBits* = *One*; ...) belassen. Lediglich die Eigenschaften *PortName* und *BaudRate* müssen Sie individuell anpassen. Letzteren Wert können Sie in der Regel dem Datenblatt des DVM entnehmen (z.B. Baudrate 2400 für das verwendete DT9602R). Und noch etwas sollten Sie nicht vergessen:

¹ Sogar der Hometrainer des Autors verfügt über einen seriellen RS232-Port.

² Preisgünstige Digitalvoltmeter mit RS232-Schnittstelle gibt es schon für ca. 30 Euro bei (fast) jedem Elektronikversand.

HINWEIS: In unserem Fall ist die *DtrEnable*-Eigenschaft des *SerialPort*-Controls auf *True* zu setzen!

Quellcode

```
Imports System.IO.Ports
```

```
Public Class Form1
```

Beim Laden des Formulars wird der Port geöffnet:

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    SerialPort1.Open()
End Sub
```

Messdaten sind eingetroffen:

```
Private Sub SerialPort1_DataReceived(sender As Object, e As SerialDataReceivedEventArgs) _
    Handles SerialPort1.DataReceived
    Try
        Dim count As Integer = SerialPort1.BytesToRead ' Anzahl Bytes im Empfangspuffer
```

Um den Inhalt des Empfangspuffers komplett als Zeichenkette auszulesen, könnte man die *ReadExisting*-Methode verwenden. Für die Übertragung von Zahlenwerten (Messdaten etc.) ist es aber zweckdienlicher, wenn wir von der Text- auf die Byte-Ebene hinabsteigen. Die im Folgenden vorgestellte Variante benutzt die *Read*-Methode der *SerialPort*-Komponente, um eine bestimmte Anzahl von Bytes aus dem Empfangspuffer in ein Byte-Array zur weiteren Verarbeitung einzulesen:

```
Dim ba() As Byte
ReDim ba(count - 1)
SerialPort1.Read(ba, 0, count)
```

Es folgt das Aufsplitten des übergebenen Byte-Arrays in zwei Strings (Messwert und Messbereich). In unserem Beispiel entsteht ein String aus acht ASCII-Zeichen. Dabei bilden die ersten fünf Zeichen den Messwert:

```
Dim data1 As String = System.Text.Encoding.Default.GetString(ba, 0, 5)
```

Danach folgt ein Leerzeichen. Die letzten beiden Zeichen verweisen auf den Messbereich:

```
Dim data2 As String = System.Text.Encoding.Default.GetString(ba, 6, 2)
```

Zwecks Anzeige muss auf den UI-Thread umgeschaltet werden:

```
Label1.Invoke(displayDataPtr, data1, data2)
```

Warum so umständlich? Die Komponenten der Benutzerschnittstelle (UI = User Interface) können nur von dem Thread aus verändert werden, von dem sie erzeugt wurden. Der Empfang der seriellen Daten läuft aber in einem anderen Thread ab. Als Lösung des Problems bietet sich ein

Methodenzeiger (Delegate) an, der über die *Invoke*-Methode der entsprechenden Komponente (hier *Label1*) aufgerufen wird¹:

```
Private Delegate Sub displayDataD1g(dat1 As String, dat2 As String)
Private displayDataPtr As displayDataD1g = AddressOf displayData
```

Die Anzeigeroutine, auf welche das *Delegate*-Objekt *displayDataPtr* verweist:

```
Private Sub displayData(d1 As String, d2 As String)
```

Messwert in Gleitkommazahl parsen:

```
Dim f As Single = Single.Parse(d1)
```

Messbereichsabhängige Umrechnung und Anzeige:

```
Select Case d2
    Case "11" : Label1.Text = f / 1000 & " V"
    Case "21" : Label1.Text = f / 100 & " V"
    Case "41" : Label1.Text = f / 10 & " mV"
    Case Else : Label1.Text = String.Empty
End Select
End Sub
```

Beim Beenden des Programms sollte auch der Port wieder geschlossen werden:

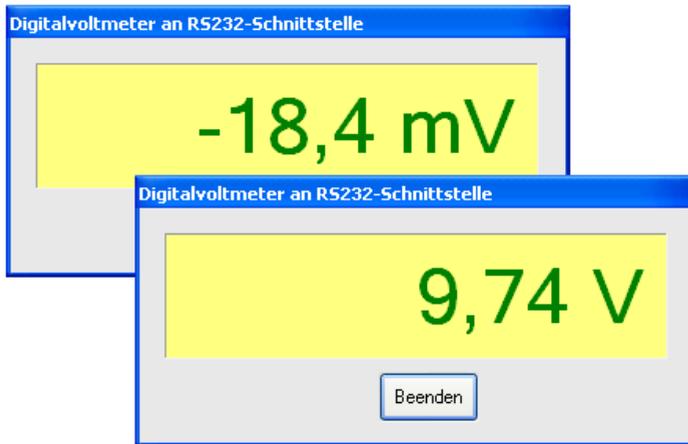
```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    SerialPort1.Close()
    Me.Close()
End Sub
End Class
```

Test

Verbinden Sie Ihr DVM über das im Zubehör enthaltene Kabel mit der seriellen COM1-Schnittstelle des PC (andernfalls *PortName*-Eigenschaft im Quellcode anpassen). Stellen Sie den Messbereichsschalter des DVM auf Gleichspannung (VOLT DC) ein und aktivieren Sie die Übertragung per RS232 (siehe Bedienungsanleitung des DVM).

In Abhängigkeit von der Wandlungsrate des DVM (die des DT9602R ist hier mit ca. 1/sek nicht sonderlich berauschend) sollte das Programm nun kontinuierlich die aktuellen Messwerte anzeigen.

¹ Mehr zur asynchronen Programmierung und zur *SerialPort*-Komponente erfahren Sie in unserem Buch [Visual Basic 2012 – Grundlagen und Profiwissen].



Bemerkungen

- Gefahren für Mensch und Computer beim Experimentieren sind so gut wie ausgeschlossen, da die RS232-Schnittstelle bei Messgeräten meist über Optokoppler herausgeführt wird.
- Falls keine Unterlagen zum DVM verfügbar sind: Um die Codierung der Messwerte und Messbereiche experimentell herauszufinden, kann das in unserem Buch [Visual Basic 2012 Grundlagen und Profiwissen] beschriebene Terminalprogramm benutzt werden (nicht vergessen: Häkchen bei DTR setzen!). Durch einige Versuche mit unterschiedlichen Spannungen lässt sich meist leicht feststellen, wie die einzelnen ASCII-Zeichen zu interpretieren sind.
- Für eine komplette Steuerung (Modelleisenbahn¹) eignet sich z.B. das preisgünstige PC-Messmodul M 232, es verfügt über 6 Analogeingänge (10-Bit-Auflösung) sowie 8 digitale Ein-/Ausgänge. Davon kann ein Eingang als Zählereingang für ein 16-Bit-Register benutzt werden.
- Neuere PCs verfügen zugunsten zahlreicher USB-Ports meist nur noch über einen einzigen seriellen COM-Port. Hier lassen sich mit Hilfe handelsüblicher *USB zu SUB-D9-Adapter* weitere serielle Ports realisieren.

R196 OpenOffice.org Writer per OLE steuern

Angesichts der reichlich vorhandenen Lösungen zur MS Word-Automatisierung entsteht für manchen Programmierer sicher auch die Frage, wie sich *OpenOffice.org Writer*, der Freeware-Pendant von Word, automatisieren lässt. Das vorliegende Beispiel soll einen ersten Ansatz aufzeigen.

Leider ist die Programmierung von OpenOffice-Anwendungen per C#/Visual Basic nicht gerade trivial, eine fast schon chaotisch wirkende Programmierschnittstelle macht es dem eingefleischten MS-Programmierer nicht gerade leicht.

¹ Das Kind im Mann freut sich!

Wir wollen Ihnen etwas Arbeit abnehmen und kapseln deshalb einige der wichtigsten Grundfunktionen in einer eigenen Klasse *OOWriter*.¹ Mit dieser lassen sich dann die Standardaufgaben relativ leicht lösen.

HINWEIS: Um die Wiederverwendbarkeit zu verbessern, haben wir die Funktionalität in eine eigene Library ausgelagert, Sie können die enthaltene Klasse *cOOWriter* aber auch genauso gut in Ihr Projekt kopieren².

VB-Library "OOWLib"

Library erstellen

Erstellen Sie zunächst ein Windows Forms-Projekt und fügen Sie in die übergeordnete Solution eine Klassenbibliothek mit dem Namen *OOWLib* ein (Kontextmenü *Hinzufügen neues Projekt*).

Löschen Sie den schon vorhandenen Klassenrumpf und erstellen Sie eine neue Klasse *cOOWriter*, auf deren Details wir im Folgenden eingehen wollen.

Quelltext (Klasse *cOOWriter*)

Unsere neue Klasse:

```
Public Class cOOWriter
```

Einige interne Objekte:

```
Private oStarOffice As Object  
Private oDesk As Object  
Private oDoc As Object  
Private oVC As Object
```

Wir erstellen ein neues, leeres Dokument:

```
Public Sub NewDocument()  
    Dim args(-1) As Object  
    oDesk = oStarOffice.CreateInstance("com.sun.star.frame.Desktop")
```

Das aktuelle Dokument:

```
oDoc = oDesk.LoadComponentFromURL("private:factory/swriter", "_blank", 0, args)
```

¹ Sollte ein Leser den Elan aufbringen, die Klasse wesentlich zu erweitern, sind die Autoren für jede Art von Feedback dankbar.

² Ganz nebenbei profitiert von dieser Library auch unsere C#-Leserschaft, da sich VB beim Zugriff auf COM-Objekte nach wie vor als überlegen erweist.

Der sichtbare Cursor¹:

```
oVC = oDoc.CurrentController.ViewCursor()
End Sub
```

Alternativ greifen wir auf eine vorhandene Datei zu:

```
Public Sub OpenDocument(doc As String, Optional ro As Boolean = False,
    Optional Passwort As String = "")
    Dim args(2) As Object
    args(0) = MakePropertyValue("ReadOnly", ro)
    args(1) = MakePropertyValue("Password", Passwort)
    args(2) = MakePropertyValue("Hidden", False)
    oDesk = oStarOffice.CreateInstance("com.sun.star.frame.Desktop")
    oDoc = oDesk.LoadComponentFromURL(ConvertToUrl(doc), "_blank", 0, args)
    oVC = oDoc.CurrentController.ViewCursor()
End Sub
```

HINWEIS: Sie müssen eine der beiden o.g. Methoden ausführen, bevor Sie das Dokument bearbeiten können.

Ausgabe von Text:

```
Public Sub InsertString(str As String)
    oDoc.Text.InsertString(oVC, str, False)
End Sub
```

Ein Zeilenumbruch:

```
Public Sub NewLine()
    oDoc.Text.InsertControlCharacter(oVC, 0, False)
End Sub
```

Ein neuer Absatz:

```
Public Sub NewParagraph()
    oDoc.Text.InsertControlCharacter(oVC, 5, False)
End Sub
```

Ein Seitenumbruch:

```
Public Sub PageBreak()
    oDoc.Text.InsertControlCharacter(oVC, 0, False)
    oVC.BreakType = 4
End Sub
```

Eine Dokumentvorlage nutzen:

```
Public Sub SetStyle(StyleName As String)
    oVC.ParaStyleName = StyleName
End Sub
```

¹ Writer kennt auch noch Textcursor, die unabhängig vom sichtbaren Cursor agieren können.

Fettschrift:

```
Public Property Bold() As Boolean
    Get
        Return (oVC.CharWeight = 150)
    End Get
    Set(value As Boolean)
        If value Then
            oVC.CharWeight = 150
        Else
            oVC.CharWeight = 100
        End If
    End Set
End Property
```

Eine Schriftart auswählen:

```
Public Sub SetFont(Fontname As String, Optional Size As Long = 12)
    oVC.CharFontname = Fontname
    If Size > 0 Then oVC.CharHeight = Size
End Sub
```

An den Anfang des Dokuments springen:

```
Public Sub GotoStart()
    oVC.JumpToFirstPage()
End Sub
```

An das Ende des Dokuments springen:

```
Public Sub GotoEnd()
    oVC.JumpToLastPage()
    oVC.jumpToEndOfPage()
End Sub
```

Datei als PDF sichern:

```
Public Sub SaveAsPDF(filename As String)
    Dim args(1) As Object
    args(0) = MakePropertyValue("FilterName", "writer_pdf_Export")
    oDoc.storeToURL(ConvertToUrl(filename), args)
End Sub
```

Dokument schließen:

```
Public Sub CloseDocument()
    oDoc.Close(True)
End Sub
```

Die folgenden internen Funktionen stammen aus dem Internet¹ und erleichtern den Zugriff auf die OpenOffice-Objekte:

¹ http://www.kalitech.fr/clients/doc/VB_APIOOo_en.html

Eine Objekt-Eigenschaft erzeugen:

```
Private Function MakePropertyValue(cName, uValue) As Object
    Dim oPropertyValue As Object
    oPropertyValue = oStarOffice.Bridge_GetStruct("com.sun.star.beans.PropertyValue")
    oPropertyValue.Name = cName
    oPropertyValue.value = uValue
    MakePropertyValue = oPropertyValue
End Function
```

Pfadangaben in Urls umwandeln:

```
Private Function ConvertToUrl(strFile As String) As String
    strFile = Replace(strFile, "\", "/")
    strFile = Replace(strFile, ":", "|")
    strFile = Replace(strFile, " ", "%20")
    strFile = "file:/// " + strFile
    ConvertToUrl = strFile
End Function
```

Ein spezielles Service-Objekt erzeugen:

```
Private Function CreateUnoService(strServiceName) As Object
    Dim oServiceManager As Object
    oServiceManager = CreateObject("com.sun.star.ServiceManager")
    CreateUnoService = oServiceManager.createInstance(strServiceName)
End Function
```

Im Konstruktor erstellen wir das zentrale *ServiceManager*-Objekt:

```
Public Sub New()
    oStarOffice = CreateObject("com.sun.star.ServiceManager")
End Sub
End Class
```

Kompilieren

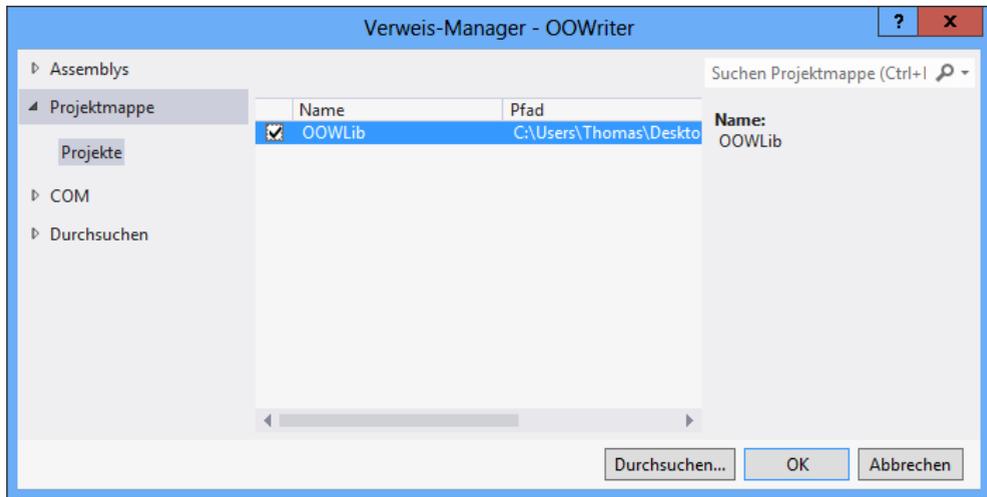
HINWEIS: Kompilieren Sie abschließend die Klassenbibliothek um die gewünschte Assembly zu erstellen.

Anwendungsprogramm

Damit endet auch schon unser kurzer Ausflug in die Library-Programmierung und wir kehren wieder zu unserer vertrauten Programmiersprache zurück.

Einbinden der Assembly

Wählen Sie in der Solution das Windows Forms-Projekt und fügen Sie eine Referenz auf die oben erstellte Klassenbibliothek hinzu (*Verweis hinzufügen*|*Projekte*).



Nachfolgend kommen wir schon in den Genuss der vollen Funktionalität unserer Assembly, wir müssen den Namespace nur noch importieren.

Oberfläche

Die Oberfläche beschränkt sich auf einen *Button* und einen *FileSaveDialog*.

Quelltext

Um das Beispiel nicht auf ein simples "Hello World" zu beschränken, wollen wir den Inhalt einer XML-Datei formatiert ausgeben.

HINWEIS: Die dazu nötige XML-Datei finden Sie in den Beispieldaten zum Buch.

Import der beiden benötigten Namespaces:

```
Imports OOWLib
Imports System.Xml.Linq
Public Class Form1
```

Eine Instanz für die Elemente der XML-Datei:

```
Private ArtikelListe As XElement
```

```
Public Sub New()
    InitializeComponent()
```

Hier laden wir die XML-Beispieldaten:

```
ArtikelListe = XElement.Load("Artikel.xml")
End Sub
```

Mit dem Klick auf die Schaltfläche beginnen die hektischen Aktivitäten:

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
```

Eine Instanz des Writers erstellen¹:

```
Dim oow As New cOOWriter()
```

Ein neues Dokument anlegen:

```
oow.NewDocument()
```

Formatvorlage *Überschrift1* auswählen:

```
oow.SetStyle("Überschrift 1")
```

Textausgabe:

```
oow.InsertString("Lagerdaten")
```

Neuer Absatz und Formatvorlage *Standard*:

```
oow.NewParagraph()  
oow.SetStyle("Standard")
```

Zeilenvorschub:

```
oow.NewLine()
```

Hier folgt die Schleife für alle *Artikel*-Elemente in den XML-Daten:

```
For Each Artikel As XElement In ArtikelListe.Elements()
```

Fettschrift und Textausgabe:

```
oow.Bold = True  
oow.InsertString("ArtikeInr : ")  
oow.Bold = False  
oow.InsertString(Artikel.Element("ArtikeInr").Value)  
oow.NewLine()  
oow.Bold = True  
oow.InsertString("ArtikeIname : ")  
oow.Bold = False  
oow.InsertString(Artikel.Element("ArtikeIname").Value)  
oow.NewLine()  
oow.Bold = True  
oow.InsertString("Lagerbestand : ")  
oow.Bold = False  
oow.InsertString(Artikel.Element("Lagerbestand").Value)  
oow.NewLine()  
oow.NewLine()  
Next Artikel
```

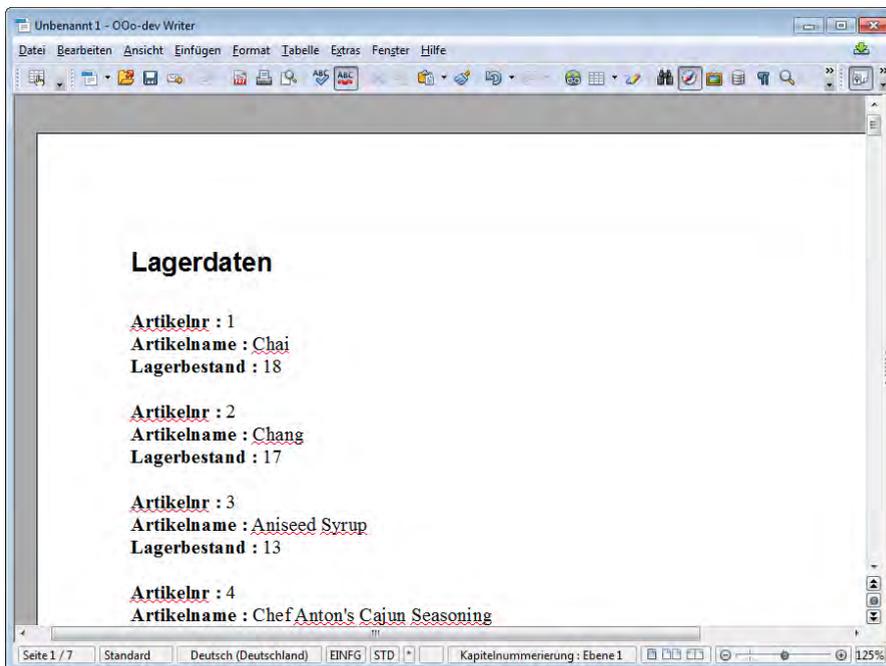
¹ Im Gegensatz zu Word/Excel etc. wird der Writer standardmäßig eingelenket, Sie müssen diesen also nicht erst sichtbar machen.

Last, but not least wollen wir das erzeugte Writer-Dokument als PDF-Datei speichern:

```
If SaveFileDialog1.ShowDialog() = System.Windows.Forms.DialogResult.OK Then
    oow.SaveAsPDF(SaveFileDialog1.FileName)
End If
oow = Nothing
End Sub
End Class
```

Test

Nach dem Klick auf die Schaltfläche sollten Sie folgende Ausgabe erhalten (vorausgesetzt, Sie haben *OpenOffice.org* installiert):



Auch der Export ins PDF-Format dürfte problemlos realisiert werden, Sie können sich mit dem Acrobat Reader vom Erfolg überzeugen.

Bemerkung

Auch wenn Sie mit obigem Beispiel recht schnell zu brauchbaren Ergebnissen kommen, viele OpenOffice-Funktionen sind nur umständlich erreichbar (z.B. Zwischenablage).

Hilfe finden Sie in diesem Fall unter:

LINK: <http://forum.openoffice.org/en/forum/>

R197 Mit OLE-Automation auf MS Access zugreifen

Da VBA nach wie vor auch für Microsoft Access als Programmiersprache eingesetzt wird, muss man leider auch in Kauf nehmen, dass die VBA-Fähigkeiten im Vergleich zu den Möglichkeiten der modernen .NET-Programmiersprachen, wie Visual Basic oder Visual C#, ziemlich eingeschränkt sind. Insbesondere trifft dies auf verteilte Umgebungen zu, d.h., Intranet- und Internet-Applikationen lassen sich mit VBA – wenn überhaupt – nur sehr umständlich entwickeln.

Um von VB aus auf Access Datenbanken zugreifen zu können, gibt es hauptsächlich zwei Alternativen:

- Automation
- und ADO.NET.

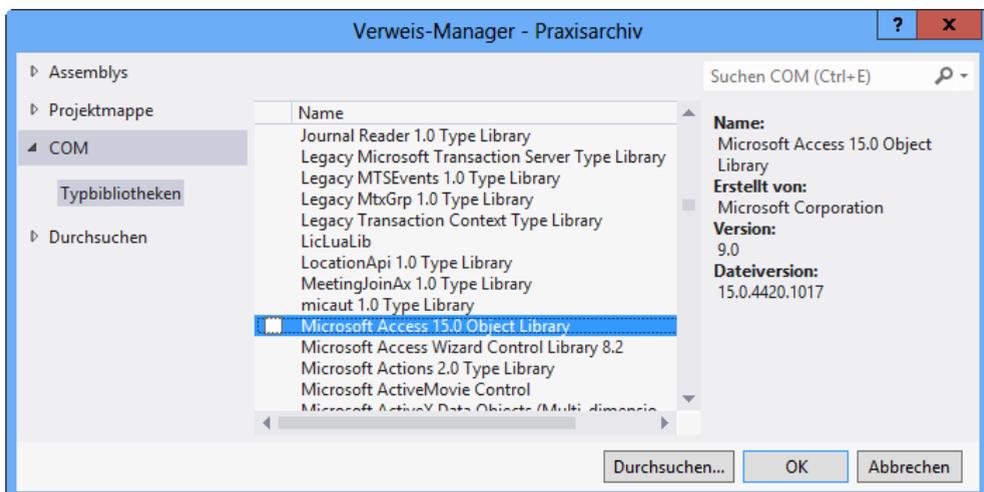
Automation erlaubt bekanntlich die quasi "Fernsteuerung" einer Anwendung von einer anderen Anwendung aus. Sie sollten Automation dann verwenden, wenn Sie an Access-spezifischen Features interessiert sind, wie z.B. die Druck- oder Vorschau-Funktionen für Access-Reports, das Anzeigen und Manipulieren eines Access-Formulars oder der Aufruf eines Makros.

Im vorliegenden Beispiel werden wir zeigen, wie Sie auf einen Access-Report per OLE-Automation zugreifen können. Um es nicht ganz so einfach zu machen, rufen wird den Report allerdings über ein Access-Formular auf, in welches Bereichsgrenzen einzugeben sind.

Oberfläche

Erstellen Sie eine neue Windows Forms-Anwendung. Auf dem Hauptformular platzieren Sie eine Schaltfläche.

Über das Menü *Projekt|Verweis hinzufügen ...* öffnen Sie den entsprechenden Dialog, wo Sie auf der COM-Seite einen Verweis auf die *Microsoft Access xx.x Object Library* einrichten:



Quelltext

Um unliebsamen Problemen mit dem Namespace *Microsoft.Office.Interop.Access* aus dem Weg zu gehen (Überschneidung bei Bezeichnungen) vergeben wir einen Alias-Namen:

```
Imports ACC = Microsoft.Office.Interop.Access
Imports System.Runtime.InteropServices
...
Public Class Form1
```

Die Schaltfläche *Access aufrufen*:

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
```

Unsere Hauptroutine hat die Aufgabe, die Datenbank *Test.mdb* aufzurufen und dort das Formular *Bericht aufrufen* anzuzeigen. Demonstriert wird außerdem die Manipulation von Steuerelementen:

```
Dim oAccess As ACC.Application = Nothing
Dim oForm As ACC.Form = Nothing
Dim dbPath As String = "" ' Datenbankpfad zu Test.mdb
```

Eine neue Access-Instanz für die Automation starten:

```
oAccess = New ACC.Application()
```

Dafür sorgen, dass Access sichtbar ist:

```
oAccess.Visible = True
```

Datenbankpfad ermitteln (die Datenbank befindet sich hier der Einfachheit wegen im Anwendungsverzeichnis!):

```
dbPath = Application.StartupPath & "\Test.mdb"
```

Datenbank öffnen (im Shared Mode):

```
oAccess.OpenCurrentDatabase(filepath:=dbPath, Exclusive:=False)
```

Eventuell geöffnete Formulare schließen:

```
For Each oFormWithinLoop As ACC.Form In oAccess.Forms
    oForm = oFormWithinLoop
    oAccess.DoCmd.Close(ObjectType:=ACC.AcObjectType.acForm,
                        ObjectName:=oFormWithinLoop.Name,
                        Save:=ACC.AcCloseSave.acSaveNo)
Next oFormWithinLoop
If oForm IsNot Nothing Then Marshal.ReleaseComObject(oForm)
oForm = Nothing
```

Das Formular im Datenbankfenster auswählen und den Fokus erteilen:

```
oAccess.DoCmd.SelectObject(ObjectType:=ACC.AcObjectType.acForm,
                           ObjectName:="Bericht aufrufen", InDatabaseWindow:=True)
```

Das Formular anzeigen und seine Beschriftung editieren:

```
oAccess.DoCmd.OpenForm(FormName:="Bericht aufrufen", View:=ACC.AcFormView.acNormal)
oForm = oAccess.Forms("Bericht aufrufen")
oForm.Caption = "Access-Automation mit .NET"
```

Die Controls-Auflistung verwenden, um die Beschriftung der Schaltfläche *Befehl1* zu editieren:

```
Dim oCtl As Object = oForm.Controls("Befehl1")
oCtl.Caption = "Report aufrufen"
Marshal.ReleaseComObject(oCtl)
oCtl = Nothing
oForm.SetFocus()
```

Das Objekt freigeben:

```
Marshal.ReleaseComObject(oForm)
oForm = Nothing
```

Das *Application*-Objekt freigeben und erlauben, dass Access durch den Anwender geschlossen wird:

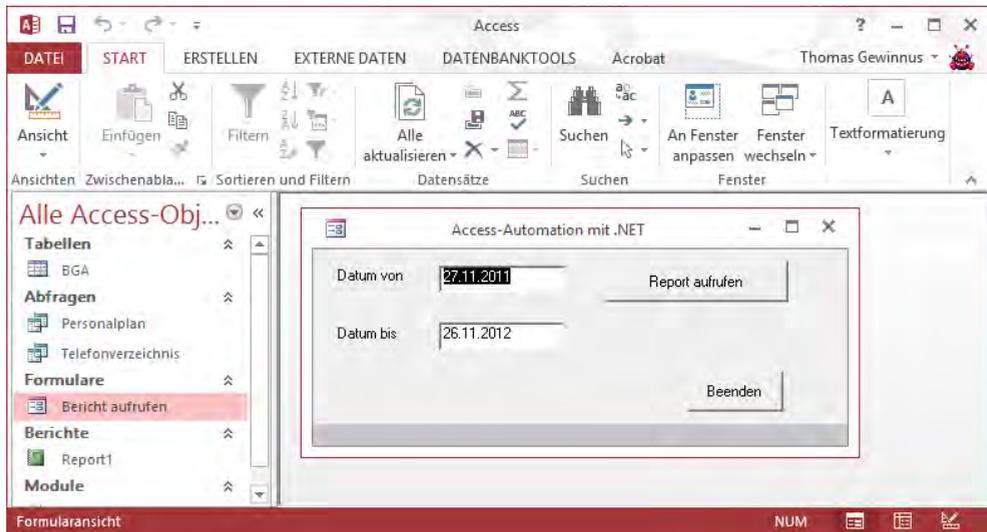
```
If Not oAccess.UserControl Then oAccess.UserControl = True
Marshal.ReleaseComObject(oAccess)
oAccess = Nothing
End Sub
```

```
End Class
```

Test

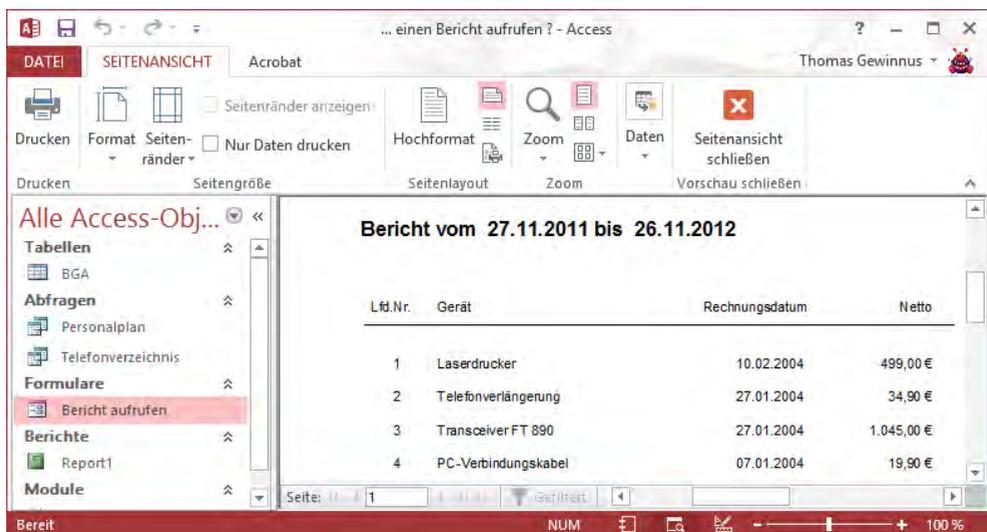
Die Datenbank *Test.mdb* (siehe Buchbeispiel) bietet keine Besonderheiten, sie besteht lediglich aus der Tabelle *BGA* und einem Formular, von welchem aus über eine Schaltfläche der Bericht *Report1* aufgerufen wird.

Nach dem Kompilieren des VB-Projekts und Betätigen der Schaltfläche *Access aufrufen* wird wie von Geisterhand Microsoft Access gestartet und es erscheint das gewünschte Formular, von welchem aus Sie dann die in Access implementierten Druck- und Vorschaufunktionen aufrufen.



Auch an den geänderten Beschriftungen von Formular und Schaltfläche sehen Sie, dass die Automation funktioniert hat.

Nach Betätigen der Schaltfläche *Report aufrufen* erscheint zunächst die Vorschau, von welcher aus Sie dann den Druck starten können:



Bemerkungen

- Natürlich wäre es auch möglich, den Druckvorgang bzw. die Druckvorschau direkt aus VB heraus zu starten. Wir haben uns jedoch für das "Dazwischenschalten" eines Access-Formulars entschieden, um den Quellcode überschaubarer zu halten.
- Auf die gleiche Weise wie hier für ein Access-Formular demonstriert, lassen sich auch alle anderen Access-Objekte (Tabellen, Berichte, Abfragen, Module) von einer beliebigen .NET-Anwendung aus über Automation quasi "fernsteuern".

R198 Ein Managed Add-In programmieren und einbinden

Sie möchten mit MS Access einige Funktionen realisieren, die mit dem boardeigenen VBA nicht realisierbar sind. Spontan fallen uns da vor allem Webdienste, Remoting, Windows Forms, WPF etc. ein.

Einen direkten Weg für den Zugriff auf obige Funktionen gibt es nicht, aber eine kleine und recht effiziente Hintertür haben die Microsoft-Entwickler offen gelassen. Die Rede ist von der Möglichkeit, Managed Add-Ins in Microsoft Access zu integrieren, die Sie zum Beispiel in C# oder VB programmieren können.

HINWEIS: Diese Add-Ins können Sie nur mit Visual Studio 2010 erstellen. In Visual Studio 2012 ist die entsprechende Unterstützung **nicht mehr vorhanden**.

Visual Studio 2010 bietet zu diesem Zweck den Projekttyp *Gemeinsames Add-In* an, Sie müssen "nur noch" eine geeignete Schnittstelle zwischen der Access-Anwendung und dem Add-In schaffen. Wer jetzt befürchtet, sich mit Unmengen an Deklarationen etc. herumschlagen zu müssen, der sei beruhigt, diese Arbeit wird von einem Assistenten weitestgehend erledigt, es bleibt wirklich nur die reine Schnittstelle übrig.

Doch welche Form der Interaktion zwischen Access-Anwendung und Add-In ist überhaupt möglich? Eigentlich gibt es da kaum Einschränkungen, nach der Übergabe einer Referenz können Sie zum Beispiel auf

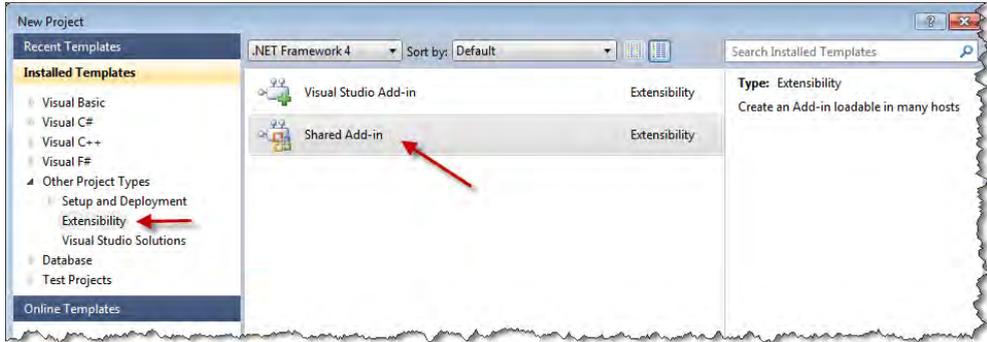
- die Anwendung (per *Application*-Objekt),
- geöffnete Formulare und Berichte (per *Me*-Referenz),
- Steuerelemente (Eigenschaften, Methoden, Ereignisse)

zugreifen.

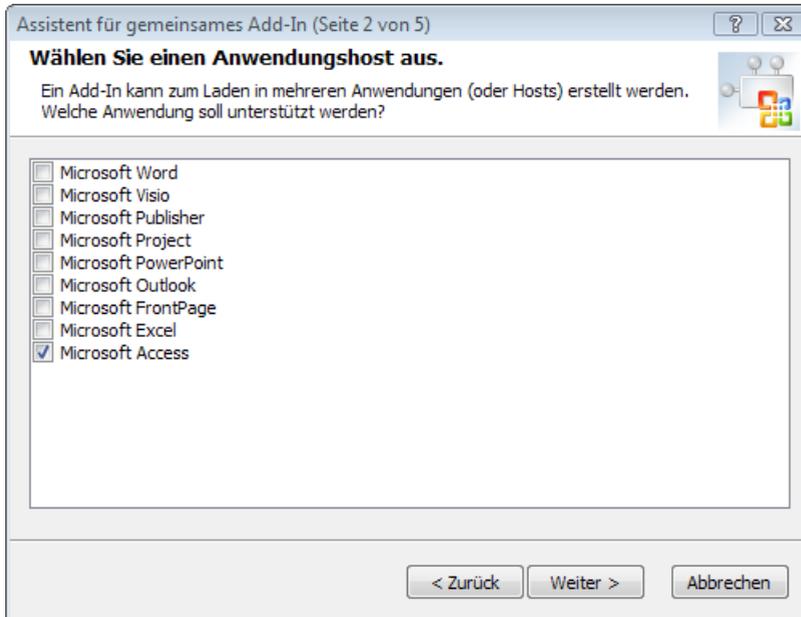
Ein kleines Beispielprogramm soll das Zusammenspiel demonstrieren. Dazu werden wir per Add-In zwei Methoden bereitstellen, eine Access-Schaltfläche konfigurieren und ein .NET-Formular zur Konfiguration eines Access-Kombinationsfeldes anzeigen. Doch der Reihe nach.

Entwurf des Add-Ins

Starten Sie Visual Studio und wählen Sie *Datei|Neu|Projekt|Andere Projekttypen|Erweiterungen|Gemeinsames Add-In*:



Nachfolgend erscheint ein Assistent, wählen Sie hier als Programmiersprache "Visual Basic". Als Anwendungshost (Schritt 2) genügt uns Microsoft Access:



Dritter Schritt ist das Festlegen von Name und Beschreibung des Add-Ins:

Assistent für gemeinsames Add-In (Seite 3 von 5)

Geben Sie einen Namen und eine Beschreibung ein.
Für ein Add-In sind ein Name und eine Beschreibung erforderlich, damit es sich den Benutzern besser erschließt. Geben Sie unten Entsprechendes ein.

Geben Sie einen Namen für das Add-In ein:

DOKO-Test-Addin

Geben Sie eine Beschreibung für das Add-In ein:

Wir spielen etwas

< Zurück Weiter > Abbrechen

Die beiden folgenden Optionen lassen wir für die Entwurfszeit markiert, auf diese Weise wird das Add-In automatisch beim Start von Access geladen und steht jederzeit zur Verfügung.

Assistent für gemeinsames Add-In (Seite 4 von 5)

Wählen Sie die Add-In-Optionen aus.
Add-In-Optionen

Das Add-In laden, wenn die Hostanwendung geladen wird.

Das Add-In soll für alle Benutzer des Computers verfügbar sein, auf dem es installiert ist, nicht nur für den Benutzer, der das Add-In installiert.

< Zurück Weiter > Abbrechen

Nach einigem Festplattenklappern sollte ein neues Projekt in Visual Studio erscheinen, das aus dem eigentlichen Add-In und einem Setup-Projekt besteht (mehr dazu später).

Verweise einrichten

So wie das Projekt derzeit konfiguriert ist, ist die Programmierung sicher nicht komfortabel, fehlen doch alle Datentypen/Objekte und Konstanten, die wir für die Access-Programmierung benötigen. Fügen Sie also noch einen Verweis auf die *Microsoft Access 12.0 Object Library* hinzu.

Klicken Sie dazu im Projektmappen-Explorer auf den Add-In-Knoten (Kontextmenü *Verweis hinzufügen*) und wählen Sie im folgenden Dialogfenster in der Rubrik "COM" die oben genannte Library aus. Nachfolgend sollten die erforderlichen Verweise in Ihrem Projekt auftauchen.

Einen weiteren Verweis müssen Sie in der Rubrik ".NET" auswählen, es handelt sich um die Assembly *System.Windows.Forms*.

Quellcode Add-In

Damit können wir uns dem Quellcode des Add-Ins zuwenden. Das im Folgenden abgedruckte Listing ist um die fett hervorgehobenen Zeilen ergänzt bzw. erweitert worden:

```
Imports Extensibility
Imports System.Runtime.InteropServices

Die Unterstützung für den Access-Namespaze:
Imports Access = Microsoft.Office.Interop.Access
Imports System.Windows.Forms

<GuidAttribute("93ED84AA-C1A0-463B-B293-9210E26B4AC8"), ProgIdAttribute("DokoAddIn.Connect")>
Public Class Connect
    Implements Extensibility.IDTExtensibility2
```

In der folgenden Variablen speichern wir eine Referenz auf die aufrufende Access-Anwendung:

```
Dim AccessApp As Access.Application
Dim addInInstance As Microsoft.Office.Core.COMAddIn
```

Hier werden einige Verweise auf Access-Objekte gespeichert:

```
Private WithEvents _Button1 As Access.CommandButton
Private WithEvents _combo1 As Access.ComboBox
Private WithEvents _form As Access.Form
```

Die erste Kontaktaufnahme zwischen Add-In und Access:

```
Public Sub OnConnection(application As Object, connectMode As
    Extensibility.ext_ConnectMode, addInInst As Object,
    ByRef custom As System.Array) _
    Implements Extensibility.IDTExtensibility2.OnConnection
```

Wir speichern die Verweise ab (Typisierung nötig, da nur *Object* übergeben wird):

```
AccessApp = CType(application, Access.Application)
addInInstance = CType(addInInst, Microsoft.Office.Core.COMAddIn)
```

Der Add-In-Instanz wird das aktuelle Objekt (*Me*) übergeben:

```
addInInstance.Object = Me
End Sub
```

Über obiges *addInInstance.Object* greifen wir auch in Access auf die Member des Add-Ins zu.

Eine erste Methode für unser Add-In:

```
Public Sub Info()
    MsgBox.Show("Hallo VB-User")
End Sub
```

Mit der folgenden Methode verbinden wir die Access-Objekte mit unserem Add-In. Dazu übergeben wir in Access die Referenzen auf die gewünschten Objekte:

```
Public Sub ControlsAnbinden(btn As Access.CommandButton,
    cb As Access.ComboBox, frm As Access.Form)
```

Abspeichern der Referenzen:

```
_button1 = btn
_form = frm
_combol = cb
```

Ab hier dürfte sich das Programm kaum von einem VBA-Programm unterscheiden, wir arbeiten mit den Access-Objekten mit dem "kleinen" Unterschied, dass es sich um VB handelt:

```
_button1.Caption = "Hier geht die Post ab ..."
```

Die folgende Deklaration ist zwar nicht für VB nötig, Access feuert aber nicht das Ereignis, wenn die folgende Zuweisung fehlt:

```
_button1.OnClick = "[Event Procedure]"
_form.OnClose = "[Event Procedure]"
```

Wir füllen eine Access-ComboBox:

```
_combol.RowSourceType = "Wertliste"
_combol.AddItem("Zeile 1")
_combol.AddItem("Zeile 2")
_combol.AddItem("Zeile 3")
_combol.AddItem("Zeile 4")
_combol.AddItem("Zeile 5")
_combol.AddItem("Zeile 6")
```

Noch die Schaltfläche beschriften

```
_form.Befehl0.Caption = "Klick mich ..." ' Nicht schön, aber möglich
End Sub
```

Die Ereignisbehandlung für den *Button*:

```
Private Sub _button1_click() Handles _button1.Click
    Dim s As String
```

Eine einfache Meldung anzeigen (nur zur Kontrolle):

```
    MessageBox.Show("Button_Click")
```

Wir instanziierten ein WinForm und füllen es mit Daten aus einem Access-Formular:

```
    Dim f As New Form1
    f.TextBox1.Text = _combo1.RowSource.Replace(";", Chr(13) & Chr(10))
```

Anzeige des Dialogs:

```
    f.ShowDialog()
```

Auswerten und Anpassen der ComboBox in Access:

```
    _combo1.RowSource = ""
    s = f.TextBox1.Text.Replace(Chr(13) & Chr(10), ";")
    _combo1.RowSource = s
End Sub
```

Hier könnten wir auf das Schließen des Access-Formulars reagieren:

```
Private Sub _form_Close() Handles _form.Close
    MessageBox.Show("Und jetzt ist das Formular zu ....")
End Sub

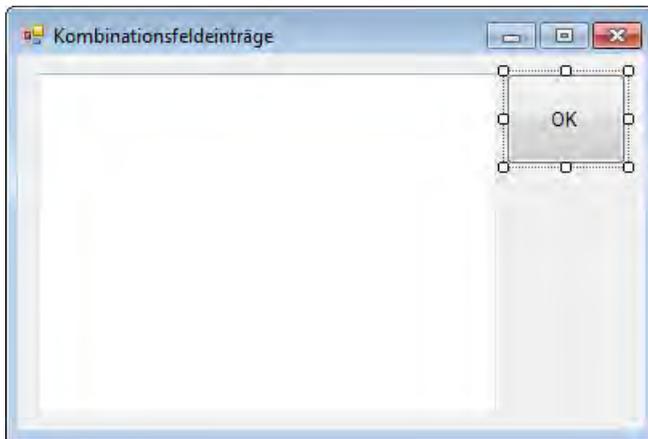
'''
End Class
```

HINWEIS: Wir hätten obige Aufgabenstellung sicher auch mit Webservices oder Remoting bzw. den Zugriff auf den SQL Server bereichern können, aber das Beispiel wird dadurch sicher nicht übersichtlicher.

Oberfläche Add-In

Wie Sie dem obigen Listing entnehmen konnten, rufen wir auch ein .NET-Formular auf. Dieses müssen wir zunächst erstellen (Menü *Projekt/Windows Form hinzufügen*). Fügen Sie eine *TextBox* und einen *Button* ein und legen Sie die *Multiline*-Eigenschaft der *TextBox* mit *True* fest.

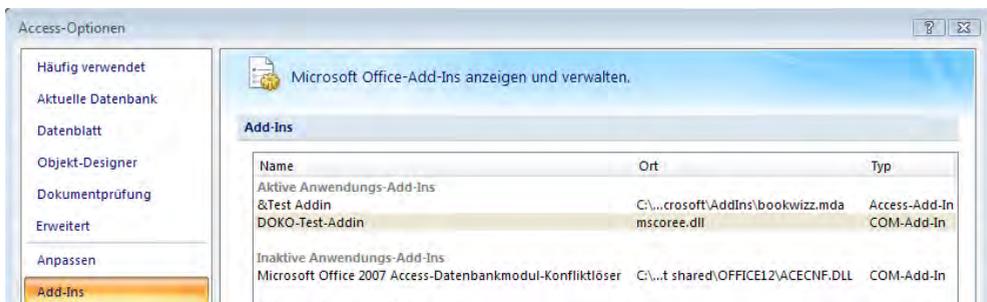
Damit wir das Formular auch deutlich von einem Access-Formular unterscheiden können, setzen wir die *Opacity*-Eigenschaft (Transparenz) auf 50%.



Erstellen der Access-Anwendung

Kompilieren und Einbinden

Nach dem Kompilieren des Add-Ins in Visual Studio (Menü *Erstellen*) können Sie sich bereits in Microsoft Access (*Optionen*) von der Anwesenheit des neuen Add-Ins überzeugen:



Doch wie können wir auf das Add-In zugreifen?

Eine kurze VBA-Routine (öffnen Sie dazu den VBA-Editor) zeigt die Vorgehensweise, bevor wir uns an ein eigenes Formular wagen:

```
Sub test()
    With COMAddIns("DOKOAddIn.Connect")
        .Connect = True
    End With
End Sub
```

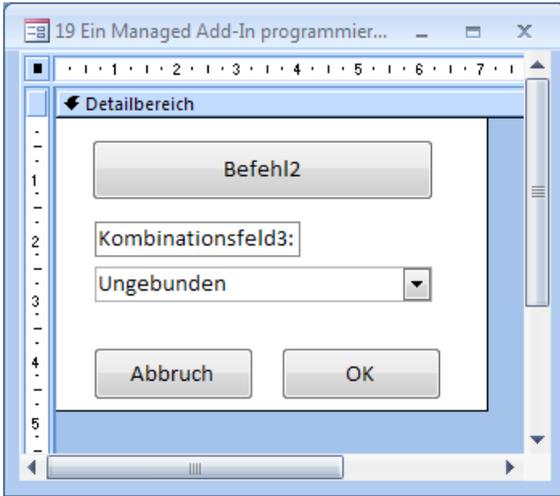
Hier rufen wir unsere Methode auf:

```
.Object.info
End With
End Sub
```

Nach dem Start mit *F5* sollte unsere *.NET-MessageBox* erscheinen.

Das Access-Formular

Erstellen Sie ein einfaches Access-Formular nach folgender Vorlage:



Mit dem Laden des Access-Formulars wird folgender Code ausgeführt:

```
Private Sub Form_Load()
    With COMAddIns("DOKOAddIn.Connect")
        .Connect = True
        .Object.ControlsAnbinden Befehl2, Kombinationsfeld1, Me ' Referenzen übergeben
    End With
End Sub
```

Test

Nach dem Starten des Formulars und dem Klick auf die obere Schaltfläche (beachten Sie die geänderte Beschriftung!) wird unser halbtransparentes *.NET-Formular* angezeigt:



Auch beim Schließen des Formulars erscheint die Meldung aus dem Add-In:



Bemerkungen

Wie Sie gesehen haben, ist eine nahtlose Integration in die Access-Umgebung möglich.

HINWEIS: Verwenden Sie im Add-In ADO-Objekte, dürfte auch der Datenzugriff auf die gerade geöffnete Datenbank kein Problem sein.

Sicher konnten wir hier zu dieser komplexen Materie nur einen ersten Einblick gewähren, das relativ einfach umsetzbare Grundprinzip dürfte jedoch erkennbar geworden sein.

Mit der kostenpflichtigen Software Add-in Express™ können Sie auch in Visual Studio 2012 Add-Ins für ältere Office-Versionen erstellen:

LINK: <http://www.add-in-express.com/add-in-net/index.php>

R199 Sound per MCI aufnehmen

Im vorliegenden Rezept wollen wir uns der Aufnahme von WAV-Dateien widmen, wobei (mittels unveraltetem Code) auf Windows-Bordmittel, d.h. auf die "gute alte" MCI-Schnittstelle, zurückgegriffen wird.

Oberfläche

Ein Formular mit einer Schaltfläche, die Sie mit *START* beschriften, genügt.

Quellcode

```
Imports System.IO
Imports System.Runtime.InteropServices
```

```
Public Class Form1
```

Die folgenden zwei API-Deklarationen greifen auf die MCI-Schnittstelle Ihres PCs zu:

```
<DllImport("winmm.dll", CharSet:=CharSet.Auto)>
```

```

Private Shared Function mciSendString(lpstrCommand As String,
                                     lpstrReturnString As StringBuilder, uReturnLength As Integer,
                                     hwndCallback As IntPtr) As Integer

End Function

<DllImport("winmm.dll", CharSet:=CharSet.Auto)>
Private Shared Function mciGetErrorString(dwError As Integer,
                                         lpstrBuffer As StringBuilder, uLength As Integer) As Integer

End Function

```

MCI-Fehlermeldung dekodieren (wir erhalten ausführliche Fehlerbeschreibungen):

```

Private Function getMciError(errCode As Integer) As String
    Dim errMsg As New StringBuilder(255)
    If mciGetErrorString(errCode, errMsg, errMsg.Capacity) = 0 Then
        Return "MCI-Fehler " & errCode
    Else
        Return errMsg.ToString()
    End If
End Function

```

Mit Hilfe der API-Funktion *GetShortPathName* können wir den kurzen Pfadnamen zum Speicherort der WAV-Datei ermitteln, da ansonsten (z.B. wegen Leerzeichen im Dateinamen etc.) die WAV-Datei von *mciSendString* nicht gespeichert werden kann:

```

<DllImport("kernel32.dll", CharSet:=CharSet.Auto)>
Private Shared Function GetShortPathName(lpszLongPath As String,
                                         lpszShortPath As StringBuilder, cchBuffer As Integer) As Integer

End Function

Private Function shortPathName() As String
    Dim shortPath As String = String.Empty
    Dim len As Long = 0
    Dim buffer As New StringBuilder(256)
    Dim s As String = Directory.GetCurrentDirectory()
    len = GetShortPathName(s, buffer, 256)
    shortPath = buffer.ToString()
    Return shortPath
End Function

```

Nach all diesen Vorbereitungen kann es nun endlich losgehen. Bereits beim Laden des Formulars wird die MCI-Schnittstelle angesprochen:

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
```

Eine neue (leere) Sound-Aufnahme eröffnen:

```

    Dim mciString As String = "open new type waveaudio alias myAlias"
    Dim res As Integer = mciSendString(mciString, Nothing, 0, IntPtr.Zero)
    If res <> 0 Then
        MessageBox.Show(getMciError(res),
                        "MCI-Fehler beim Öffnen des Geräts ('Open New'-Befehl)",

```

```

        MessageBoxButtons.OK, MessageBoxIcon.Error)
    End If

```

Parameter einstellen: 8000 Abtastungen pro Sekunde mit je 1 Byte (da es sich um die Standardwerte handelt, könnte dieser MCI-Aufruf auch weggelassen werden):

```

        mciString = "set myAlias time format ms bitspersample" &
            " 8 channels 1 samplespersec 8000 bytespersec 8000"
    res = mciSendString(mciString, Nothing, 0, IntPtr.Zero)
    If res <> 0 Then
        MessageBox.Show(getMciError(res),
            "MCI-Fehler beim Zuweisen der Parameter ('Set-Befehl)",
            MessageBoxButtons.OK, MessageBoxIcon.Error)
    End If
End Sub

```

Und schließlich der *START-/STOPP*-Button:

```

Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    If Button1.Text = "START" Then
        ' auf START geklickt

```

Die Aufnahme beginnt:

```

        Dim mciString As String = "record myAlias"
        Dim res As Integer = mciSendString(mciString, Nothing, 0, IntPtr.Zero)
        If res <> 0 Then
            MessageBox.Show(getMciError(res),
                "MCI-Fehler beim Starten der Aufnahme ('Record-Befehl)",
                MessageBoxButtons.OK, MessageBoxIcon.Error)
        End If
        Button1.Text = "STOPP"
    Else
        ' auf STOPP geklickt

```

Die Sound-Datei wird im aktuellen Verzeichnis abgespeichert:

```

        Dim mciStr As String = "save myAlias " & shortPathName() & "\Test.wav"
        Dim res As Integer = mciSendString(mciStr, Nothing, 0, IntPtr.Zero)
        If res <> 0 Then
            MessageBox.Show(getMciError(res),
                "MCI-Fehler beim Speichern der Datei ('Save'-Befehl)",
                MessageBoxButtons.OK, MessageBoxIcon.Error)
            Button1.Text = "START"
        End If
    End If
End Sub
End Class

```

Test

Vergewissern Sie sich, dass an Ihren PC ein Mikrofon angeschlossen ist und dass es korrekt funktioniert (siehe Bemerkungen).

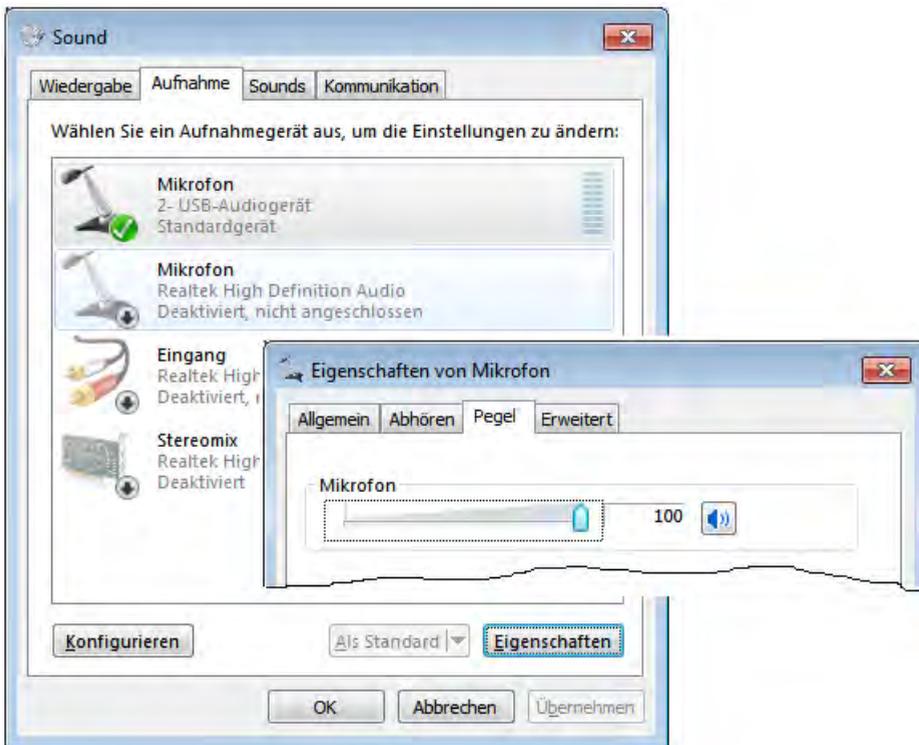
Nach Klick auf die *START*-Schaltfläche ändert sich deren Beschriftung in *STOPP* und die Aufnahme beginnt. Jetzt können Sie zum Beispiel eine Probe Ihrer Gesangkünste abliefern. Nach Aufnahmestopp findet sich die Sounddatei *Test.wav* im aktuellen Projektverzeichnis (bzw. im *.../bin/Debug*-Unterverzeichnis Ihres Projektordners).

Nach Doppelklick auf die Sounddatei dürfte sich normalerweise der Windows Media Player öffnen und Sie können sich das Ergebnis Ihrer gesanglichen Darbietungen anhören, oder aber Sie verwenden zur Wiedergabe das Rezept

► R202 Sound- und Video-Dateien per MCI abspielen

Bemerkungen

- Über die Systemsteuerung (oder auch die Taskbar) haben Sie Zugriff auf die Dialoge für Mikrofonauswahl und Pegelinstellungen:



- Bei schlechter Soundqualität sollten Sie vergleichsweise zunächst mit dem im Windows-Zubehör enthaltenen Sound Recorder einige Probeaufnahmen machen.
- Die ausführlichen MCI-Meldungstexte helfen Ihnen bei der Fehlersuche, zum Beispiel wenn ein falscher oder doppelt gebener Alias verwendet wurde.

R200 Mikrofonpegel anzeigen

Bei Soundaufnahmen wünscht man sich meist auch eine Anzeige des Mikrofonpegels, z.B. in Form eines Aussteuerungsbalkens. Im Folgenden wollen wir das Vorgängerrezept R199 mit einer einfachen Pegelanzeige ergänzen.

Oberfläche

Zum Formular des Vorgängerbeispiels fügen Sie eine *ProgressBar* (*Maximum* = 128) und einen *Timer* (*Interval* = 10) hinzu.

Quellcode

Ergänzen Sie zunächst den *Load*-Eventhandler des Formulars wie folgt:

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    ...
```

Leider kann ein geöffnetes Audiogerät nicht gleichzeitig eine Sound-Datei aufnehmen und den Pegel anzeigen. Wir öffnen deshalb eine zweite Sound-Aufnahme mit unterschiedlichem Aliasnamen:

```
Dim mciString As String = "open new type waveaudio alias myAlias"
Dim res As Integer = mciSendString(mciString, Nothing, 0, IntPtr.Zero)
If res <> 0 Then
    MessageBox.Show(getMciError(res),
                    "MCI-Fehler beim Öffnen des Geräts ('Open New'-Befehl)",
                    MessageBoxButtons.OK, MessageBoxIcon.Error)
End If
```

Auf eine Parameterzuweisung können wir in unserem Fall verzichten, da mit den Standardwerten gearbeitet wird.

```
Timer1.Start()
End Sub
```

In der zweiten Programmergänzung wird im 10ms Takt der Pegel abgerufen:

```
Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles Timer1.Tick
```

Der aktuelle Pegel kann mittels *status*-Befehl über das *level*-Flag abgefragt werden:

```
Dim mciString As String = "status myAlias2 level"
Dim buffer As New StringBuilder(20)
Dim res As Integer = mciSendString(mciString, buffer, buffer.Capacity, IntPtr.Zero)
If res <> 0 Then
    MessageBox.Show(getMciError(res),
                    "MCI-Fehler bei der Status-Abfrage ('status ... level'-Befehl)",
                    MessageBoxButtons.OK, MessageBoxIcon.Error)
End If
```

Der Input-Level ist ein vorzeichenloser Wert. Für die von uns gewählte 8-Bit Abtastgenauigkeit liegt er zwischen 0 und 127 (0x7F), bei 16-Bit Abtastung zwischen 0 und 32.767 (0x7FFF):

```
Dim level As Byte = Byte.Parse(buffer.ToString()) ' 0 ... 127
ProgressBar1.Value = level
End Sub
End Class
```

Test

Die Aussteuerungsanzeige funktioniert sowohl vor als auch während der Aufnahme.



R201 Pegeldiagramm aufzeichnen

So richtig professionell wirkt ein Sound Recorder erst dann, wenn man den Pegelverlauf in Echtzeit auf einem Diagramm verfolgen kann. Diese, u.a. auch bei medizinischen Überwachungsgeräten übliche Darstellung wollen wir anhand einer Programmerweiterung des Vorgängerrezepts

► R200 Mikrofonpegel anzeigen

demonstrieren.

Oberfläche

Ergänzen Sie das vorhandene Formular durch eine *PictureBox* und ein repräsentatives *Label*. Gestalten Sie die Oberfläche entsprechend der Laufzeitabbildung am Schluss dieses Rezepts.

Quellcode-Ergänzungen

Um wegen der zahlreichen Ergänzungen die Übersicht nicht zu verlieren, empfiehlt es sich insbesondere bei diesem Rezept, erst den kompletten Code von den Buchdaten in die Entwicklungsumgebung zu laden und dann mit der Lektüre fortzufahren.

```
...
Imports System.Drawing           ' für die Grafikoperationen
```

Zunächst sind einige globale Variablen hinzuzufügen:

```
Private g As Graphics = Nothing      ' Graphics-Objekt für PictureBox
Private BA() As Byte = Nothing      ' Puffer-Array für die Pegelwerte
Private xmax As Single              ' Breite des Diagramms (Pixel)
Private y0 As Single                ' y-Koordinate der Diagramm-Mittellinie (Pixel)
Private v As Single                  ' Streckungsfaktor der Amplitude
Private i As Integer                ' Zähler für Timer-Ticks ab Start
Private dStart As DateTime          ' Startzeit der Aufnahme
Private msec As Integer              ' seit Start abgelaufene Zeit in Millisekunden

Private iv As Integer                ' Timer-Intervall (ms)
Private Const tmax As Integer = 60000 ' Länge der Zeitachse (ms)
```

Zwecks Initialisierung der globalen Variablen wird der *Load*-Eventhandler ergänzt:

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    g = PictureBox1.CreateGraphics()
    iv = Timer1.Interval           ' Voreinstellung 10ms
```

Die erforderliche Größe des Puffer-Arrays ergibt sich aus obigen Konstanten:

```
ReDim BA(tmax / iv)              ' 6000 gespeicherte Werte (ca. 60sek)
```

Abmessungen des Diagramms:

```
xmax = PictureBox1.Width
y0 = PictureBox1.Height / 2
```

Der vom MCI gelieferte Maximalpegel beträgt 127. Da bei Vollaussteuerung der obere bzw. untere Rand der *PictureBox* nicht ganz erreicht werden soll, bauen wir in den Streckungsfaktor eine kleine Reserve ein:

```
v = y0 / 140
...
End Sub
```

Der *Click*-Eventhandler des Buttons wird modifiziert:

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    If Button1.Text = "START" Then           ' auf START geklickt
        Dim mciString As String = "record myAlias"
        Dim res As Integer = mciSendString(mciString, Nothing, 0, IntPtr.Zero)
        If res <> 0 Then
            MessageBox.Show(getMciError(res),
                "MCI-Fehler beim Starten der Aufnahme ('Record-Befehl)",
                MessageBoxButtons.OK, MessageBoxIcon.Error)
        End If
        Button1.Text = "STOPP"
```

Nach dem Klick auf START beginnt eine neue Aufnahme, d.h., die Startzeit wird zugewiesen, der Zähler zurückgesetzt und das alte Diagramm gelöscht:

```

    dStart = DateTime.Now
    i = 0
    PictureBox1.Refresh()
Else                                     ' auf STOPP geklickt

```

Das Abspeichern der Sound-Datei wird in eine separate Prozedur (siehe unten) verlagert:

```

    saveFile()
    Button1.Text = "START"
End If
End Sub

```

Die Prozedur zum Abspeichern der Sound-Datei im aktuellen Verzeichnis:

```

Private Sub saveFile()
    Dim mciStr As String = "save myAlias " & shortPathName() & "\Test.wav"
    Dim res As Integer = mciSendString(mciStr, Nothing, 0, IntPtr.Zero)
    If res <> 0 Then
        MessageBox.Show(getMciError(res),
            "MCI-Fehler beim Speichern der Datei ('Save'-Befehl)", MessageBoxButtons.OK,
            MessageBoxIcon.Error)
    End If
End Sub

```

Bei einem Timerintervall von 10ms und 6000 Durchläufen ist die Endzeit in der Regel deutlich länger als die erwarteten 60 Sekunden¹. Wollen Sie eine exakte Anzeige der seit Start abgelaufenen Sekunden hinzufügen, so kommen Sie um ein *TimeSpan*-Objekt nicht herum.

Der Aufbau der Grafik erfolgt synchron mit der Pegelabfrage. Der vorhandene *Tick*-Eventhandler des *Timers* ist deshalb wie folgt zu ergänzen:

```

Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles Timer1.Tick
    ...

```

Die folgende Zeile ist kein Fehler, denn während die Soundaufnahme läuft, hat der Button die Beschriftung STOPP:

```

If Button1.Text = "STOPP" Then
    Dim ts As New TimeSpan(DateTime.Now.Ticks - dStart.Ticks)
    Dim sec As Integer = Convert.ToInt32(ts.TotalSeconds)

```

Anzeige der abgelaufenen Zeit (Sekunden):

```

Label1.Text = sec.ToString()

```

Abbruchkriterium bei 60sek:

```

If sec = tmax / 1000 Then

```

¹ Windows ist kein Echtzeit-Betriebssystem und Prozesse mit höherer Priorität können solche mit niedrigerer Priorität ausbremsen.

Speichern am Schluss:

```

        saveFile()
        Button1.Text = "START"
        Return
    End If

```

Um die x -Koordinate exakt berechnen zu können, muss die Auflösung zumindest im Millisekundenbereich liegen:

```

    msec = Convert.ToInt32(ts.TotalMilliseconds)
    Dim x As Single = msec * xmax / tmax

```

Pegel in Diagramm eintragen:

```

        g.DrawLine(New Pen(Color.Red, 0.1F), x, y0 - level * v, x, y0 + level * v)

```

Abspeichern des Pegels im Puffer-Array:

```

        BA(i) = level
        i += 1
    End If
End Sub

```

Damit nach einem Verdecken des Formulars durch andere Fenster das Diagramm nicht auf Nimmerwiedersehen verschwindet, muss es im *Paint*-Eventhandler der *PictureBox* komplett neu gezeichnet werden:

```

Private Sub PictureBox1_Paint(sender As Object, e As PaintEventArgs) _
    Handles PictureBox1.Paint
    Dim g As Graphics = e.Graphics

```

Abstand zwischen zwei gespeicherten Werten auf Position i :

```

    Dim dx As Single = xmax / tmax * msec / i

```

Der Pufferspeicher wird bis zur Position i ausgelesen und angezeigt:

```

    For j As Integer = 0 To i - 1
        g.DrawLine(New Pen(Color.Red, 0.1F), j * dx, y0 - BA(j) * v, j * dx, y0 + BA(j) * v)
    Next j
End Sub

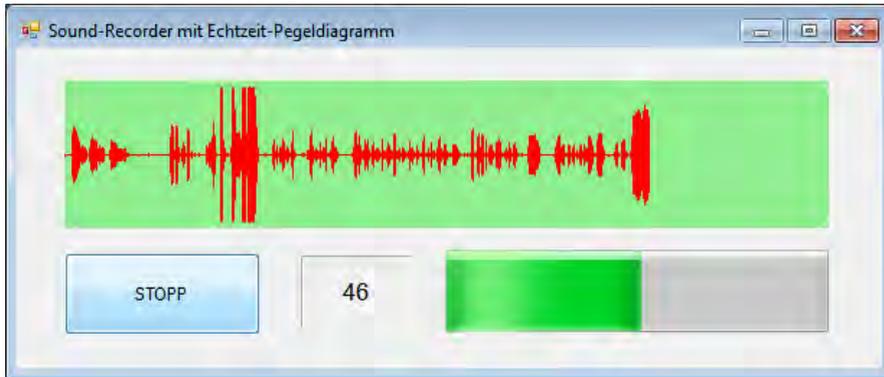
End Class

```

Test

Jetzt können Sie unmittelbar während der Sound-Aufnahme den kontinuierlichen Pegelverlauf "live" beobachten (siehe folgende Abbildung).

Bei STOPP, bzw. nach Ablauf von 60 Sekunden, wird die Datei *Test.wav* automatisch im Projektverzeichnis abgelegt.



R202 Sound- und Video-Dateien per MCI abspielen

Für die Wiedergabe von Sound und Videos in Windows Forms-Anwendungen gibt es bereits einige vorgefertigte .NET- oder auch ActiveX-Steuerelemente (*SoundPlayer*, *WindowsMediaPlayer*, ...).

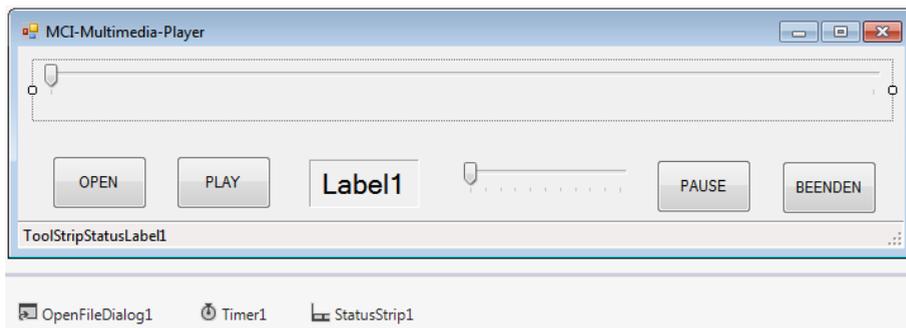
Wollen Sie aber individuelle Wünsche realisieren, wie zum Beispiel eine eigene Benutzeroberfläche, oder einfach nur mit verschiedenen Befehlen experimentieren, so können Sie die MCI-Schnittstelle direkt ansprechen. Diese allerdings ist nicht Teil von .NET, sondern gehört zum Uralt-Bestand von Windows (seit 3.1), d.h., die Fremdbibliothek *winmm.dll* muss importiert werden.

Mittels einfacher Kommandos (*Open*, *Play*, ...) können Sie dann auf alle verfügbaren Multimedia-Geräte zugreifen, für die ein MCI-Treiber existiert.

Das vorliegende Rezept demonstriert dies anhand eines einfachen "selbstgestrickten" Players für Sound- und Videodateien unterschiedlichster Formate (Wave, MP3, Midi, AVI, WMV, MPEG etc.).

Oberfläche

Öffnen Sie eine neue Windows Forms-Anwendung und gestalten Sie die Oberfläche von *Form1* mit vier *Buttons*, zwei *TrackBars* und je einem *Label*, *OpenFileDialog*, *Timer* und *StatusStrip*. Setzen Sie die *TopMost*-Eigenschaft von *Form1* auf *True*.



Klasse C_MCI

Den Zugriff auf die MCI-Schnittstelle kapseln wir zwecks späterer Wiederverwendbarkeit in einer separaten Klasse *C_MCI*, deren Gerüst wir über das Menü *Projekt/Klasse hinzufügen...* erzeugen:

```
...
Imports System.IO
Imports System.Text
```

Vergessen Sie keinesfalls, für den (unmanaged) Zugriff auf die Windows-API den folgenden Namespace einzubinden:

```
Imports System.Runtime.InteropServices
```

```
Public class C_MCI
```

Die beiden erforderlichen MCI-Funktionen sind Teil der Datei *winmm.dll*. Dabei definiert der (optionale) *CharSet*-Parameter die automatische Übergabe von Zeichenfolgen (Marshalling) entsprechend des jeweiligen Zielbetriebssystems.

```
<DllImport("winmm.dll", CharSet:=CharSet.Auto)>
```

Die Funktion *mciSendString* ist das Herzstück des Programms, sie sendet unsere Befehle an die MCI-Schnittstelle:

```
Private Shared Function mciSendString(lpstrCommand As String,
                                     lpstrReturnString As StringBuilder, uReturnLength As Integer,
                                     hwndCallback As IntPtr) As Integer

End Function
```

Die MCI-Funktion *mciGetErrorString* gibt uns Fehlertexte für einen bestimmten Fehlercode zurück, was insbesondere beim Experimentieren sehr hilfreich ist:

```
<DllImport("winmm.dll", CharSet:=CharSet.Auto)>
Private Shared Function mciGetErrorString(dwError As Integer,
                                          lpstrBuffer As StringBuilder, uLength As Integer) As Integer

End Function
```

Mit der Funktion *getMciError* kapseln wir den Aufruf obiger API-Funktion:

```
Private Function getMciError(errCode As Integer) As String
    Dim errMsg As New StringBuilder(255)
    If mciGetErrorString(errCode, errMsg, errMsg.Capacity) = 0 Then
        Return "MCI-Fehler " & errCode
    Else
        Return errMsg.ToString()
    End If
End Function
```

Ein bestimmtes MCI-Gerät wird innerhalb der Befehlszeichenkette über einen frei definierbaren Alias identifiziert:

```
Private _alias As String = "mciAlias"
```

Eine private Zustandsvariable zeigt an, ob das angesprochene MCI-Gerät geöffnet ist:

```
Private _isOpen As Boolean = False
```

Der Lesezugriff:

```
Public ReadOnly Property IsOpen As Boolean
    Get
        Return _isOpen
    End Get
End Property
```

Alles beginnt mit dem Befehl zum Öffnen einer Multimedia-Datei:

```
Public Sub Open(filename As String)
```

Vorher schließen wir eine eventuell zuvor noch geöffnete Datei:

```
    If _isOpen Then Me.Close()
```

Nun wird überprüft, ob die übergebene Multimedia-Datei tatsächlich existiert:

```
    If Not File.Exists(filename) Then
        MessageBox.Show("Die Datei '" & filename & "' ist nicht vorhanden!", "Fehler",
            MessageBoxButtons.OK, MessageBoxIcon.Error)
    End If
```

Jetzt kommen wir zum wohl wichtigsten Parameter, dem MCI-Befehlsstring. Dieser besteht aus einzelnen Kommandos, die durch Leerzeichen voneinander getrennt sind.

Der komplette MCI-Befehlsstring zum Öffnen einer Datei besteht aus dem *Open*-Kommando, dem Dateinamen, dem Medientyp und dem Alias:

```
    Dim mciStr As String = "open " & Chr(34) & filename & Chr(34) &
        " type mpegvideo alias " & _alias
```

Notwendige Erklärung zu obiger Zeile: Da auch der Dateipfad Leerzeichen enthalten kann, führt das häufig zu unergründbaren Fehlern. Wir müssen deshalb den Dateinamen in doppelte Anführungszeichen einschließen, also gewissermaßen "String im String" realisieren, am einfachsten geht das unter VB mit *Chr(34)*.

Nun kann das MCI-Gerät geöffnet werden:

```
    Dim res As Integer = mciSendString(mciStr, Nothing, 0, IntPtr.Zero)
```

Zur Fehleranzeige verwenden wir hier Einfachheit halber ein Meldungsfenster, könnten aber stattdessen auch eine eigene Fehlerklasse definieren und eine Ausnahme werfen (*Throw MCI_Exception*):

```
    If res <> 0 Then
        MessageBox.Show(getMciError(res),
            "MCI-Fehler beim Öffnen des Geräts ('Open'-Befehl)",
            MessageBoxButtons.OK, MessageBoxIcon.Error)
    End If
Exit Sub
End If
```

HINWEIS: Eine Fehlerbehandlung wie die obige haben wir bei allen MCI-Aufrufen implementiert, im Folgenden aber aus Platz- und Übersichtlichkeitsgründen darauf verzichtet (siehe Buch-Beispieldaten).

Das Zeitformat für Positionsangaben auf Millisekunden festlegen:

```
mciStr = "set " & Me._alias & " time format ms"
res = mciSendString(mciStr, Nothing, 0, IntPtr.Zero)
```

Unsere Multimedia-Datei ist nun geöffnet:

```
_isOpen = True
End Sub
```

Was nun, ist die Kapselung anderer wichtiger MCI-Kommandos als Eigenschaften oder Methoden.

Die folgende Eigenschaft liefert die Abspiellänge der Multimedia-Datei (in Millisekunden):

```
Public ReadOnly Property Length As Integer
    Get
        Dim buffer As New StringBuilder(255)
        Dim res As Integer = mciSendString("status " & _alias & " length", buffer,
            buffer.Capacity, IntPtr.Zero)
        Return Convert.ToInt32(buffer.ToString())
    End Get
End Property
```

Die Multimedia-Datei zwischen zwei Positionen abspielen:

```
Public Sub Play(von As Integer, bis As Integer)
    Dim res As Integer = mciSendString("Play " & _alias & " From " & von & " To " &
        bis, Nothing, 0, IntPtr.Zero)
End Sub
```

Die Multimedia-Datei in voller Länge abspielen:

```
Public Sub Play()
    Play(0, Me.Length)
End Sub
```

Die folgende Eigenschaft liest/schreibt die aktuelle Abspielposition (Millisekunden):

```
Public Property Position As Integer
    Get
        Dim buffer As New StringBuilder(261)
        Dim res As Integer = mciSendString("status " & _alias & " position",
            buffer, buffer.Capacity, IntPtr.Zero)
        Return Convert.ToInt32(buffer.ToString())
    End Get
    Set(value As Integer)
        Dim res As Integer = mciSendString("seek " & _alias & " to " & value,
            Nothing, 0, IntPtr.Zero)
```

```

        res = mciSendString("play " & _alias, Nothing, 0, IntPtr.Zero)
    End Set
End Property

```

Eine Pause:

Den Abspielvorgang unterbrechen:

```

Public Sub Pause()
    Dim res As Integer = mciSendString("Pause " & _alias, Nothing, 0, IntPtr.Zero)

End Sub

```

Den Abspielvorgang fortsetzen:

```

Public Sub Resume_()
    Dim res As Integer = mciSendString("Resume " & _alias, Nothing, 0, IntPtr.Zero)
End Sub

```

Die Lautstärke festlegen:

```

Public Sub volume(wert As Integer)
    Dim res As Integer = mciSendString("setaudio " & _alias & " volume to " & wert,
        Nothing, 0, IntPtr.Zero)
End Sub

```

Den Abspielvorgang stoppen und auf den Anfang zurücksetzen:

```

Public Sub Stop_()
    Dim res As Integer = mciSendString("Stop " & _alias, Nothing, 0, IntPtr.Zero)
End Sub

```

Ein geöffnetes MCI-Gerät schließen:

```

Public Sub Close()
    If _isOpen Then
        Dim res As Integer = mciSendString("Close " & _alias, Nothing, 0, IntPtr.Zero)
        _isOpen = False
    End If
End Sub

```

```
End Class
```

Form1

Nun steht der Realisierung unseres Multimedia-Players nichts mehr im Weg, und wir können die eingangs entworfene Bedienoberfläche beispielsweise mit folgendem Code hinterlegen:

```
Public Class Form1
```

Die beim Laden des Formulars zugewiesenen Anfangswerte könnten Sie stattdessen auch direkt im Eigenschaftenfenster zuweisen:

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
```

```

    TrackBar1.Maximum = 10000
    TrackBar1.LargeChange = 1
    TrackBar2.Maximum = 1000
    TrackBar2.Value = 1000
    TrackBar2.LargeChange = 100
    Timer1.Interval = 100
End Sub

```

Eine Instanz unseres Multimedia-Players erzeugen:

```
Private myMCI As New C_MCI()
```

Korrekturfaktor für *TrackBar*-Anzeige:

```
Private kf As Single = 1
```

Eine Multimedia-Datei öffnen (OPEN-Schaltfläche):

```

Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    If OpenFileDialog1.ShowDialog() = DialogResult.OK Then
        myMCI.Open(OpenFileDialog1.FileName)
    End If
End Sub

```

Bei der Berechnung des Korrekturfaktors ist eine Typkonvertierung erforderlich, da sonst die Nachkommastellen abgeschnitten werden:

```

    kf = Convert.ToSingle(myMCI.Length) / TrackBar1.Maximum
    ToolStripStatusLabel1.Text = OpenFileDialog1.FileName
    Button2.Text = "PLAY"
End If
End Sub

```

Die Schaltfläche PLAY/STOPP:

```

Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
    If Button2.Text = "PLAY" Then
        myMCI.Play()
        Timer1.Enabled = True
        Button2.Text = "STOPP"
    Else
        myMCI.Stop_()
        Timer1.Enabled = False
        TrackBar1.Value = 0
        Label1.Text = "00:00"
        Button2.Text = "PLAY"
    End If
End Sub

```

Die Schaltfläche PAUSE/WEITER:

```

Private Sub Button3_Click(sender As Object, e As EventArgs) Handles Button3.Click
    If Button3.Text = "PAUSE" Then
        myMCI.Pause()
        Button3.Text = "WEITER"
    End If
End Sub

```

```

Else
    Button3.Text = "PAUSE"
    myMCI.Resume_()
End If
End Sub

```

Es folgt eine Hilfsfunktion für die Bereitstellung der laufenden Zeitanzeige in Minuten und Sekunden. Anhand der Reglerposition von *TrackBar1* werden zunächst die Sekunden ermittelt. Mittels Integer- und Modulo-Division werden dann die Minuten und restlichen Sekunden berechnet und durch einen Doppelpunkt getrennt:

```

Private Function getMinutes() As String
    Dim sekunden As Integer = Convert.ToInt32((TrackBar1.Value * kf / 1000))
    Return (sekunden / 60).ToString("00") & ":" & (sekunden Mod 60).ToString("00")
End Function

```

Im *Tick*-Event des *Timers* werden der Regler der *Trackbar* vorwärts bewegt und die Zeitanzeige aktualisiert:

```

Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles Timer1.Tick
    Dim val As Integer = Convert.ToInt32(myMCI.Position / kf)
    If val < TrackBar1.Maximum Then
        TrackBar1.Value = val
        Label1.Text = getMinutes()
    End If
End Sub

```

Das Ende unserer MM-Datei ist erreicht:

```

Else
    myMCI.Stop_()
    Timer1.Enabled = False
    TrackBar1.Value = 0
    Label1.Text = "00:00"
    Button2.Text = "PLAY"
End If
End Sub

```

Wenn Sie am Regler der *TrackBar* ziehen, können Sie die Abspielposition verändern:

```

Private Sub TrackBar1_Scroll(sender As Object, e As EventArgs) Handles TrackBar1.Scroll
    Try
        Me.myMCI.Position = Convert.ToInt32(TrackBar1.Value * kf)
    Catch ex As Exception
        MessageBox.Show(ex.Message, Application.ProductName,
            MessageBoxButtons.OK, MessageBoxIcon.Error)
    End Try
End Sub

```

Die Lautstärke lässt sich mit der kleinen *TrackBar* steuern:

```

Private Sub TrackBar2_Scroll(sender As Object, e As EventArgs) Handles TrackBar2.Scroll
    myMCI.volume(TrackBar2.Value)
End Sub

```

Die Schaltfläche zum Beenden:

```
Private Sub Button4_Click(sender As Object, e As EventArgs) Handles Button4.Click
    Me.Close()
End Sub
```

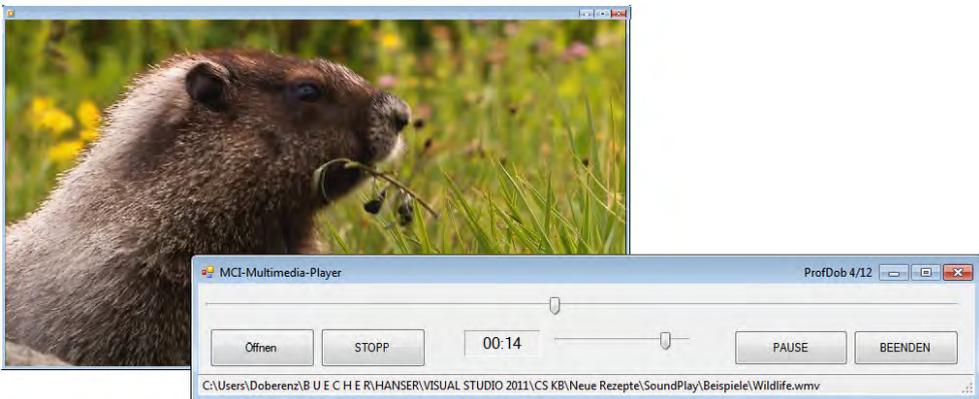
MCI-Player schließen:

```
Private Sub Form1_FormClosing(sender As Object, e As FormClosingEventArgs) _
    Handles MyBase.FormClosing
    myMCI.Close()
End Sub
End Class
```

Test

Ring frei für Ihre Experimente! Starten Sie das Programm und laden Sie über die *Öffnen*-Schaltfläche eine beliebige Sound- oder Video-Datei (.wav, .wma, .mp3, .midi, .avi, .wmv)¹. Spulen Sie mit dem oberen Regler vor und zurück und testen Sie die übrigen Bedienfunktionen.

Die Abbildung zeigt eine Momentaufnahme beim Abspielen der zu Windows mitgelieferten Video-datei *Wildlife.wmv*.



Bemerkungen

- In einer Windows Forms-Anwendung könnten Sie MCI auch durch die DirectX-Bibliothek ersetzen. Letztere müsste dann aber in vielen Fällen separat installiert werden, während hingegen jeder Computer von Haus aus auch über die erforderlichen MCI-Treiber verfügt.
- MCI bietet eine enorme Vielfalt von Befehlen für unterschiedlichste Dateiformate und Geräte, über die Sie sich am besten bei

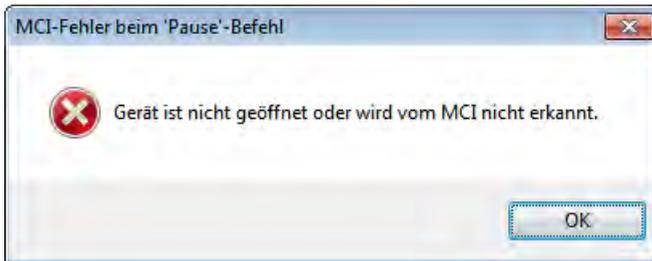
LINK: [http://msdn.microsoft.com/en-us/library/windows/desktop/dd743572\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/dd743572(v=vs.85).aspx)

¹ Beispieldateien finden sich im Windows-Zubehör oder auch in den Buchdaten.

informieren. So können Sie zum Beispiel mit dem folgenden Aufruf die Position und die Abmessungen des Fensters für die Video-Wiedergabe individuell festlegen:

```
mciSendString("put " & _alias + " window at " + x + " " + y + " " + width + " " +  
height, Nothing, 0, IntPtr.Zero)
```

- Auf fehlerhafte MCI-Kommandos werden Sie durch aussagekräftige Fehlermeldungen hingewiesen, zum Beispiel:



R203 Eine C-DLL in VB einbinden

Aus verschiedensten Gründen kann es mitunter notwendig sein, dass Sie von Ihrer VB-Anwendung aus auch auf unverwalteten (unmanaged) Code zugreifen müssen. Sei es, dass Sie eine aus Altbeständen noch vorhandene und in C++ geschriebene DLL nutzen oder aber einen Teil der Programmlogik auslagern wollen, um eine Offenlegung Ihres wertvollen VB-Quellcodes zu verhindern oder zumindest zu erschweren¹.

Die "Mehrzweckwaffe" Visual Studio 2012 erlaubt Ihnen unter anderem auch das Erstellen von C++-DLLs, was wir am Beispiel einer einfachen Sortieroutine, die wir in ein VB-Projekt einbinden werden, demonstrieren wollen.

HINWEIS: Das vorliegende Rezept befasst sich mit dem Programmieren einer "echten" DLL, also nicht mit ActiveX-DLLs bzw. COM-Komponenten.

DLL-Projekt erstellen

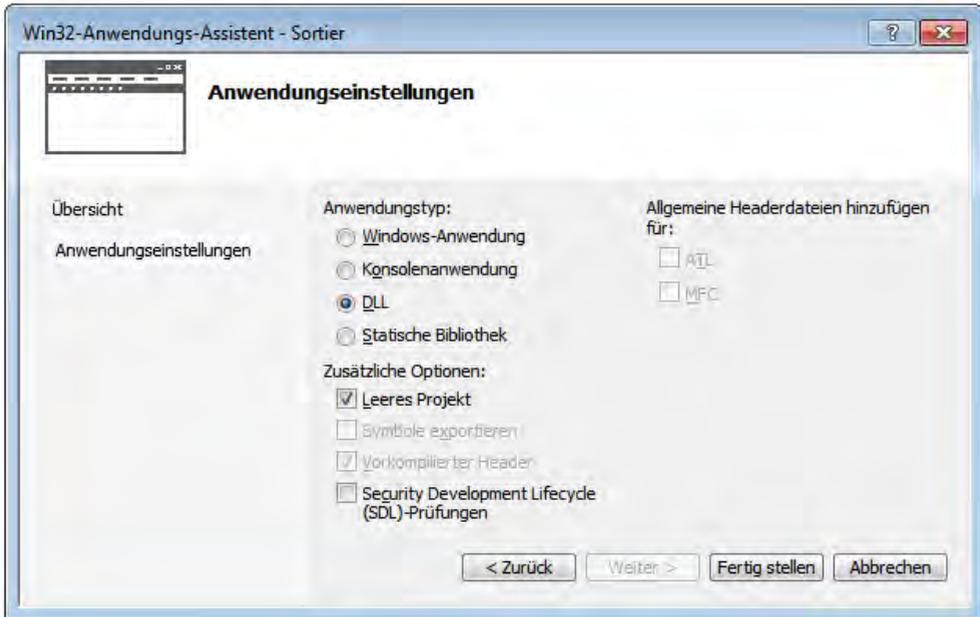
Starten Sie die Entwicklungsumgebung Visual Studio 2012 und klicken Sie auf *Neues Projekt...* Wählen Sie links unter der Vorlage *Andere Sprachen* die Sprache *Visual C++* und in der mittleren Spalte *Win32-Projekt*.

Vergeben Sie den Namen *Sortier* für das Projekt und klicken Sie auf *Ok*.

¹ Das Decompilieren einer .NET-Assembly ist leider sehr leicht möglich, was einen Schutz Ihres geistigen Eigentums schwierig und teuer macht.

HINWEIS: Die Variante *MFC DLL* bringt nur unnötigen Overhead mit sich, den wir nicht benötigen.

In weiteren Verlauf öffnet sich ein neuer Assistent, in welchem wir den Projekttyp spezifizieren. Markieren Sie die Optionen *DLL* und *Leeres Projekt* und klicken Sie auf *Fertig stellen*.



Wählen Sie das Menü *Ansicht/Projektmappen-Explorer*. Dieser präsentiert Ihnen ein noch leeres Projekt, es sind derzeit keine Dateien vorhanden. In ein DLL-Projekt müssen Sie aber zumindest zwei Dateien einbinden: eine *.def*-Datei und eine Datei, in der sich Ihr C-Quellcode (*.cpp*) befindet.

CPP-Datei

Wählen Sie *Projekt/Neues Element hinzufügen...* und dann *C++-Datei(.cpp)* und speichern Sie diese unter ihrem Standardnamen *Quelle.cpp* im Projektverzeichnis ab. Im (noch leeren) Codefenster wartet jetzt einige Arbeit als C++-Programmierer auf uns.

Binden Sie zunächst die Header-Datei *windows.h* zwecks Unterstützung grundlegender Windows-Deklarationen ein:

```
#include <windows.h>
```

Bei der folgenden *DllMain* handelt es sich um die so genannte DLL-Eintrittsprozedur, dem Pendant zum Hauptprogramm einer normalen Anwendung. Mit dieser Prozedur reagieren wir auf das Laden/Entladen unserer DLL durch externe Anwendungen. Dazu ist, wie in den meisten anderen Fällen auch, der Parameter *dwReason* auszuwerten. Im Beispiel zeigen wir zwei Meldungsfenster an:

```

BOOL WINAPI DllMain (HINSTANCE hDLL, DWORD dwReason, LPVOID lpReserved)
{
    switch (dwReason) {
        case DLL_PROCESS_ATTACH:
            MessageBoxA( GetFocus(), "DLL wurde geladen", "Info", MB_OK | MB_SYSTEMMODAL);
            break;
        case DLL_PROCESS_DETACH:
            MessageBoxA( GetFocus(), "DLL wurde entladen", "Info", MB_OK | MB_SYSTEMMODAL);
            break;
    }
    return TRUE;
}

```

Nun endlich kommen wir zu unserer Sortieroutine. Dabei ist zu beachten, dass alle Funktionen, die wir aus der DLL exportieren möchten, eine bestimmte Aufrufkonvention einhalten müssen. Diese beeinflusst die Ausrichtung der übergebenen Argumente auf dem Stack (von rechts nach links), verändert den internen Namen und erweitert die übergebenen Argumente auf ein Vielfaches von 4 Byte. Gleichzeitig ist damit bestimmt, wer für das Aufräumen des Stacks zuständig ist (DLL oder aufrufendes Programm). Für unsere Zwecke ist die `_stdcall`-Syntax geeignet:

```

void _stdcall Sortiere(int a[], int n)
{
    int k, i, j;
    int h;
    k = n/2;
    while (k>0)
    {
        for (i=0; i<n-k; i++)
        {
            j=i;
            while (j>=0 && a[j]>a[j+k])
            {
                h=a[j]; a[j]=a[j+k]; a[j+k]=h;
                j=j-k;
            }
        }
        k=k/2;
    }
}

```

DEF-Datei

Wählen Sie *Projekt/Neues Element hinzufügen...* und dann *DEF-Datei(.def)* und speichern Sie diese unter ihrem Standardnamen *Source.def* im Projektverzeichnis ab.

Der Code verweist auf die zu exportierende Routine:

```

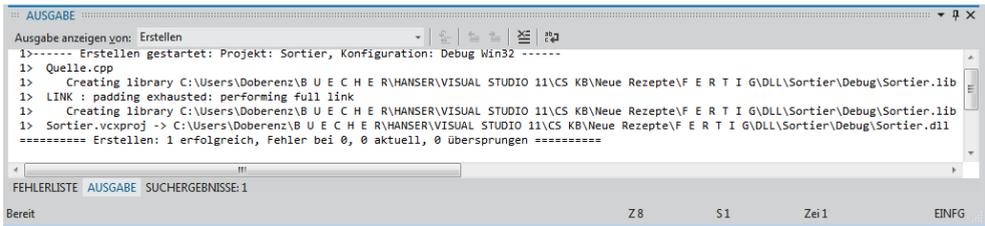
LIBRARY Sortier
EXPORTS
    Sortiere @1

```

Erstellen der DLL

Öffnen Sie das Ausgabefenster (Menü *Ansicht/Ausgabe*) und klicken Sie anschließend auf den Menüpunkt *Erstellen/Projektmappe erstellen*.

Am unteren Bildrand können Sie jetzt C++-Compiler und -Linker in Aktion erleben:



```

AUSGABE
Ausgabe anzeigen von: Erstellen
1>----- Erstellen gestartet: Projekt: Sortier, Konfiguration: Debug Win32 -----
1> Quelle.cpp
1> Creating library C:\Users\Doberenz\B U E C H E R\HANSER\VISUAL STUDIO 11\CS KB\Neue Rezepte\F E R T I G\DLL\Sortier\Debug\Sortier.lib
1> LINK : padding exhausted: performing full link
1> Creating library C:\Users\Doberenz\B U E C H E R\HANSER\VISUAL STUDIO 11\CS KB\Neue Rezepte\F E R T I G\DLL\Sortier\Debug\Sortier.lib
1> Sortier.vcxproj -> C:\Users\Doberenz\B U E C H E R\HANSER\VISUAL STUDIO 11\CS KB\Neue Rezepte\F E R T I G\DLL\Sortier\Debug\Sortier.dll
***** Erstellen: 1 erfolgreich, Fehler bei 0, 0 aktuell, 0 übersprungen *****
FEHLERLISTE AUSGABE SUCHERGEBNISSE:1
Bereit Z8 S1 Zei1 EINGF

```

Die eben erzeugte Dynamic Link Library *Sortier.dll* belassen wir vorläufig im *\Debug-* (bzw. *Release-*) Unterverzeichnis des Projektordners. Später können wir sie in das Windows-System32-Verzeichnis oder, wie in unserem Fall, in dasselbe Verzeichnis wie die noch zu erstellende VB-Anwendung kopieren.

Einbinden der DLL in VB

Schließen Sie Visual Studio und öffnen Sie es erneut. Erstellen Sie eine neue Windows Forms-Anwendung mit dem Namen *Test_DLL* und platzieren Sie auf dem Startformular *Form1* je zwei *ListBox*en und zwei *Buttons* (siehe spätere Laufzeitanzeige).

Unbedingt erforderlich ist das Einbinden des folgenden Namespace:

```
Imports System.Runtime.InteropServices
```

```
Public Class Form1
```

Die Einbindung der externen DLL und die Deklaration der Routine *Sortiere*:

```
<DllImport("Sortier.dll")>
Private Shared Sub Sortiere(a As Integer(), n As Integer)
End Sub
```

Für unsere Testzwecke verwenden wir ein Integer-Array mit 100 Feldern:

```
Private Const len As Integer = 100
Private intArray() As Integer
```

Über die Schaltfläche *NEU* wird das Array mit Zufallszahlen gefüllt:

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    ReDim intArray(len)
    ListBox1.Items.Clear()
    Dim rnd As New Random()
    For i As Integer = 0 To len - 1
        intArray(i) = rnd.Next(0, len - 1)
    End For
End Sub
```

```
Next i
```

Anzeige des unsortierten Array-Inhalts in der ersten *ListBox*:

```
For i As Integer = 0 To len - 1
    ListBox1.Items.Add(intArray(i).ToString())
Next i
End Sub
```

Die Schaltfläche *SORTIER*:

```
Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
    ListBox2.Items.Clear()

```

Nun der alles entscheidende DLL-Aufruf:

```
Sortiere(intArray, intArray.Length)
```

Anzeige des sortierten Array-Inhalts in der zweiten *ListBox*:

```
For i As Integer = 0 To len - 1
    ListBox2.Items.Add(intArray(i).ToString())
Next i
End Sub
End Class
```

DLL-Test

- Kompilieren Sie das Projekt *Test_DLL* und kopieren Sie die Datei *Sortier.dll* in das *\Debug-*Unterverzeichnis des Projektordners *Test_DLL*, bzw. in das gleiche Verzeichnis wie auch die Datei *Test_DLL.exe*.
- Durch Klick auf die Schaltfläche *NEU* füllt sich die linke *ListBox* mit Zufallszahlen. Nachdem Sie aber auf die Schaltfläche *SORTIER* klicken, wird Ihr Tatendrang höchstwahrscheinlich durch die Meldung *Loader Lock wurde erkannt*. gestoppt. Wie Sie diese Meldung (die in der Regel nicht auf einen echten Fehler hinweist, sondern eine reine Vorsichtsmaßnahme der IDE ist) umgehen können, erläutern wir unter den Bemerkungen am Schluss.
- Für unser einfaches Beispiel kommen Sie aber wahrscheinlich viel eher zum Erfolg, wenn Sie Visual Studio verlassen und die Datei *Test_DLL.exe* direkt durch Doppelklick öffnen. Nach dem Füllen der linken *ListBox* und Klick auf die Schaltfläche *SORTIER* dürfte zunächst die folgende Meldung erscheinen:



Der Inhalt der rechten *ListBox* überzeugt Sie nun (hoffentlich) vom korrekten Funktionieren Ihrer DLL:



Ein erneuter Klick auf SORTIER führt nicht zu einer erneuten Anzeige obiger Informationsmeldung, da die DLL noch geladen ist. Erst nach dem Beenden der Anwendung taucht unerwartet ein weiterer Dialog auf:



Bemerkungen

- Damit Sie Ihren Programmcode unter Visual Studio debuggen können, gibt es die Möglichkeit, das Anhalten des Debuggers beim Ereignis *LoaderLock* zu deaktivieren. Gehen Sie dazu über das Menü *Debuggen/Ausnahmen...* zum Knoten *Managed Debugging Assistants*. Im Dialog *Ausnahmen* entfernen Sie das Häkchen bei *LoaderLock* in der Spalte *Ausgelöst*.



- Achten Sie bei der DLL-Einbindung immer auf die korrekten Datentypen und sichern Sie Ihr Projekt. Auf mögliche Probleme bei der Übergabe diverser Datentypen können wir hier aus Platzgründen leider nicht näher eingehen.
- Ein DLL-Name sollte eindeutig kennzeichnen, ob es sich um eine 32-Bit- oder eine 64-Bit-Variante handelt. Hängen Sie dazu an den Namen "32" oder "64" an. Dem Namen sollte weiterhin eine Beziehung zu Ihrem Produkt oder Ihrer Firmenbezeichnung zu entnehmen sein.

```
mstool32.dll  
vb40032.dll
```

- Möglichen Konflikten mit verschiedenen DLL-Versionen sollten Sie mit einer einfachen Versionskontrolle vorbeugen. Ihr VB-Programm muss als Erstes prüfen, ob die DLL-Version korrekt ist. Dies könnte zum Beispiel im *Load*-Event des Startformulars geschehen. Damit lässt sich zum einen immer kontrollieren, ob auch die aktuelle Version der DLL verwendet wird, zum anderen wird mit diesem Aufruf die DLL in den Speicher geladen, die Platte "rasselt" also nicht erst später beim eigentlichen Funktionsaufruf los.

```
int _stdcall version (void)  
{ return 4711; }
```

- Während der Testphase ist es empfehlenswert, sich die Versionsnummer der DLL anzeigen zu lassen. Erweitern Sie dazu die Eintrittsprozedur wie folgt:

```
BOOL WINAPI DllMain (HINSTANCE hDLL, DWORD dwReason, LPVOID lpReserved)  
{  
    switch (dwReason)  
    {  
        case DLL_PROCESS_ATTACH:  
            MessageBoxA ( GetFocus(),(LPCSTR) "Build 1.241",  
                (LPCSTR) "Info", MB_OK | MB_SYSTEMMODAL);  
            ...  
    }  
}
```

- Obiges Rezept will nicht mehr als nur einen kleinen Einblick in die Programmierung von C++-DLLs und deren Einbindung in ein VB-Programm vermitteln, wobei viele heikle Aspekte ausgeklammert wurden¹.

¹ Die Autoren sind sich durchaus im Klaren darüber, dass sie damit die "Büchse der Pandora" öffnen.