

# 6 Der Umstieg von Visual Basic.NET 2002 und 2003

---

185 Neue Sprachelemente in Visual Basic 2005

190 Übersicht über weitere Neuerungen in Visual Basic 2005

---

## Neue Sprachelemente in Visual Basic 2005

Zwar hat sich nicht viel, aber immerhin ein wenig an der Syntax der Basic-Sprache in der VB2005-Version geändert. Das betrifft einerseits eine Modifikation in der Behandlung von Schleifen, andererseits die gezielte Steuerung der Freigabe des Objektspeicherbedarfs sowie einige andere Punkte.

### Continue in Schleifen

Continue erlaubt ein vorzeitiges Wiederholen (nicht Beenden) einer Schleife. Für ein solches Konstrukt mussten Sie sich bislang entweder einer Hilfsvariablen oder dem (verpönten) GoTo-Befehl bedienen. Der folgende Code zeigt zwei Schleifenbeispiele, die exakt dasselbe machen und die auch dieselbe Performance aufweisen – einmal jedoch mit Goto und ein anderes Mal mit Continue arbeiten.

```
Public Class Form1
```

```
    Private Sub btnContinue_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles  
        btnContinue.Click
```

```
        'For/Next-Schleife - Variante 1 - Mit Goto  
        'Zahlen ausgeben, aber bis 10 nur gerade, ab 10 nur ungerade  
        For c As Integer = 0 To 20  
            If c < 10 Then  
                If (c \ 2) * 2 <> c Then  
                    GoTo SkipLoop  
                End If  
                Debug.Print("Gerade: " & c)  
            End If  
  
            If c > 10 Then  
                If (c \ 2) * 2 = c Then  
                    GoTo SkipLoop  
                End If  
                Debug.Print("Ungerade: " & c)  
            End If
```

### SkipLoop:

```
Next

'For/Next-Schleife - Variante 2 - Mit Continue For
'Zahlen ausgeben, aber bis 10 nur gerade, ab 10 nur ungerade
For c As Integer = 0 To 20
    If c < 10 Then
        If (c \ 2) * 2 <> c Then
            Continue For
        End If
        Debug.Print("Gerade: " & c)
    End If

    If c > 10 Then
        If (c \ 2) * 2 = c Then
            Continue For
        End If
        Debug.Print("Ungerade: " & c)
    End If
Next
End Sub
End Class
```

---

**HINWEIS:** Continue können Sie auch in Do- bzw. While-Schleifen einsetzen. Auch in diesem Fall bewirkt Continue einen Sprung zurück zum Schleifenanfang. Bei geschachtelten Schleifen desselben Typs, z.B. einer Do-Schleife in einer weiteren Do-Schleife, springt eine Continue Do-Anweisung zum Beginn der inneren Do-Schleife. Sie können hier hingegen nicht mit Continue zur äußeren Schleife desselben Typs springen. Bei geschachtelten Schleifen von unterschiedlichem Typ, z.B. einer Do-Schleife in einer For-Schleife, können Sie mit Continue Do bzw. Continue For gezielt zur nächsten Iteration einer der beiden Schleifenkonstrukte springen.

---

## Gezieltes Freigeben von Objekten mit Using

Normalerweise sorgt die in .NET eingebaute »Müllabfuhr« (der Garbage Collector) dafür, dass Objekte, die Sie verwendet haben, die aber nicht mehr benötigt werden, speichertechnisch entsorgt werden. Sie müssen sich also nicht selber darum kümmern, Speicher freizugeben, wenn Sie ein Objekt nicht mehr benötigen.

Wenn Sie ein Objekt aus einer Klasse mit New instanzieren, reservieren Sie Speicher in einem bestimmten Teil des Hauptspeichers – im so genannten Managed Heap – in dem es seine Daten ablegt. Wird das Objekt nicht mehr benötigt, und das ist beispielsweise dann der Fall, wenn es innerhalb einer Prozedur deklariert wird, die Prozedur dann aber verlassen wird, sorgt der Garbage Collector irgendwann dafür, dass der Speicher wieder freigegeben wird.

In einigen Fällen kann es aber erforderlich sein, ein Objekt gezielt zur Freigabe zu markieren, dem Framework also mitzuteilen: »Dieses Objekt benötige ich nicht mehr, du kannst es beim nächsten Mal, wenn du zur Mülltonne gehst, bitte mitnehmen.« Dieser Fall tritt dann ein, wenn nicht nur Speicher für das Objekt auf dem Managed Heap (also im Arbeitsspeicher) für das Speichern seiner Daten reserviert ist, sondern wenn auch noch andere Ressourcen des Betriebssystems reserviert werden müssen – zum Beispiel Dateikanäle beim Lesen aus oder Schreiben in Dateien. In diesem Fall ergibt es Sinn, dem Objekt mitzuteilen: »Ich brauch dich jetzt nicht mehr, bitte gib alle Systemressourcen,

die du vielleicht belegt hast, wieder frei. Und wo wir schon mal dabei sind: Teile dem Müllmann mit, wenn er das nächste Mal vorbeikommt, dass er dich gleich mitnehmen soll.« Das klingt zwar herzlos, ist aber pragmatisch.

Objekte, die es gestatten, sich selbst auf die Entsorgung hinzuweisen, implementieren eine Methode namens `Dispose` (etwa: *entsorgen*). Rufen Sie `Dispose` eines Objektes auf, gibt es alle belegten Systemressourcen frei und signalisiert dem Garbage Collector gleichzeitig, dass er es »mitnehmen« kann. Eine typische Vorgehensweise zur Anwendung sieht dann so aus, wie es das folgende Beispiellisting demonstriert, das eine Textdatei schreibt:

```
Dim locSw As StreamWriter
Try
    locSw = New StreamWriter("C:\Textdatei1.txt")
    Try
        locSw.WriteLine("Erste Textzeile")
        locSw.WriteLine("Zweite Textzeile")
        'Schreibpuffer leeren
        locSw.Flush()
    Catch ex As Exception
        Debug.Print("Fehler beim Schreiben der Datei!")
    Finally
        'Alle Systemressourcen wieder freigeben
        locSw.Dispose()
    End Try
Catch ex As Exception
    Debug.Print("Fehler beim Öffnen der Datei!")
End Try
```

Weil das Freigeben der belegten Systemressourcen in diesem Fall so wichtig ist, befindet sich der Aufruf der `Dispose`-Methode in diesem Fall im `Finally`-Teil des `Try/Catch`-Blocks. Dadurch wird sichergestellt, dass die Systemressourcen des Objektes in jedem Fall freigegeben werden, egal ob es innerhalb des `Try`-Teil zu einer Ausnahme kam oder nicht.

---

**HINWEIS:** Zwei geschachtelte `Try`-Blöcke sind hier übrigens notwendig, weil beim Öffnen der Datei andere Ausnahmen auftreten können, als beim Schreiben. Die Systemressourcen selbst werden aber nur nach *erfolgreichem* Öffnen des `StreamWriter`-Objektes von diesem belegt, und müssen deswegen auch nur in diesem Fall (innerer `Try/Catch`-Block) mit `Dispose` wieder freigegeben werden.

---

`Using` erlaubt nun, die Lebensdauer eines Objektes, das die `Dispose`-Methodik implementiert, gezielt zu steuern. Das folgende Beispiellisting entspricht dem obigen Beispiel im Detail – `Using` vereinfacht aber den Umgang mit dem Objekt, und der Code wird leichter lesbar:

```
'Schreiben einer Datei - mit Using wird die Lebensdauer von locSw2 gesteuert
Try
    'Alternativ ginge auch die Weiterverwendung von locSw:
    'locSw = New StreamWriter("C:\Textdatei2.txt")
    'Using locSw
    Using locSw2 As New StreamWriter("C:\Textdatei2.txt")
        Try
            locSw2.WriteLine("Erste Textzeile")
            locSw2.WriteLine("Zweite Textzeile")
            'Schreibpuffer leeren
```

```

        locSw2.Flush()
    Catch ex As Exception
        Debug.Print("Fehler beim Schreiben der Datei!")
    End Try
    'Hier wird automatisch locSw.Dispose durchgeführt
End Using
Catch ex As Exception
    Debug.Print("Fehler beim Öffnen der Datei!")
End Try

```

Sie sehen hier, dass der `Finally`-Teil des inneren `Try/Catch`-Blocks überflüssig geworden ist – denn am `End Using` kommt das Programm nicht vorbei. Selbst, wenn es versuchte, beispielsweise mit `Return` aus dem `Using`-Block herauszuspringen, würde die `Dispose`-Methode des Objektes dennoch vor dem eigentlichen Verlassen der Prozedur noch ausgeführt.

Mehr zum Thema `Dispose` und dem `Garbage Collector` erfahren Sie übrigens in ► Kapitel 12.

---

**TIPP:** `Using` wird besonders häufig bei Verbindungen zum `SQL Server` eingesetzt. Sobald Sie eine Verbindung zum `SQL Server` benötigen, verwenden Sie das dazu benötigte `Connection`-Objekt in einer `Using`-Anweisung. Führen Sie alle Operationen durch, für die Sie eine offene Verbindung benötigen. Am Ende aller Operationen schließen Sie die `SQL Server`-Verbindung mit `End Using`. Auf diese Weise können Sie auch gefahrlos einen solchen Codeblock verlassen, ohne explizit dafür Sorge getragen haben zu müssen, die Verbindung zum `SQL Server` wieder zu schließen. Ihre Anwendung wird dadurch robuster, denn Sie vermeiden doppelt oder mehrfach offene Verbindungen, weil Sie es etwa schlicht vergessen haben, eine offene Verbindung wieder zu schließen, wenn Sie einen Codeblock, in dem `SQL`-Operationen verarbeitet wurden, vorzeitig verlassen haben.

---

## Zugriff auf den Framework-System-Namespace mit `Global`

Das Schlüsselwort `Global` gestattet Ihnen den Zugriff auf ein `.NET Framework`-Programmierelement auch dann, wenn Sie es mit einer `Namespace`-Struktur blockiert haben.

Wenn Sie eine geschachtelte Hierarchie aus `Namespaces` definiert haben, kann Code innerhalb dieser Hierarchie möglicherweise nicht auf den `System`-Namespace von `.NET Framework` zugreifen. Im folgenden Beispiel wird eine Hierarchie gezeigt, in der der `SpecialSpace`-`System`-Namespace den Zugriff auf `System` blockiert, da er durch `System` im voll qualifizierten `Namespace`-Namen einen Namenskonflikt verursacht:

```

Namespace SpecialSpace
    Namespace System
        Class abc
            Function getValue() As System.Int32
                Dim n As System.Int32
                Return n
            End Function
        End Class
    End Namespace
End Namespace

```

Das führt dazu, dass der `Visual Basic`-Compiler den Verweis auf `System.Int32` nicht auflösen kann, da `Int32` durch `SpecialSpace.System` nicht definiert wird. Sie können das `Global`-Schlüsselwort verwenden, um die Qualifikationskette an der obersten Ebene der `.NET Framework`-Klassenbibliothek zu begin-

nen. Dies ermöglicht es Ihnen, den System-Namespace oder irgendeinen anderen Namespace in der Klassenbibliothek anzugeben. Dies wird anhand des folgenden Beispiels veranschaulicht:

```
Namespace SpecialSpace
  Namespace System
    Class abc
      Function GetValue() As Global.System.Int32
        Dim n As Global.System.Int32
        Return n
      End Function
    End Class
  End Namespace
End Namespace
```

Sie können `Global` verwenden, um auf andere Namespaces auf Stammebene zuzugreifen, z.B. `Microsoft.VisualBasic`, sowie auf jeden anderen Namespace, der dem Projekt zugeordnet ist.

Das `Global`-Schlüsselwort kann in den folgenden Kontexten verwendet werden:

- Class-Anweisung
- Const-Anweisung
- Declare-Anweisung
- Delegate-Anweisung
- Dim-Anweisung
- Enum-Anweisung
- Event-Anweisung
- For...Next-Anweisung
- For Each...Next-Anweisung
- Function-Anweisung
- Interface-Anweisung
- Operator-Anweisung
- Property-Anweisung
- Structure-Anweisung
- Sub-Anweisung
- Try...Catch...Finally-Anweisung
- Using-Anweisung

## Über mehrere Codedateien aufgeteilter Klassencode – Partial Class

Sie können die Definition einer Klasse oder Struktur mithilfe des `Partial`-Schlüsselworts auf mehrere Deklarationen aufteilen. Das bedeutet: Sie können beliebig viele Teildeklarationen einer Klasse in beliebig vielen unterschiedlichen Quelldateien verwenden. Alle Deklarationen müssen jedoch in der gleichen Assembly und dem gleichen Namespace enthalten sein.

---

**HINWEIS:** Visual Basic verwendet partielle Klassendefinitionen, um in jeweils eigenen Quelldateien generierten Code von Code zu trennen, der vom Benutzer erstellt wurde. Zum Beispiel definiert der Windows Form-Designer partielle Klassen für Steuerelemente, z.B. Form. Sie sollten den generierten Code in diesen Steuerelementen nicht ändern.

---

Normalerweise wird die Entwicklung einer einzelnen Klasse oder Struktur nicht auf zwei oder mehr Deklarationen aufgeteilt. In der Regel benötigen Sie das `Partial`-Schlüsselwort daher nicht. Sinn ergibt die Anwendung von `Partial` aber dann, wenn der Code einer Klasse dermaßen anwächst, dass die Übersicht sehr darunter leiden würde. Eine gute Vorgehensweise besteht dann darin, innerhalb der Projektmappe für die mit `Partial` aufgeteilte Klasse einen zusätzlichen Ordner anzulegen, und die einzelnen Codedateien der Klasse dort abzulegen.

Beim Erstellen einer partiellen Klasse oder Struktur gelten übrigens alle Regeln für die Erstellung von Klassen und Strukturen, beispielsweise diejenigen für die Verwendung und Vererbung von Modifizierern.

Übrigens: Nur die zusätzlichen Codedateien benötigen das `Partial`-Schlüsselwort, um den Klassencode zu erweitern – beim Ausgangsklassencode *kann* man es angeben, muss dies aber nicht tun.

Ein Beispiel für die Anwendung von `Partial` finden Sie in ► Kapitel 21 beim Beispiel des Formel Parsers.

## Übersicht über weitere Neuerungen in Visual Basic 2005

Weitere Neuerungen, die in Visual Basic 2005 implementiert wurden, würden den Rahmen an dieser Stelle sprengen und auch zu unnötigen Redundanzen innerhalb dieses Buches führen. Aus diesem Grund finden Sie im Folgenden eine Tabelle, die die wichtigsten Neuerungen zusammenfasst und Ihnen einen Verweis innerhalb dieses Buches nennt, der zu einem entsprechenden Kapitel oder Abschnitt führt, der diese Neuerungen genauer unter die Lupe nimmt.

Thema	Kurzbeschreibung	Mehr Info siehe
Eigenschaften von Klassen und Strukturen	Eigenschaften können so definiert werden, dass der Lesezugriff mit anderen Zugriffsmodifizierern erfolgt als der Schreibzugriff.	► Kapitel 8 »Unterschiedliche Zugriffsmodifizierer für Eigenschaften-Accessors«.
Aufbau und Umgang mit Formularen bei Windows Forms-Anwendungen	Anders als bei Visual Basic .NET, wird der Designer-Code und die eigentliche Angabe der Vererbung von Form in einer separaten Codedatei abgelegt.	► Kapitel 28 hält genauere Informationen zu diesem Thema bereit. ► Kapitel 3 zeigt den Umgang mit den Neuheiten im Formular-Designer am Beispiel.
Operatorenprozeduren	Klassen bzw. Strukturen können eigene Operatoren implementieren. Dadurch wird der Umgang mit ihnen wesentlich vereinfacht.	► Kapitel 13 informiert Sie über dieses Thema genauer. ►

Thema	Kurzbeschreibung	Mehr Info siehe
Generische Datentypen (Generics)	Generische Datentypen dienen zur allgemein gültigen Formulierung typsicherer Codes. Sie vereinen damit die Flexibilität von Object mit Typsicherheit.	► Kapitel 14 hält Genauer zu diesem Thema bereit.
Primitive Datentypen	Mit Visual Basic 2005 gibt es eine Reihe neuer primitiver Datentypen.	Einige dieser Datentypen wurden bereits in ► Kapitel 5 angesprochen. Genauer über die neuen Typen erfahren Sie in ► Kapitel 16.
Primitive Typen, die »nullbar« sind (Nullable)	Dank des Konzeptes der generischen Datentypen gibt es neu in .NET 2.0 die so genannten »Nullables«. Dabei handelt es sich um primitive Datentypen, die über ihre Werttypen hinaus auch Nothing speichern können, was beispielsweise der Datenbankprogrammierung zugute kommt.	Mehr zu diesem Thema finden Sie in ► Kapitel 20.
Generische Auflistungsklassen	Generische Auflistungsklassen dienen zur einfachen Typisierung von Auflistungen, stellen neue Funktionalitäten für Auflistungen zur Verfügung, sorgen für robusteren und teils auch schnelleren Code.	Zu diesem Thema finden Sie in ► Kapitel 20 Genaueres.
My-Namespace	Mit dem My-Namespace haben es gerade VB6-Entwickler einfacher, sich in der .NET-Welt zurecht zu finden. Der My-Namespace stellt quasi einen Schnellzugriff für Entwickler auf häufig verwendete Funktionen dar, stellt teilweise sogar neue Funktionalitäten bereit.	In ► Kapitel 25 finden Sie detaillierte Beschreibungen und eine umfangreiche Beispielanwendung, die den Umgang mit My verdeutlicht.
Anwendungsframework	Das Anwendungsframework dient in VB2005 für das einfache Handling von Windows Forms-Anwendungen aus Entwicklersicht.	► Kapitel 26 klärt Details zum Anwendungsframework.
BackgroundWorker-Komponente	In Sachen Threading hat sich nicht nur durch die Entwicklung neuer Mehrkern-Prozessoren einiges getan. Die BackgroundWorker-Komponente bietet den einfachsten und komplikationslosesten Einstieg in das Multithreading.	► Kapitel 31 beschreibt den Umgang mit der neuen Komponente. <b>TIPP:</b> Sie sollten sich dennoch das komplette Kapitel zu Gemüte führen, auch wenn es zunächst für die Anwendung dieser Komponente nicht nötig erscheint.
ADO.NET 2.0; SQL Server 2005	Viele neue Features haben Einzug in ADO.NET gehalten, und nicht zuletzt durch SQL Server 2005 Express werden Visual Basic-Anwendungen, die auf SQL Server basieren, ältere Access-basierende Anwendungen zunehmend ablösen.	► Kapitel 32 beschreibt nicht nur die Konfiguration von SQL Server 2005 Express, sondern führt auch in die Programmierung der ADO.NET 2.0-Klassen ein.

**Tabelle 6.1:** Neues in Visual Basic 2005 – hier finden Sie die Verweise auf die Kapitel im Buch

