

VB.net und ADO.net

5. AEK 28. - 29. Sept. 2002

Referent: Hendrik Lindemann

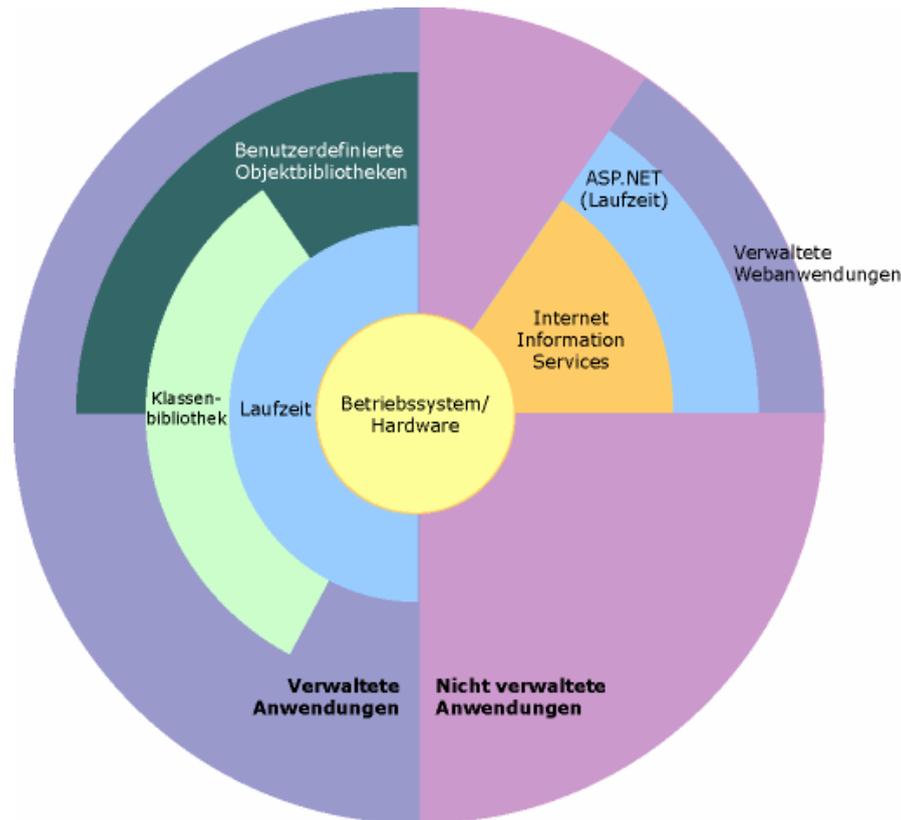
Überblick über das .net Framework

- Einführung in das .net Framework

Microsoft
.net Framework

Überblick über das .net Framework

■ .NET Framework im Kontext



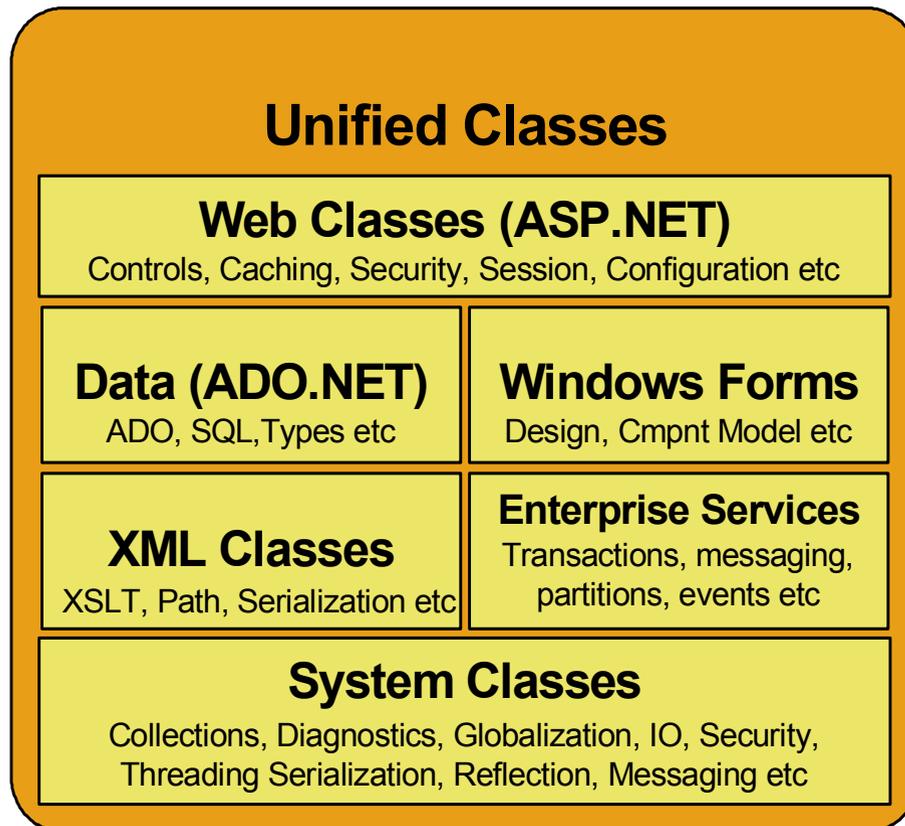
Überblick über das .net Framework

■ Common Language Runtime (CLR)



Überblick über das .net Framework

■ Vereinheitlichte Programmierklassen



Überblick über das .net Framework

■ Entwicklung von Clientanwendungen



Windows-Anwendung



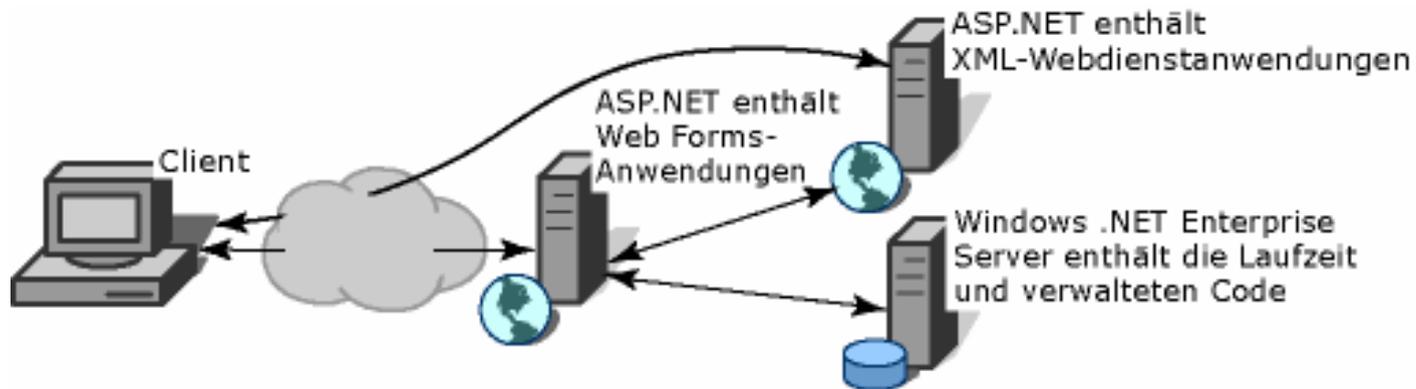
Konsolenanwendung



Windows-Steuerele...

Überblick über das .net Framework

■ Entwicklung von Serveranwendungen



Unterschiede zwischen VB.NET und VBA in der Praxis

■ Der neue Projekt- und Programmaufbau

```
'Einfügen von Namespaces in den Gültigkeitsbereich mit Imports
Imports System.Math

Module AreaCalculator

    Sub Main()

        'Auffangen von Fehlern mit der strukturierten Ausnahmebehandlung
        Try
            Dim myTriangle As New Triangle(2, 3, 4)
            'Ausgabe von Meldungen mit Platzhalter {0-n}
            Console.WriteLine("Berechne die Fläche eines Triangels mit den Seiten {0},{1},{2}", myTriangle.a,
                Console.WriteLine("Die Fläche des Triangels beträgt: {0}", myTriangle.ComputeArea())
            Console.WriteLine("Berechne die Fläche eines Kreises mit dem Radius {0}", 2)
            Console.WriteLine("Die Fläche des Kreises beträgt: {0}", Circle.computeArea(2))
        Catch ex As Exception
            Console.WriteLine(ex.Message)
        Finally
            Console.WriteLine("Fertig! Drücken Sie eine beliebige Taste um die Anwendung zu beenden.")
            If Console.ReadLine() <> "" Then
                End
            End If
        End Try
    End Sub
End Module

Class Triangle...

Class Circle...
```


Unterschiede zwischen VB.NET und VBA in der Praxis

■ Änderungen an Visual Basic / VBA Funktionen

Visual Basic 6.0-Programmierelement	Visual Basic .NET-Äquivalent	Position von Namespace, Klasse oder Laufzeitbibliothek
Abs-Funktion	Abs-Methode	Systemnamespace, Math-Klasse
AscB-Funktion	Asc-Funktion	Member der Visual Basic-Laufzeitbibliothek, Zeichenfolgen-Modul
As Any-Schlüsselwortausdruck	In Visual Basic .NET nicht unterstützt.	Nicht vorhanden
Atn-Funktion	Atan-Methode	Systemnamespace, Math-Klasse
Calendar-Eigenschaft	CurrentCulture-Eigenschaft	System.Globalization-Namespace, CultureInfo-Klasse
ChDir-Anweisung	ChDir-Funktion	Member der Visual Basic-Laufzeitbibliothek, FileSystem-Modul
ChDrive-Anweisung	ChDrive-Funktion	Member der Visual Basic-Laufzeitbibliothek, FileSystem-Modul
Chr\$-Funktion, ChrB-Funktion	Chr-Funktion	Member der Visual Basic-Laufzeitbibliothek, Zeichenfolgen-Modul
Close-Anweisung	FileClose-Funktion	Member der Visual Basic-Laufzeitbibliothek, FileSystem-Modul
Cos-Funktion	Cos-Methode	Systemnamespace, Math-Klasse
Currency-Datentyp	Decimal-Datentyp;	Member der Visual Basic-Laufzeitbibliothek, , VariantType-Enumeration
CVDate-Funktion	DateValue-Funktion	Member der Visual Basic-Laufzeitbibliothek, DateAndTime-Modul
CVError-Funktion	Error-Anweisung	Nicht vorhanden
Date-Funktion, Date-Anweisung	Now-Eigenschaft, Today-Eigenschaft	Member der Visual Basic-Laufzeitbibliothek, DateAndTime-Modul

Unterschiede zwischen VB.NET und VBA in der Praxis

■ Änderungen an Visual Basic / VBA Funktionen

Visual Basic 6.0-Programmierelement	Visual Basic .NET-Äquivalent	Position von Namespace, Klasse oder Laufzeitbibliothek
Date\$-Funktion	DateString-Eigenschaft	Member der Visual Basic-Laufzeitbibliothek, DateAndTime-Modul
Debug.Assert-Methode	Methoden Assert, Fail	System.Diagnostics-Namespace, Debug-Klasse
Debug.Print-Methode	Methoden Write, Writelf, WriteLine und WriteLinelf	System.Diagnostics-Namespace, Debug-Klasse
Deftype-Anweisungen	In Visual Basic .NET nicht unterstützt.	Nicht vorhanden
DeleteSetting-Anweisung	DeleteSetting-Funktion	Member der Visual Basic-Laufzeitbibliothek, Interaction-Modul
DoEvents-Funktion	DoEvents-Methode	System.Windows.Forms-Namespace, Application-Klasse
Empty-Schlüsselwort	Nothing	Nicht vorhanden
Eqv-Operator	Operator =;	Nicht vorhanden
Exp-Funktion	Exp-Methode	Systemnamespace, Math-Klasse
FileCopy-Anweisung	FileCopy-Funktion	Member der Visual Basic-Laufzeitbibliothek, FileSystem-Modul
Get-Anweisung	FileGet-Funktion	Member der Visual Basic-Laufzeitbibliothek, FileSystem-Modul
GoSub-Anweisung	In Visual Basic .NET nicht unterstützt; verwenden Sie die Return-Anweisung.	Nicht vorhanden
Initialize-Ereignis	In Visual Basic .NET nicht unterstützt; verwenden Sie Sub New.	Nicht vorhanden
Imp-Operator	In Visual Basic .NET nicht unterstützt.	Nicht vorhanden

Unterschiede zwischen VB.NET und VBA in der Praxis

■ Änderungen an Visual Basic / VBA Funktionen

Visual Basic 6.0- Programmierelement	Visual Basic .NET- Äquivalent	Position von Namespace, Klasse oder Laufzeitbibliothek
Anweisungen Input #, Input\$, Funktionen Input\$, InputB, InputB\$	Input-Funktion	Member der Visual Basic-Laufzeitbibliothek, FileSystem-Modul
Instancing-Eigenschaft	In Visual Basic .NET nicht unterstützt.	Nicht vorhanden
InStrB-Funktion	InStr-Funktion	Member der Visual Basic-Laufzeitbibliothek, Zeichenfolgen-Modul
IsEmpty-Funktion	IsNothing-Funktion	Member der Visual Basic-Laufzeitbibliothek, Information-Modul
IsMissing-Funktion	In Visual Basic .NET nicht unterstützt.	Nicht vorhanden
IsNull-Funktion	IsDBNull-Funktion	Member der Visual Basic-Laufzeitbibliothek, Information-Modul
IsObject-Funktion	IsReference-Funktion	Member der Visual Basic-Laufzeitbibliothek, Information-Modul
Kill-Anweisung	Kill-Funktion	Member der Visual Basic-Laufzeitbibliothek, FileSystem-Modul
LCase\$-Funktion	LCase-Funktion	Member der Visual Basic-Laufzeitbibliothek, Zeichenfolgen-Modul
Funktionen Left\$, LeftB, LeftB\$	Left-Funktion	Member der Visual Basic-Laufzeitbibliothek, Zeichenfolgen-Modul
LenB-Funktion	Len-Funktion	Member der Visual Basic-Laufzeitbibliothek, Zeichenfolgen-Modul
Zuweisungsanweisungen Let, Set	In Visual Basic .NET nicht unterstützt; die neue Set-Anweisung hat keinen Bezug zur älteren Anweisung.	Nicht vorhanden
Line Input #-Anweisung	LineInput-Funktion	Member der Visual Basic-Laufzeitbibliothek, FileSystem-Modul
Lock-Anweisung	Die Funktionen "Lock" und "Unlock"	Member der Visual Basic-Laufzeitbibliothek, FileSystem-Modul

Unterschiede zwischen VB.NET und VBA in der Praxis

■ Änderungen an Visual Basic / VBA Funktionen

Visual Basic 6.0- Programmierelement	Visual Basic .NET- Äquivalent	Position von Namespace, Klasse oder Laufzeitbibliothek
Log-Funktion	Log-Methode	Systemnamespace, Math-Klasse
Anweisungen LSet, RSet	LSet-Funktion, PadRight, PadLeft;	Member der Visual Basic-Laufzeitbibliothek, Zeichenfolgen-Modul und Systemnamespace, String-Klasse
LTrim\$-Funktion	Ltrim-Funktion	Member der Visual Basic-Laufzeitbibliothek, Zeichenfolgen-Modul
MidB-Funktion	Mid-Funktion	Member der Visual Basic-Laufzeitbibliothek, Zeichenfolgen-Modul
MidB-Anweisung	Mid-Anweisung	Member der Visual Basic-Laufzeitbibliothek, Zeichenfolgen-Modul
MkDir-Anweisung	MkDir-Funktion	Member der Visual Basic-Laufzeitbibliothek, FileSystem-Modul
Name-Anweisung	Rename-Funktion	Member der Visual Basic-Laufzeitbibliothek, FileSystem-Modul
Now-Funktion	Now-Eigenschaft	Member der Visual Basic-Laufzeitbibliothek, DateAndTime-Modul
Null-Schlüsselwort	Nothing	Nicht vorhanden
Oct\$-Funktion	Oct-Funktion	Member der Visual Basic-Laufzeitbibliothek, Konvertierungen-Modul
On ... GoSub-Konstruktion	In Visual Basic .NET nicht unterstützt; verwenden Sie die Select Case-Anweisung.	Nicht vorhanden
On ... GoTo construction	Not supported in Visual Basic .NET; use Select Case Statement.	Nicht vorhanden
Open-Anweisung	FileOpen-Funktion	Member der Visual Basic-Laufzeitbibliothek, FileSystem-Modul
Option Base-Anweisung	In Visual Basic .NET nicht unterstützt.	Nicht vorhanden
Option Private Module-Anweisung	In Visual Basic .NET nicht unterstützt; verwenden Sie die Module-Anweisung.	Nicht vorhanden

Unterschiede zwischen VB.NET und VBA in der Praxis

■ Änderungen an Visual Basic / VBA Funktionen

Visual Basic 6.0- Programmirelement	Visual Basic .NET- Äquivalent	Position von Namespace, Klasse oder Laufzeitbibliothek
Print #-Anweisung	Die Funktionen "Print" und "PrintLine"	Member der Visual Basic-Laufzeitbibliothek, FileSystem-Modul
Anweisungen Property Get, Property Let, Property Set	In Visual Basic .NET nicht unterstützt.	Nicht vorhanden
Put-Anweisung	FilePut-Funktion	Member der Visual Basic-Laufzeitbibliothek, FileSystem-Modul
Reset-Anweisung	Reset-Funktion	Member der Visual Basic-Laufzeitbibliothek, FileSystem-Modul
Funktionen Right\$, RightB	Right-Funktion	Member der Visual Basic-Laufzeitbibliothek, Zeichenfolgen-Modul
Rmdir-Anweisung	Rmdir-Funktion	Member der Visual Basic-Laufzeitbibliothek, FileSystem-Modul
Round-Funktion	Round-Methode	Systemnamespace, Math-Klasse
Anweisungen RSet, LSet	Rset-Funktion	Member der Visual Basic-Laufzeitbibliothek, Zeichenfolgen-Modul
RTrim\$-Funktion	Rtrim-Funktion	Member der Visual Basic-Laufzeitbibliothek, Zeichenfolgen-Modul
SaveSetting-Anweisung	SaveSetting-Funktion	Member der Visual Basic-Laufzeitbibliothek, Interaction-Modul
Scale-Methode	In Visual Basic .NET nicht unterstützt	Nicht vorhanden
Zuweisungsanweisungen Set, Let	In Visual Basic .NET nicht unterstützt; die neue Set-Anweisung hat keinen Bezug zur älteren Anweisung	Nicht vorhanden
SetAttr-Anweisung	SetAttr-Funktion	Member der Visual Basic-Laufzeitbibliothek, FileSystem-Modul
Sgn-Funktion	Sign-Funktion	Systemnamespace, Math-Klasse
Sin-Funktion	Sin-Methode	Systemnamespace, Math-Klasse
Sqr-Funktion	Sqrt-Funktion	Systemnamespace, Math-Klasse

Unterschiede zwischen VB.NET und VBA in der Praxis

■ Änderungen an Visual Basic / VBA Funktionen

Visual Basic 6.0- Programmierelement	Visual Basic .NET- Äquivalent	Position von Namespace, Klasse oder Laufzeitbibliothek
String-Funktion	String-Konstruktor	Systemnamespace, String-Klasse
String (\$) -Funktionen	In Visual Basic .NET nicht unterstützt.	Nicht vorhanden
Terminate-Ereignis	In Visual Basic .NET nicht unterstützt; verwenden Sie Sub Dispose und Sub Finalize.	Nicht vorhanden
Time-Funktion, Time- Anweisung	TimeOfDay-Eigenschaft; siehe DateTime-Struktur, Date-Datentyp	Member der Visual Basic-Laufzeitbibliothek, DateAndTime-Modul
Time\$-Funktion	TimeString-Eigenschaft	Member der Visual Basic-Laufzeitbibliothek, DateAndTime-Modul
Timer-Funktion	Timer-Eigenschaft	Member der Visual Basic-Laufzeitbibliothek, FileSystem-Modul
Trim\$-Funktion	Die Funktionen "LTrim", "RTrim" und "Trim"	Member der Visual Basic-Laufzeitbibliothek, Zeichenfolgen-Modul
Type-Anweisung	In Visual Basic .NET nicht unterstützt; verwenden Sie die Structure-Anweisung.	Nicht vorhanden
UCase\$-Funktion	UCase-Funktion	Member der Visual Basic-Laufzeitbibliothek, Zeichenfolgen-Modul
Unlock-Anweisung	Die Funktionen "Lock" und "Unlock"	Member der Visual Basic-Laufzeitbibliothek, FileSystem-Modul
Variant-Datentyp	Object-Datentyp; siehe Änderungen am universellen Datentyp in Visual Basic	Nicht vorhanden
Wend-Schlüsselwort	While...End While-Anweisungen und End-Anweisung;	Nicht vorhanden
Width #-Anweisung	FileWidth-Funktion	Member der Visual Basic-Laufzeitbibliothek, FileSystem-Modul
Write #-Anweisung	Die Funktionen "Write" und "WriteLine"	Member der Visual Basic-Laufzeitbibliothek, FileSystem-Modul

Unterschiede zwischen VB.NET und VBA in der Praxis

- **Neues in VB.NET**
 - **Vererbung**
 - **Ausnahmebehandlung**
 - **Überladung**
 - **Überschreiben von Eigenschaften und Methoden**
 - **Konstruktoren und Destruktoren**
 - **Neue Datentypen**
 - **Schnittstellen**
 - **Delegaten**
 - **Freigegebene Member**
 - **Verweise**
 - **Namespaces**
 - **Assemblies**
 - **Attribute**
 - **Multithreading**

Unterschiede zwischen VB.NET und VBA in der Praxis

■ Der Imports-Befehl

Code-Beispiel

Variante a) Ohne den *Imports*-Befehl

```
Module Module1
    Sub Main()
        System.Console.WriteLine("Hallo, Welt - wie geht's?")
    End Sub
End Module
```

Da *WriteLine* im aktuellen Namensraum nicht bekannt ist, muss der komplette Klassenpfad aufgeführt werden.

Variante b) Mit *Imports*-Befehl

```
Imports System.Console
Module Module1
    Sub Main()
        WriteLine("Hallo, Welt - wie geht's?")
    End Sub
End Module
```

Unterschiede zwischen VB.NET und VBA in der Praxis

■ Der Block als Gültigkeitsbereich

Code-Beispiel

```
Imports System.Console
Module Module1
    Private nWert As Long
    Sub Main()
        nWert = 77
        WriteLine(nWert)
        Do
            Dim nWert As Long
            nWert = 99
            WriteLine(nWert)
            If nWert = 99 Then Exit Do
        Loop
    End Sub
End Module
```

Unterschiede zwischen VB.NET und VBA in der Praxis

■ Zuweisungen

Code Beispiel:

```
Imports System.Console
Module Module1
    Sub Main()
        Dim intI As Integer = 0
        Dim intMax As Integer = 99
        For intI = 0 To intMax
            'Erhöht den Wert der Variablen intI um 1
            intI += 1
            WriteLine(intI)
        Next
        ReadLine()
    End Sub
End Module
```

Unterschiede zwischen VB.NET und VBA in der Praxis

■ Variablen und Konstanten

- Die Datentypen basieren (zwangsläufig) auf dem Common Type System (CTS).
- Decimal ist ein eigenständiger Datentyp und kein Unterdatentyp von Variant mehr.
- Wird bei der Deklaration einer Variablen kein Datentyp angegeben, erhält sie den Datentyp Object
- Da es Variant nicht mehr gibt, gibt es auch die Spezialwerte Empty, Missing und Null nicht mehr.
- Variablen können mit ihrer Deklaration initialisiert werden. Das ist eine längst überfällige Erweiterung.
- Werden mehrere Variablen mit einem Befehl deklariert, gibt der Datentyp der letzten Variablen den Datentyp der übrigen Variablen vor
- Es gibt neue Zuweisungsoperatoren, die gleichzeitig eine Addition oder Subtraktion durchführen.
- Bei den logischen Operatoren wird zwischen logischen Verknüpfungen und bitweisen Verknüpfungen unterschieden.

Unterschiede zwischen VB.NET und VBA in der Praxis

■ Namensregeln

- An den Namensregeln für Variablen und Konstanten hat sich nichts geändert.
- Die Groß-/Kleinschreibung spielt (anders als in C#) nach wie vor keine Rolle.

Unterschiede zwischen VB.NET und VBA in der Praxis

■ Neue Datentypen

Code-Beispiel

```
Dim c As Char
```

```
    c = "x"
```

Richtig(ist)

```
    c = CChar("x")
```

Unterschiede zwischen VB.NET und VBA in der Praxis

- Geänderte Datentypen

Code-Beispiel

```
Dim nKlein As System.Int32
```

oder

```
Dim nGross As System.Int64
```

Unterschiede zwischen VB.NET und VBA in der Praxis

- Datentypen, die es nicht mehr gibt
- Kein Currency-Datentyp mehr
- Nachfolger ist Decimal

Unterschiede zwischen VB.NET und VBA in der Praxis

■ Die Deklaration von Variablen

Code-Beispiel

```
Dim nWert1, nWert2, nWert3 As Long
```

Code-Beispiel

```
Dim nWert1 As Long = 1234, nWert2 As Long = 456,  
    nWert3 As Long
```

Unterschiede zwischen VB.NET und VBA in der Praxis

■ Alles ist ein Objekt

Code-Beispiel

```
Dim i As Integer = 123
Dim j As System.Int32 = 123
Dim k As New Integer()
    k = 123
```

Code-Beispiel 2

```
Dim i, j As System.Int32
i = 4
j = 5
WriteLine("{0}+{1}={2}", i, j, i + j)
Dim s As Integer = i + j
WriteLine(4.ToString + "+" + 5.ToString + "=" + s.ToString)
WriteLine(4 + "+" + 5 + "=" + s.ToString)
```

Unterschiede zwischen VB.NET und VBA in der Praxis

■ Der neue Option Strict-Befehl

Code-Beispiel

```
Dim sWert As String
    sWert = 1234
Dim nWert As Long
    nWert = "1234,,
```

Code-Beispiel

```
Dim nWert As Long
    nWert = CLng("1234")
```

Code-Beispiel

```
Sub Main
    Dim i, j As Short
    j = 25000
    i = j * 2
    WriteLine("Alles ergibt einen Sinn: " & i)
End Sub
```

■ Der gute alte Option Explicit-Befehl ist geblieben – bloß wo?

Unterschiede zwischen VB.NET und VBA in der Praxis

■ Strings

```
Dim sAusgabe As String
```

Code-Beispiel

```
Dim sName As New String("Ein kleiner Text")
    WriteLine("Die Länge des Strings: {0}",
sName.Length)
    WriteLine("Die Position des k ist: {0}",
sName.IndexOf("k"))
    WriteLine("Der neue String lautet: {0}",
sName.Replace("e", "o"))
    WriteLine("Und jetzt alles klein: {0}",
sName.ToLower())
```

Unterschiede zwischen VB.NET und VBA in der Praxis

■ Strings

```
Dim sAusgabe As String
```

Code-Beispiel

```
Dim sName As New String("Ein kleiner Text")
    WriteLine("Die Länge des Strings: {0}",
sName.Length)
    WriteLine("Die Position des k ist: {0}",
sName.IndexOf("k"))
    WriteLine("Der neue String lautet: {0}",
sName.Replace("e", "o"))
    WriteLine("Und jetzt alles klein: {0}",
sName.ToLower())
```

Unterschiede zwischen VB.NET und VBA in der Praxis

■ Strings als unveränderbare Objekte

Code-Beispiel

```
Dim sString1, sString2 As String
sString1 = "Hallo, Welt"
sString2 = sString1
WriteLine("String1 = String2: " + (sString1 = sString2).ToString())
WriteLine("String1 Is String2: " + (sString1 Is sString2).ToString())
Mid(sString1, 8, 1) = "Z"
WriteLine("String1 = String2: " + (sString1 = sString2).ToString())
WriteLine("String1 Is String2: " + (sString1 Is sString2).ToString())
Mid(sString2, 8, 1) = "Z"
WriteLine("String1 = String2: " + (sString1 = sString2).ToString())
WriteLine("String1 Is String2: " + (sString1 Is sString2).ToString())
```

Wird das Programm gestartet, erscheint die folgende Ausgabe in der Konsole:

```
String1 = String2: True
String1 Is String2: True
String1 = String2: False
String1 Is String2: False
String1 = String2: True
String1 Is String2: False
```

Unterschiede zwischen VB.NET und VBA in der Praxis

■ Neue Zuweisungsoperatoren

Tabelle 4.2: Die neuen Operatoren in der Übersicht

Früher	Neu	Bedeutung
$n = n + 1$	$n+=1$	n wird um eins erhöht.
$n = n - 1$	$n-=1$	n wird um eins vermindert.
$n = n + m$	$n+=m$	n und m werden addiert.
$n = n - m$	$n-=m$	m wird von n subtrahiert.
$n = n * 4$	$n*=4$	n wird multipliziert.
$n = n / 2$	$n/=2$	n wird dividiert.
$s1 = s1 \& s2$	$s1\&=s2$	Zwei Zeichenketten werden verknüpft.

Unterschiede zwischen VB.NET und VBA in der Praxis

■ Auswertung mehrerer logischer Ausdrücke

Code-Beispiel

```
If IsNumeric(oZahl) = True AndAlso oZahl < 0 Then  
    Console.WriteLine("Bitte keine negativen Zahlen  
eingeben")  
End If
```

Unterschiede zwischen VB.NET und VBA in der Praxis

■ Feldvariablen (Arrays)

Code-Beispiel

```
Private a_iWerte() As Integer = {11, 12, 33}
```

Unterschiede zwischen VB.NET und VBA in der Praxis

■ Feldvariablen (Arrays)

Code-Beispiel

```
Private a_iWerte() As Integer = {11, 12, 33}
```

```
WriteLine("Anzahl Elemente: " & a_nWerte.Length)
```

Unterschiede zwischen VB.NET und VBA in der Praxis

■ Die Array-Klasse

Code-Beispiel (ConsoleApplication3 Module1.vb)

Das folgende Beispiel löscht in dem Array *a_nWerte* über die *Clear*-Methode der *Array*-Klasse ab dem 5. Wert genau 2 Elemente.

```
Imports System.Console
Imports Microsoft.VisualBasic.Information
Imports System.Array
Module Modul1
    Private a_nWerte() As Long = {11, 12, 33, 44, 55, 66, 77}
    Sub Main()
        WriteLine("Anzahl Elemente: " & a_nWerte.Length)
        WriteLine("Das ist vom Typ: " & TypeName(a_nWerte))
        WriteLine("Das 5. Element ist: " & a_nWerte(4))
        System.Array.Clear(a_nWerte, 4, 2)
        WriteLine("Das 5. Element ist: " & a_nWerte(4))
        WriteLine("Das 6. Element ist: " & a_nWerte(5))
        WriteLine("Das 7. Element ist: " & a_nWerte(6))
        ReadLine()
    End Sub
End Module
```

Unterschiede zwischen VB.NET und VBA in der Praxis

■ Schleifen

Code-Beispiel

```
While x > 0  
  ' Tue hier irgendetwas  
End While
```

Unterschiede zwischen VB.NET und VBA in der Praxis

■ Benutzerdefinierte Datentypen (Strukturen)

Code-Beispiel

Der benutzerdefinierte Datentyp *udtSpieler* sieht in VB wie folgt aus:

```
Type udtSpieler
    Name As String * 32
    SpielerNr As Byte
    HighScore As Byte
End Type
```

In VB.NET lautet die Definition wie folgt:

```
Structure strucSpieler
    Public Name As String
    Public SpielerNr As Byte
    Public HighScore As Byte
End Structure
```

Unterschiede zwischen VB.NET und VBA in der Praxis

■ Eingebaute Funktionen

Code-Beispiel

Das folgende Beispiel zeigt, wie die neue *Sqrt*-Funktion der .NET-Bibliothek aufgerufen wird.

```
Imports System.Console
Imports System.Math
Module Modul1
    Sub Main()
        Dim snZahl, snErgebnis As Double
        snZahl = 222
        snErgebnis = sqrt(snZahl)
        WriteLine("Ja, die Wurzel, die Wurzel ist: " &
snErgebnis)
    End Sub
End Module
```

Unterschiede zwischen VB.NET und VBA in der Praxis

■ Funktionen

Code-Beispiel

```
Private Sub cmdTest_Click()  
    Dim oRef As Ctest  
    oRef = New Ctest()  
    oRef.Wert = 123  
    P1(oRef)  
    MsgBox(oRef.Wert)  
End Sub  
Sub P1(ByRef objRef As CTest)  
    Dim oRef2 As Ctest  
    oRef2 = New Ctest()  
    oRef2.Wert = 456  
    objRef = oRef2  
End Sub
```

Unterschiede zwischen VB.NET und VBA in der Praxis

■ Rückgabewerte von Funktionen

Code-Beispiel

```
Function KreisFlaeche(ByVal r As Single) As Single
    Dim Pi = 22 / 7 As Single
    Return r ^ 2 * Pi
End Function
```

Unterschiede zwischen VB.NET und VBA in der Praxis

■ Die Definition einer Klasse

Code-Beispiel

```
Class Cspieler  
End Class
```

Unterschiede zwischen VB.NET und VBA in der Praxis

■ Instanzieren einer Klasse

Code-Beispiel

Der folgende Befehl instanziert die Klasse *CSpieler* und weist die resultierende Referenz der Variablen *oTorwart* zu.

```
Dim oTorwart As New CSpieler()
```

```
Dim oTorwart As CSpieler  
oTorwart = New CSpieler()
```

Unterschiede zwischen VB.NET und VBA in der Praxis

■ Implementieren von Eigenschaften und Methoden

Code-Beispiel

Das folgende Beispiel implementiert eine Eigenschaft *Name* in der Klasse *CSpieler*.

```
Dim m_sName As String
Property Name() As String
    Get
        Name = m_sName
    End Get
    Set(ByVal Value As String)
        m_sName = Value
    End Set
End Property
```

Unterschiede zwischen VB.NET und VBA in der Praxis

■ Implementieren von Eigenschaften und Methoden

Unter Visual Basic 6.0 würde die gleiche Klassendefinition wie folgt lauten:

```
Dim m_sName As String
Property Get Name() As String
    Name = m_sName
End Property
Property Let(ByVal Value As String)
    m_sName = Value
End Property
```

Unterschiede zwischen VB.NET und VBA in der Praxis

■ Die Rolle des Konstruktors

Code-Beispiel

Das folgende Beispiel ergänzt die Klasse *CSpieler* um einen Konstruktor, der dafür sorgt, dass die Eigenschaft *Name* mit dem Instanzieren der Klasse einen Wert erhält.

```
Sub New (ByVal Name As String)
    m_sName = Name
End Sub
```

Dank des Konstruktors, dem beliebig viele Argumente übergeben werden können, erhält die Eigenschaft *Name* mit dem Instanzieren der Klasse ihren Wert:

```
Dim oTorwart As CTorwart =
New CTorwart("Dino Zoff", 34)
WriteLine("Der Spieler heisst: " & oTorwart.Name)
```

Unterschiede zwischen VB.NET und VBA in der Praxis

■ Vererbung

Code-Beispiel (ConsoleApplication5 Module1.vb)

Das folgende Beispiel erweitert das Beispiel aus dem letzten Abschnitt um eine neue Klasse *CTorwart*, die über den *Inherits*-Befehl alle Mitglieder der (Basis-) Klasse *CSpieler* erbt.

Eine Besonderheit gibt es bezüglich des Konstruktors zu beachten. Da die neue Klasse von einer Klasse mit Konstruktor erbt, muss sie diesen Konstruktor in ihrem Konstruktor aufrufen, was über den Aufruf

```
MyBase.New (Name)
```

geschieht.

Unterschiede zwischen VB.NET und VBA in der Praxis

■ **Multithreading**

- Multithreading ist auch bei VB.NET nicht einfach zu programmieren
- Durch das Starten mehrerer Threads ergibt sich nicht automatisch eine Geschwindigkeitssteigerung.
- Durch die Verwendung von Threads stellen sich völlig neuartige Probleme

Unterschiede zwischen VB.NET und VBA in der Praxis

- **Multithreading-Unterstützung in VB.NET**
- Über das Thread Objekt der .NET Klasse System.Threading lassen sich neue Threads anlegen und wieder beenden. Die Adresse der als Thread auszuführenden Funktion oder Prozedur wird über den bekannten AddressOf Operator übergeben.
- Über den SynchLock Befehl wird eine Synchronisation mehrerer Threads bei der Ausführung ein und desselben Programmbereichs bewirkt.
- Die Visual Studio .NET IDE unterstützt das Debuggen von Multithreading Programmen.

Unterschiede zwischen VB.NET und VBA in der Praxis

■ Das Anlegen eines Threads - Code-Beispiel

```
' Dieser Thread zählt von 1 bis 200
Class CThread
    ' Die Methode ThreadWorker wird als eigener Thread
ausgeführt
    Public Sub ThreadWorker()
        Dim iCounter As Integer
        For iCounter = 1 To 200
            WriteLine("Meldung aus dem Tread: " &
iCounter.ToString())
        Next iCounter
    End Sub
End Class
```

Unterschiede zwischen VB.NET und VBA in der Praxis

■ Das Anlegen eines Threads - Code-Beispiel

■ Schritt 2:

■ Definieren Sie die Main Prozedur des Modul.

' Hier beginnt die Programmausführung

' Dies ist der Hauptthread des Programms

```
Sub Main()
```

Unterschiede zwischen VB.NET und VBA in der Praxis

■ Das Anlegen eines Threads - Code-Beispiel

■ Schritt 3:

- Die Aufgabe von Main ist es, ein neues Thread Objekt anzulegen:

```
Dim iCounter As Integer
' Eine neue Instanz der Klasse mit dem Thread anlegen
Dim oMT As CThread = New CThread()
' Ein neues Thread-Objekt anlegen
' Die Thread-Klasse ist Teil des System.Threading-Namensraums
Dim oNewThread As Thread
' Anlegen eines neuen Threads
' Thread soll die Methode ThreadWorker ausführen
oNewThread = New Thread(New ThreadStart(AddressOf
oMT.ThreadWorker))
```

Unterschiede zwischen VB.NET und VBA in der Praxis

■ Das Anlegen eines Threads - Code-Beispiel

- Schritt 4:
- Damit wurde ein neuer Thread angelegt, der im Folgenden auch gestartet werden soll (ansonsten passiert nichts):

```
' Der neue Thread wird gestartet
    ' Die Methode ThreadWorker läuft in einem
anderen Thread
    oNewThread.Start()
```

Unterschiede zwischen VB.NET und VBA in der Praxis

■ Das Anlegen eines Threads - Code-Beispiel

- Schritt 5:
- Damit sich feststellen lässt, dass ein weiterer Thread aktiv ist, soll auch der Hauptthread etwas arbeiten:

```
' Auch der Hauptthread soll etwas arbeiten
  For iCounter = 1 To 200
    WriteLine("Meldung aus dem Hauptprogramm: " & _
      iCounter.ToString())
  Next iCounter
  ReadLine()
```

Unterschiede zwischen VB.NET und VBA in der Praxis

- **Das Anlegen eines Threads - Code-Beispiel**
- Schritt 6:
- Starten Sie das Programm über [F5] oder kompilieren Sie die Datei und führen Sie sie aus.

Unterschiede zwischen VB.NET und VBA in der Praxis

■ Strukturierte Ausnahmebehandlung

Code-Beispiel

Try

```
Open(1, sDateiName, OpenMode.Input)
```

Sollte dieser Befehl fehlschlagen, werden automatisch die auf *Catch* folgenden Befehle ausgeführt.

Catch

```
sDateiName = _
```

```
  InputBox("Bitte Dateinamen korrigieren", ,
```

```
sDateiName)
```

```
Open(1, sDateiName, OpenMode.Input)
```

Unterschiede zwischen VB.NET und VBA in der Praxis

■ *Try-Catch-Finally*

Code-Beispiel

```
Sub Main()  
    Dim sDateiName As String = "C:\Autoexec.bat"  
    Dim sInhalt, sZeile As String  
    Try  
        Open(1, sDateiName, OpenMode.Input)  
    Catch  
        sDateiName = _  
            InputBox("Bitte Dateinamen korrigieren", , sDateiName)  
        Open(1, sDateiName, OpenMode.Input)  
    Finally  
        ' Hier gibt es nichts zu tun  
    End Try  
    Do While Not EOF(1)  
        sZeile = LineInput(1)  
        sInhalt += sZeile & vbCrLf  
    Loop  
    Close(1)  
    MsgBox(sInhalt)  
End Sub
```

Unterschiede zwischen VB.NET und VBA in der Praxis

■ *Abfragen der Fehlerinformationen*

```
Catch oSysEx As SystemException  
    MsgBox(oSysEx.Message)
```

Windows Forms programmieren

■ Anatomie des Visual Basic-Codes hinter Windows Forms

```
Public Class Form1
    Inherits System.Windows.Forms.Form
    #Region " Vom Windows Form Designer generierter Code "
    Public Sub New()
        MyBase.New()
        ' Dieser Aufruf ist für den Windows Form-Designer erforderlich.
        InitializeComponent()
        ' Initialisierungen nach dem Aufruf InitializeComponent() hinzufügen
    End Sub
    ' Die Form überschreibt den Löschvorgang der Basisklasse, um Komponenten zu bereinigen.
    Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub
    ' Für Windows Form-Designer erforderlich
    Private components As System.ComponentModel.IContainer
    'HINWEIS: Die folgende Prozedur ist für den Windows Form-Designer erforderlich
    'Sie kann mit dem Windows Form-Designer modifiziert werden.
    'Verwenden Sie nicht den Code-Editor zur Bearbeitung.
    <System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()
        components = New System.ComponentModel.Container()
        Me.Text = "Form1"
    End Sub
#End Region
End Class
```

Windows Forms programmieren

- Anatomie des Visual Basic-Codes hinter Windows Forms

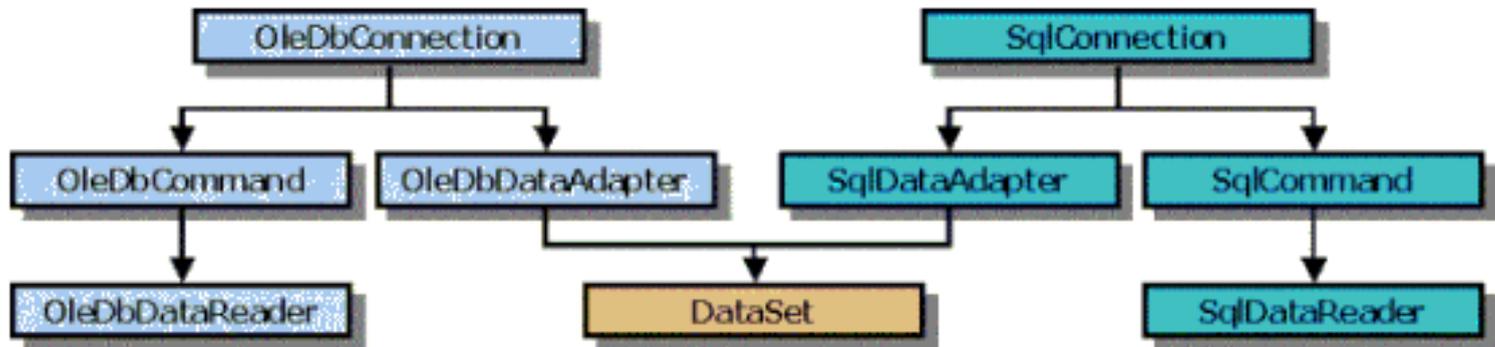
- **InitializeComponent**
- **Public Sub New():** Der Klassenkonstruktor.

Datenbankzugriff mit ADO.NET

- **Unterschiede zwischen ADO und ADO .NET:**
- **Verbindungsloser Datenbankzugriff**
- **DataSet-Objekt ersetzt das RecordSet-Objekt**
- **Client- und Serverseitige Cursor**
- **Volle XML-Unterstützung**

Datenbankzugriff mit ADO.NET

■ Erste Schritte in ADO.NET



Datenbankzugriff mit ADO.NET

■ Erste Schritte in ADO.NET

```
' Office VBA- und ADO-Code – ADOCode.bas.
```

```
Public Sub ADOExample()
```

```
    ' Zunächst Verweis auf ADO-Bibliothek festlegen.
```

```
    Dim objConn As ADODB.Connection
```

```
    Dim objRS As ADODB.Recordset
```

```
    Set objConn = New ADODB.Connection
```

```
    objConn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0; User ID=Admin;"Data  
Source=C:\Programme\Microsoft Office\Office10\Samplesordwind.mdb"
```

```
    objConn.Open
```

```
    Set objRS = objConn.Execute("SELECT * FROM Artikel")
```

```
    objRS.MoveFirst
```

```
    MsgBox Prompt:=objRS.Fields("ProductName").Value & ", " & _
```

```
        objRS.Fields("UnitsInStock").Value
```

```
    objRS.Close
```

```
    objConn.Close
```

```
End Sub
```

Datenbankzugriff mit ADO.NET

■ Erste Schritte in ADO.NET

```
Imports System.Data.OleDb
Module Module1
    Sub Main()
        ' Zunächst Verweis auf System.Data.dll festlegen.
        Dim objConn As New OleDbConnection()
        objConn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;" & _
            "Data Source=C:\Programme\Office XP\Office10\Samples\Nordwind.mdb"
        objConn.Open()
        Dim objCmd As New OleDbCommand _
            ("SELECT * FROM Artikel", objConn)
        Dim objDataReader As OleDbDataReader = objCmd.ExecuteReader
        Do While objDataReader.Read() = True
            Console.Write(objDataReader.Item("Artikelname") & ", " & _
                objDataReader.Item("LagerBestand"))
        Loop
        Console.ReadLine()
    End Sub
End Module
```

Datenbankzugriff mit ADO.NET

■ Erste Schritte in ADO.NET

```
Imports System.Data.OleDb
Module Module1
    Sub Main()
        ' Zunächst Verweis auf System.Data.dll festlegen.
        Dim objConn As New OleDbConnection _
            ("Provider=Microsoft.Jet.OLEDB.4.0;" & _
            "Data Source=C:\Programme\Office XP\Office10\Samples\Nordwind.mdb")
        objConn.Open()
        Dim objAdapter As New OleDbDataAdapter _
            ("SELECT * FROM Artikel", objConn)
        Dim objDataSet As New DataSet()
        objAdapter.Fill(objDataSet)
        With objDataSet.Tables("Table").Rows(0)
            Console.WriteLine(.Item("ArtikelName") & ", " & _
                & .Item("LagerBestand"))
        End With
        Console.ReadLine()
    End Sub
End Module
```

Datenbankzugriff mit ADO.NET

- **Erste Schritte in ADO.NET**
- **Kann ich ADO.NET in Office VBA verwenden?**

Datenbankzugriff mit ADO.NET

- **Verwenden der Visual Studio .NET-Datenzugriffstools**
- **Datenverbindungen**

Datenbankzugriff mit ADO.NET

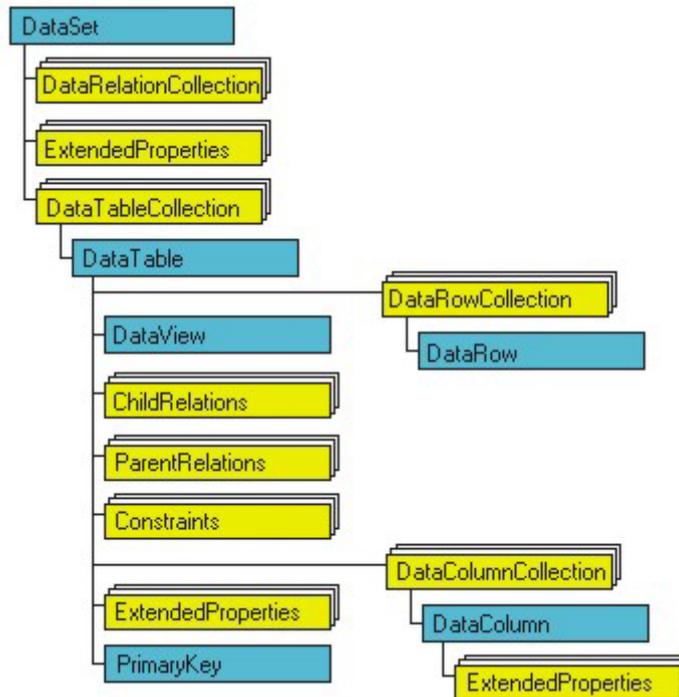
- **Verwenden der Visual Studio .NET-Datenzugriffstools**
- **Datenbankprojekte**

Datenbankzugriff mit ADO.NET

- **Verwenden der Visual Studio .NET-Datenzugriffstools**
- **DataForm-Assistent**

Datenbankzugriff mit ADO.NET

■ Objekte in ADO.NET



Datenbankzugriff mit ADO.NET

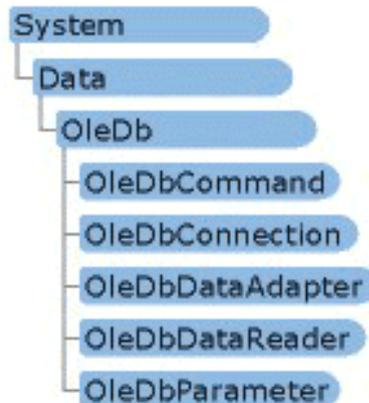
■ Objekte in ADO.NET

Objekt	Beschreibung
DataSet	Das Objekt wird in Verbindung mit anderen Datensteuerelementen benutzt, um Ergebnisse von Commands und DataAdapters zu speichern. Im Gegensatz zum Recordset von ADO und DAO ist das DataSet in der Lage, Daten hierarchisch darzustellen. Mit Hilfe der Eigenschaften und Auflistungen des DataSet-Objekts können Sie alles von der Beziehung bis hin zur einzelnen Zeile oder Spalte erreichen.
DataTable	DataTable ist eines der Objekte des DataSets, das es Ihnen erlaubt, einzelne Datentabellen zu bearbeiten. Es ähnelt dem Recordset von ADO.
DataGridView	Mit diesem Objekt können Sie Ihre Daten filtern und sortieren, um verschiedene Ansichten der Daten zu haben. Jedes DataTable-Objekt hat einen DataGridView, der den Ausgangs-DataGridView darstellt. Dieser kann modifiziert und gespeichert werden.
DataRow	Dieses Objekt ermöglicht es Ihnen, einzelne Zeilen Ihrer DataTable zu modifizieren. Sie können sich das Objekt wie einen Datencache vorstellen, den Sie bearbeiten können, das heißt, Sie können Daten ändern, hinzufügen und löschen. Die Änderungen schreiben Sie dann zurück in das Recordset, indem Sie SQL-Befehle auf dem Server ausführen.
DataColumn	Das Objekt repräsentiert Spalten. Das Interessante daran ist, dass Sie sowohl Schemainformationen als auch Daten erhalten können. Möchten Sie zum Beispiel ein Listenfeld mit Feldnamen füllen, können Sie die DataColumn Collection einer DataRow durchlaufen und die Feldnamen auslesen.
PrimaryKey	Dieses Objekt erlaubt es Ihnen, einen Primärschlüssel für eine DataTable anzugeben, der zum Beispiel bei der Verwendung der Find-Methode wichtig ist.

Datenbankzugriff mit ADO.NET

■ Objekte in ADO.NET

OleDb Manage-Provider Namespace



SqlClient Managed-Provider Namespace



Datenbankzugriff mit ADO.NET

■ Objekte in ADO.NET

Objekt	Beschreibung
Command	Ähnlich dem ADO-Command-Objekt dient es dazu, Stored Procedures auszuführen. Im Gegensatz zu ADO können Sie ein DataReader- Objekt erstellen, indem Sie die Methode ExecuteReader ausführen.
Connection	Dieses Objekt öffnet eine Verbindung zum Server und zu der Datenbank, mit der Sie arbeiten wollen. Im Unterschied zum ADO-Connection-Objekt hängt es von dem Objekt ab, mit dem Sie arbeiten (DataReader oder DataSet), ob die Verbindung bestehen bleibt.
DataAdapter	Der DataAdapter ist ein echtes Arbeitstier. Das Objekt ermöglicht das Erzeugen von SQL-Befehlen und das Füllen von Datasets. Es erzeugt außerdem Aktionsabfragen wie Insert, Update und Delete.
DataReader	Erstellt einen read-only, forward-only Datastream, der sich besonders für Steuerelemente wie Listen- und Kombinationsfelder eignet.
Parameter	Dieses Objekt erlaubt es Ihnen, Parameter für DataAdapter-Objekte zu spezifizieren.