# Codekonventionen für C# (C#-Programmierhandbuch)

**Visual Studio 2015**

Die C#-Sprachspezifikation definiert keinen Codierungsstandard.Die Richtlinien in diesem Thema werden jedoch von Microsoft verwendet, um Beispiele und die Dokumentation zu entwickeln.

Codierungskonventionen dienen den folgenden Zwecken:

- Sie sorgen für eine konsistente Gestaltung des Codes, damit sich die Leser auf den Inhalt, nicht auf das Layout konzentrieren können.
- Sie ermöglichen es den Lesern, Code schneller zu verstehen, da Rückschlüsse basierend auf den bisherigen Erfahrungen gezogen werden können.
- Sie erleichtern das Kopieren, Ändern und Pflegen des Codes.
- Sie zeigen die Best Practices für C#.

## Namenskonventionen

- Verwenden Sie in kurzen Beispielen, die keine using-Direktiven umfassen, Namespacequalifizierungen.Wenn Sie wissen, dass ein Namespace standardmäßig in ein Projekt importiert wird, müssen Sie den Namen aus diesem Namespace nicht voll qualifizieren.Qualifizierte Namen können nach einem Punkt (.) unterbrochen werden, wenn sie für eine einzelne Zeile zu lang sind, wie im folgenden Beispiel gezeigt.

  **C#**

  ```
  var currentPerformanceCounterCategory = new System.Diagnostics.
      PerformanceCounterCategory();
  ```

- Sie müssen die Namen von Objekten, die mit den Visual Studio Designertools erstellt wurden nicht ändern, um sie anderen Richtlinien anzupassen.

## Layoutkonventionen

Ein gutes Layout verwendet Formatierungen, um die Struktur des Codes hervorzuheben und um den Code verständlicher zu gestalten.Microsoft-Beispiele entsprechen den folgenden Konventionen:

- Verwenden Sie die Code-Editor-Standardeinstellungen (Intelligenter Einzug, vierstelliger Einzug, als Leerzeichen gespeicherte Tabulatoren).Weitere Informationen finden Sie unter Optionen, Text-Editor, C#, Formatierung
- Schreiben Sie pro Zeile nur eine Anweisung.
- Schreiben Sie pro Zeile nur eine Deklaration.
- Wenn Fortsetzungszeilen nicht automatisch eingezogen werden, rücken Sie diese um einen Tabstopp (vier Leerzeichen) ein.
- Fügen Sie zwischen Methoden- und Eigenschaftsdefinitionen mindestens eine Leerzeile ein.
- Verwenden Sie Klammern, um Klauseln in einem Ausdruck zu kennzeichnen, wie im folgenden Code gezeigt.

**C#**

```
if ((val1 > val2) && (val1 > val3))
{
    // Take appropriate action.
}
```

# Konventionen für Kommentare

- Fügen Sie den Kommentar in einer eigenen Zeile und nicht am Ende einer Codezeile ein.
- Beginnen Sie Kommentartext mit einem Großbuchstaben.
- Beenden Sie den Kommentartext mit einem Punkt.
- Fügen Sie ein Leerzeichen zwischen dem Kommentartrennzeichen (//) und dem Kommentartext ein, wie im folgenden Beispiel gezeigt.

**C#**

```
// The following declaration creates a query. It does not run
// the query.
```

- Erstellen Sie keine formatierten Blöcke mit Sternchen um Kommentare.

# Sprachrichtlinien

In den folgenden Abschnitten werden die Vorgehensweisen beschrieben, denen das C#-Team folgt, um Codebeispiele zu erstellen.

## String-Datentyp

- Verwenden Sie den **+**-Operator, um kurze Zeichenfolgen zu verketten, wie im folgenden Code gezeigt.

**C#**

```
string displayName = nameList[n].LastName + ", " + nameList[n].FirstNam
```

- Verwenden Sie ein StringBuilder-Objekt, um Zeichenfolgen in Schleifen anzufügen, besonders bei der Arbeit mit großen Textmengen.

**C#**

```
var phrase = "lalalalalalalalalalalalalalalalalalalalalalalalalalalalalal
var manyPhrases = new StringBuilder();
for (var i = 0; i < 10000; i++)
{
    manyPhrases.Append(phrase);
}
//Console.WriteLine("tra" + manyPhrases);
```

## Implizit typisierte lokale Variablen

- Verwenden Sie die implizite Typisierung für lokale Variablen, wenn der Typ der Variablen auf der rechten Seite der Zuweisung offensichtlich ist oder wenn der genaue Typ nicht von Bedeutung ist.

**C#**

```
// When the type of a variable is clear from the context, use var
// in the declaration.
var var1 = "This is clearly a string.";
var var2 = 27;
var var3 = Convert.ToInt32(Console.ReadLine());
```

- Verwenden Sie keine var, wenn der Typ nicht von der rechten Seite der Zuweisung offensichtlich ist.

**C#**

```
// When the type of a variable is not clear from the context, use an
// explicit type.
int var4 = ExampleClass.ResultSoFar();
```

- Verlassen Sie sich nicht auf den Variablennamen, um den Typ der Variable anzugeben.Er ist unter Umständen nicht korrekt.

**C#**

```
// Naming the following variable inputInt is misleading.
// It is a string.
var inputInt = Console.ReadLine();
Console.WriteLine(inputInt);
```

- Vermeiden Sie den Einsatz von **var** anstelle von dynamic.
- Verwenden Sie die implizite Typisierung, um den Typ der Schleifenvariablen in for- und foreach-Schleifen zu bestimmen.

  Im folgenden Beispiel wird die implizite Typisierung in einer **for**-Anweisung verwendet.

  **C#**

```
var syllable = "ha";
var laugh = "";
for (var i = 0; i < 10; i++)
{
    laugh += syllable;
    Console.WriteLine(laugh);
}
```

  Im folgenden Beispiel wird die implizite Typisierung in einer **foreach**-Anweisung verwendet.

  **C#**

```
foreach (var ch in laugh)
{
    if (ch == 'h')
        Console.Write("H");
    else
        Console.Write(ch);
}
Console.WriteLine();
```

## Datentyp ohne Vorzeichen

- Verwenden Sie im Allgemeinen **int** anstelle von Typen ohne Vorzeichen.Die Verwendung von **int** ist in C# üblich; durch den Einsatz von **int** wird die Interaktion mit anderen Bibliotheken vereinfacht.

## Arrays

- Verwenden Sie die präzise Syntax, wenn Sie Arrays in der Deklarationszeile initialisieren.

  **C#**

```
// Preferred syntax. Note that you cannot use var here instead of strin
string[] vowels1 = { "a", "e", "i", "o", "u" };
```

```
// If you use explicit instantiation, you can use var.
var vowels2 = new string[] { "a", "e", "i", "o", "u" };

// If you specify an array size, you must initialize the elements one a
var vowels3 = new string[5];
vowels3[0] = "a";
vowels3[1] = "e";
// And so on.
```

## Delegaten

- Verwenden Sie die präzise Syntax, um Instanzen eines Delegattyps zu erstellen.

**C#**

```
  // First, in class Program, define the delegate type and a method tha
// has a matching signature.

// Define the type.
public delegate void Del(string message);

// Define a method that has a matching signature.
public static void DelMethod(string str)
{
    Console.WriteLine("DelMethod argument: {0}", str);
}
```

**C#**

```
  // In the Main method, create an instance of Del.

// Preferred: Create an instance of Del by using condensed syntax.
Del exampleDel2 = DelMethod;

// The following declaration uses the full syntax.
Del exampleDel1 = new Del(DelMethod);
```

## try-catch- und using-Anweisungen in der Ausnahmebehandlung

- Verwenden Sie eine try-catch--Anweisung für die meisten Ausnahmebehandlungen.

**C#**

```
static string GetValueFromArray(string[] array, int index)
{
    try
    {
        return array[index];
    }
    catch (System.IndexOutOfRangeException ex)
    {
        Console.WriteLine("Index is out of range: {0}", index);
        throw;
    }
}
```

- Vereinfachen Sie den Code mithilfe der C#-Anweisung using.Verwenden Sie bei einer try-finally-Anweisung, in der der einzige Code im **finally**-Block ein Aufruf der Dispose-Methode ist, stattdessen eine **using**-Anweisung.

**C#**

```
// This try-finally statement only calls Dispose in the finally block.
Font font1 = new Font("Arial", 10.0f);
try
{
    byte charset = font1.GdiCharSet;
}
finally
{
    if (font1 != null)
    {
        ((IDisposable)font1).Dispose();
    }
}


// You can do the same thing with a using statement.
using (Font font2 = new Font("Arial", 10.0f))
{
    byte charset = font2.GdiCharSet;
}
```

## &&- und ||-Operatoren

- Um Ausnahmen zu vermeiden und die Leistung zu verbessern, indem Sie unnötige Vergleiche überspringen, verwenden Sie && anstelle von & und || anstelle von | , wenn Sie Vergleiche ausführen, wie im folgenden Beispiel gezeigt.

**C#**

```
Console.Write("Enter a dividend: ");
var dividend = Convert.ToInt32(Console.ReadLine());
```

```
Console.Write("Enter a divisor: ");
var divisor = Convert.ToInt32(Console.ReadLine());

// If the divisor is 0, the second clause in the following condition
// causes a run-time error. The && operator short circuits when the
// first expression is false. That is, it does not evaluate the
// second expression. The & operator evaluates both, and causes
// a run-time error when divisor is 0.
if ((divisor != 0) && (dividend / divisor > 0))
{
    Console.WriteLine("Quotient: {0}", dividend / divisor);
}
else
{
    Console.WriteLine("Attempted division by 0 ends up here.");
}
```

## New-Operator

- Verwenden Sie die präzise Form der Objektinstanziierung mit impliziter Typisierung, wie in der folgenden Deklaration dargestellt.

**C#**

```
var instance1 = new ExampleClass();
```

Die vorherige Zeile entspricht der folgenden Deklaration.

**C#**

```
ExampleClass instance2 = new ExampleClass();
```

- Verwenden Sie Objektinitialisierer, um die Objekterstellung zu vereinfachen.

**C#**

```
// Object initializer.
var instance3 = new ExampleClass { Name = "Desktop", ID = 37414,
    Location = "Redmond", Age = 2.3 };

// Default constructor and assignment statements.
var instance4 = new ExampleClass();
instance4.Name = "Desktop";
instance4.ID = 37414;
instance4.Location = "Redmond";
instance4.Age = 2.3;
```

## Ereignisbehandlung

- Wenn Sie einen Ereignishandler definieren, den Sie später nicht entfernen müssen, verwenden Sie einen Lambda-Ausdruck.

**C#**

```
  public Form2()
{
    // You can use a lambda expression to define an event handler.
    this.Click += (s, e) =>
        {
            MessageBox.Show(
                ((MouseEventArgs)e).Location.ToString());
        };
}
```

**C#**

```
  // Using a lambda expression shortens the following traditional defin
public Form1()
{
    this.Click += new EventHandler(Form1_Click);
}

void Form1_Click(object sender, EventArgs e)
{
    MessageBox.Show(((MouseEventArgs)e).Location.ToString());
}
```

## Statische Member

- Rufen Sie statische Member über den Klassennamen *Klassenname.StatischerMember* auf.Durch diese Empfehlung ist der Code besser lesbar, da der statische Zugriff eindeutig ist.Qualifizieren Sie keinen statischen Member, der in einer Basisklasse mit dem Namen einer abgeleiteten Klasse definiert ist.Während dieser Code kompiliert wird, ist die Lesbarkeit des Codes irreführend, und der Code kann später beschädigt werden, wenn Sie der abgeleiteten Klasse einen statischen Member mit dem gleichen Namen hinzufügen.

## LINQ-Abfragen

- Verwenden Sie aussagekräftige Namen für Abfragevariablen.Im folgenden Beispiel wird `seattleCustomers` für Kunden in Seattle verwendet.

**C#**

```
var seattleCustomers = from cust in customers
```

```
                          where cust.City == "Seattle"
                          select cust.Name;
```

- Verwenden Sie Aliase, um mithilfe der Pascal-Schreibweise sicherzustellen, dass die korrekte Großschreibung von Eigenschaftennamen anonymer Typen verwendet wird.

**C#**

```
var localDistributors =
    from customer in customers
    join distributor in distributors on customer.City equals distributo
    select new { Customer = customer, Distributor = distributor };
```

- Benennen Sie Eigenschaften um, wenn die Eigenschaftennamen im Ergebnis nicht eindeutig sind.Wenn die Abfrage beispielsweise einen Kundennamen und eine Händler-ID zurückgibt, anstatt sie als `Name` und `ID` im Ergebnis beizubehalten, benennen Sie sie um, um zu verdeutlichen, dass `Name` der Name eines Kunden und `ID` die ID eines Händlers ist.

**C#**

```
var localDistributors2 =
    from cust in customers
    join dist in distributors on cust.City equals dist.City
    select new { CustomerName = cust.Name, DistributorID = dist.ID };
```

- Verwenden Sie die implizierte Typisierung in der Deklaration von Abfragevariablen und Bereichsvariablen.

**C#**

```
var seattleCustomers = from cust in customers
                       where cust.City == "Seattle"
                       select cust.Name;
```

- Richten Sie Abfrageklauseln unter der from-Klausel aus, wie in den vorherigen Beispielen gezeigt.

- Verwenden Sie vor anderen Abfrageklauseln where-Klauseln, um sicherzustellen, dass nachfolgende Abfrageklauseln für den reduzierten, gefilterten Datensatz ausgeführt werden.

**C#**

```
var seattleCustomers2 = from cust in customers
                        where cust.City == "Seattle"
                        orderby cust.Name
                        select cust;
```

- Verwenden Sie mehrere **from**-Klauseln anstelle von einer join-Klausel, um auf die inneren Auflistungen zuzugreifen.Eine Auflistung von `Student`-Objekten kann beispielsweise jeweils eine Auflistung von Testergebnissen enthalten.Wenn die folgende Abfrage ausgeführt wird, wird jedes Ergebnis über 90 zusammen mit dem Nachnamen des Studenten zurückgegeben, der das Testergebnis erzielt hat.

C#

```
// Use a compound from to access the inner sequence within each element
var scoreQuery = from student in students
                 from score in student.Scores
                 where score > 90
                 select new { Last = student.LastName, score };
```

## Sicherheit

Befolgen Sie die Richtlinien in Secure Coding Guidelines.

## Siehe auch

Visual Basic Coding Conventions
Secure Coding Guidelines