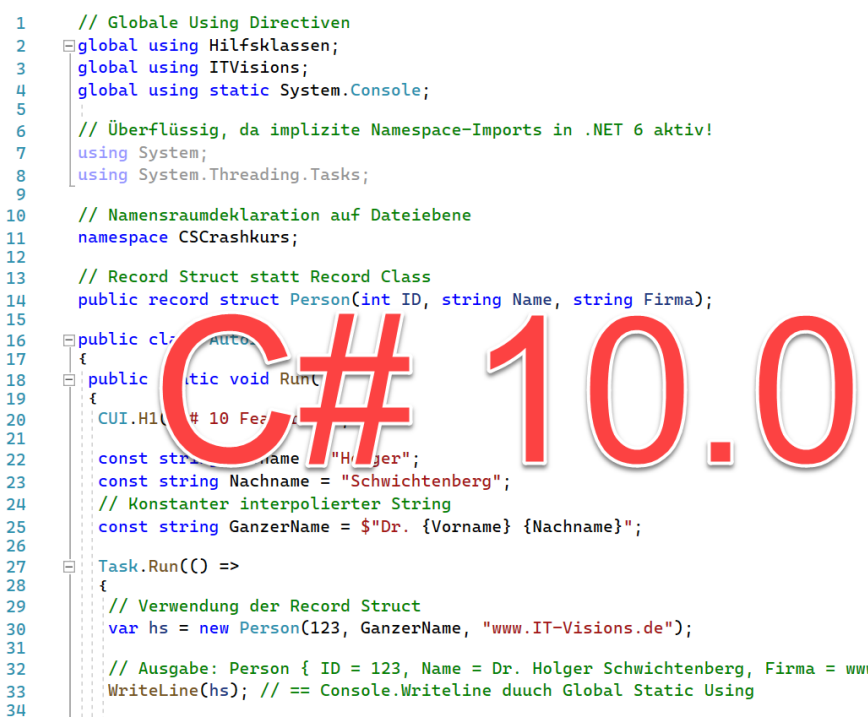
Dr. Holger Schwichtenberg

C# 10.0 Crashkurs

Die Syntax der Programmiersprache C# für die Softwareentwicklung in .NET Framework, .NET Core, .NET 5.0 und .NET 6.0



www.IT-Visions.de®
Dr. Holger Schwichtenberg

Buchversion:	5.7.0 vom 13.11.2021
Verlag:	www.IT-Visions.de, Fahrenberg 40b, D-45257 Essen
Sprachliche Korrektur:	Matthias Bloch, M.A.
ISBN:	978-3-934-27941-4
Bezugsquelle Druck:	*www.amazon.de/exec/obidos/ASIN/3934279414/itvisions-21*
Bezugsquelle Kindle:	*www.amazon.de/exec/obidos/ASIN/B09G2RG7JB/itvisions-21*
Bezugsquelle PDF:	*www.leanpub.com/CSharp10*

1 Inhaltsverzeichnis (Hauptkapitel)

2 Inhaltsverzeichnis (Details)

3 Vorwort

Liebe Leserinnen und Leser,

der "C# Crashkurs" ist ein prägnanter Überblick über die Syntax der Programmiersprache C# in der aktuellen Version 10.0.

Dieses Buch ist geeignet für Softwareentwickler, die von einer anderen objektorientierten Programmiersprache (z.B. C++, Java, JavaScript, Visual Basic .NET, Delphi oder PHP) auf C# umsteigen wollen oder bereits C# einsetzen und ihr Wissen erweitern insbesondere die neusten Sprachfeatures kennenlernen wollen. Wir schulen bei www.IT-Visions.de jedes Jahr hunderte Entwickler auf C# bzw. die neuste Version der Sprache um. Da es viele Umsteiger von Visual Basic .NET zu C# gibt, werden hier die Unterschiede von C# gegenüber Visual Basic .NET an einigen Stellen im Buch hervorgehoben.

Für Neueinsteiger, die mit C# erstmals überhaupt eine objektorientiere Programmiersprache (OOP) erlernen wollen, ist dieses Werk hingegen nicht geeignet, denn es werden die OO-Grundkonzepte nicht erklärt, da die meisten Softwareentwickler heutzutage diese aus anderen Sprachen kennen und das Buch nicht mit diesen Grundlagen unnötig in die Länge gezogen werden soll.

Dieser Crashkurs erhebt nicht den Anspruch, alle syntaktischen Details zu C# aufzuzeigen, sondern nur die in der Praxis wichtigsten Konstrukte.

In diesem Buch werden bewusst alle Syntaxbeispiele anhand von Konsolenanwendungen gezeigt. So brauchen Sie als Leser kein Wissen über irgendeine (oft kurzlebige) GUI-Bibliothek und die Beispiele sind prägnant fokussiert auf die Syntax.

Dieses Buch wird vertrieben:

- PDF-E-Book bei Leanpub.com ab 19,99 Dollar (der Autor erhält ca. 14,00 Euro): *www.leanpub.com/CSharp10*
- gedruckt (Print-on-Demand) bei Amazon.de für 29,99 Euro (der Autor erhält 12,52 Euro): *www.amazon.de/exec/obidos/ASIN/3934279414/itvisions-21*
- Kindle-E-Book bei Amazon.de für 9,99 Euro (der Autor erhält 6,29 Euro): *www.amazon.de/exec/obidos/ASIN/B09G2RG7JB/itvisions-21*

Tipp: Als Käufer bei Leanpub.com können jederzeit Aktualisierungen des PDF-Buchs (gleiche Hauptversion) kostenfrei dort beziehen. Amazon erlaubt dies leider nicht! Ich biete daher Käufern bei Amazon die PDF-Version zum Sonderpreis von 9,99 Dollar an: *www.leanpub.com/CSharp10/c/Sloborn*

Ich habe mich für den Vertriebsweg des gedruckten Buchs über Amazon entschieden, weil ich dort ständig Updates zu dem Buch einreichen kann. Per Print-on-Demand erhalten Leser dann immer das topaktuelle Buch. Oft liefert Amazon dennoch am Tag nach der Bestellung das Buch schon aus. Der Vertrieb dieses Buch über klassische IT-Verlage, die leider heutzutage immer noch größere Auflagen vorproduzieren, sind für ein sehr agiles Softwareprodukt wie .NET/C# keine Alternative mehr.

Da solch niedrige Preise leider nicht nennenswert dazu beitragen können, den Lebensunterhalt meiner Familie zu bestreiten, ist dieses Projekt ein Hobby. Dementsprechend kann ich nicht garantieren, wann es Updates zu diesem Buch geben wird. Ich werde dann an diesem Buch arbeiten, wenn ich neben meinem Beruf als Softwarearchitekt, Berater und Dozent und meinen sportlichen Betätigungen noch etwas Zeit für das Fachbuchautorenhobby übrig habe.

Falls mir in diesem Buch oder den zugehörigen Downloads menschliche Fehler passiert sind, möchte ich mich dafür schon jetzt in aller Form entschuldigen bei Ihnen. Bitte geben Sie mir einen freundlichen, genau beschriebenen Hinweis auf meine Fehler. Ich freue mich immer über konstruktives Feedback und Verbesserungsvorschläge. Bitte verwenden Sie dazu das Kontaktformular: *www.dotnet-doktor.de/Leserfeedback*

> **Tipp:** Ich belohne Sie mit E-Books für gemeldete Fehler, siehe Kapitel "Über dieses Buch / Ihre Belohnung, wenn Sie helfen, dieses Buch zu verbessern".

Ich helfe Ihnen gerne, Ihren eigenen Programmcode zu schreiben, aber ich hoffe, Sie verstehen, dass ich dies nicht ehrenamtlich tun kann. Wenn Sie **technische Hilfe** zu Entity Framework und Entity Framework Core oder anderen Themen rund um die Entwicklung und den Betrieb von Anwendungen (Desktop, Web und Mobile) sowie Server und Cloud benötigen, stehe ich Ihnen im Rahmen meiner beruflichen Tätigkeit für die Firmen *www.IT-Visions.de* (Beratung, Schulung, Support) und MAXIMAGO GmbH (Softwareentwicklung, siehe *www.MAXIMAGO.de*) gerne zur Verfügung. Bitte wenden Sie sich für ein Angebot an das jeweilige Kundenteam. Bitte kontaktieren Sie die Firmen aber nicht für Feedback und Verbesserungsvorschläge zu diesem Buch, da dieses Buch reine Privatsache ist.

Auf der von mir ehrenamtlich betriebenen **Leser-Website** unter *www.IT-Visions.de/Leser*, können Sie die Beispiele zu diesem Buch herunterladen. Dort müssen Sie sich registrieren. Bei der Registrierung wird ein Losungswort abgefragt. Bitte geben Sie dort bei der Registrierung das Losungswort **Sloborn** ein.

Herzliche Grüße aus Essen, dem Herzen der Metropole Ruhrgebiet

Holger Schwichtenberg

4 Über den Autor

- Studienabschluss Diplom-Wirtschaftsinformatik an der Universität Essen
- Promotion an der Universität Essen im Fachgebiet komponentenbasierter Softwareentwicklung
- Seit 1996 selbstständig als unabhängiger Berater, Dozent, Softwarearchitekt und Fachjournalist
- Fachlicher Leiter des Expertenteams bei *www.IT-Visions.de* in Essen
- Chief Technology Expert (CTE) der Softwareentwicklung bei der MAXIMAGO GmbH in Dortmund (*www.MAXIMAGO.de*)
- Über 80 Fachbücher beim Carl Hanser Verlag, bei O'Reilly, Microsoft Press, APress und Addison-Wesley sowie mehr als 1300 Beiträge in Fachzeitschriften
- Gutachter in den Wettbewerbsverfahren der EU gegen Microsoft (2006-2009)
- Ständiger Mitarbeiter der Zeitschriften iX (seit 1999), dotnetpro (seit 2000) und Windows Developer (seit 2010) sowie beim Online-Portal heise.de (seit 2008)
- Regelmäßiger Sprecher auf nationalen und internationalen Fachkonferenzen (z.B. BASTA!, enterJS, Microsoft TechEd, Microsoft Summit, Microsoft IT Forum, OOP, IT Tage, .NET Architecture Camp, Advanced Developers Conference, Developer Week, DOTNET Cologne, MD DevDays, Community in Motion, DOTNET-Konferenz, VS One, NRW.Conf, Net.Object Days, Windows Forum, Container Conf)
- Zertifikate und Auszeichnungen von Microsoft:
  - Microsoft Most Valuable Professional (MVP), kontinuierlich ausgezeichnet seit 2004
  - Microsoft Certified Solution Developer (MCSD)
- Thematische Schwerpunkte:
  - Softwarearchitektur, mehrschichtige Softwareentwicklung, Softwarekomponenten
  - Visual Studio, Continous Integration (CI) und Continous Delivery (CD) mit Azure DevOps
  - Microsoft .NET (.NET Framework, .NET Core), C#, Visual Basic
  - .NET-Architektur/Auswahl von .NET-Techniken
  - Einführung von .NET und Visual Studio/Migration auf .NET
  - Webanwendungsentwicklung und Cross-Plattform-Anwendungen mit HTML, ASP.NET (Core), JavaScript/TypeScript und Webframeworks wie Angular, Vue.js und Blazor
  - Enterprise .NET, verteilte Systeme/Webservices mit .NET, insbesondere WebAPI, gRPC und WCF
  - Relationale Datenbanken, XML, Datenzugriffsstrategien
  - Objekt-Relationales Mapping (ORM), insbesondere ADO.NET Entity Framework und Entity Framework Core
  - PowerShell
  - Architektur- und Code-Reviews
  - Performance-Analysen und -Optimierung
  - Entwicklungsrichtlinien
- Ehrenamtliche Community-Tätigkeiten:
  - Vortragender für die International .NET Association (INETA) und .NET Foundation
  - Betrieb diverser Community-Websites: *www.dotnet-lexikon.de*, *www.dotnetframework.de*, *www.windows-scripting.de*, *www.aspnetdev.de* u. a.
- Firmenwebsites: *www.IT-Visions.de* und *www.MAXIMAGO.de*
- Weblog: *www.dotnet-doktor.de*

www.IT-Visions.de
Dr. Holger Schwichtenberg

MAXIMAGO

- Kontakt für Anfragen zu Schulung und Beratung:
  kundenteam@IT-Visions.de
  Telefon **0201 / 64 95 90 - 50**
- Kontakt für Anfragen zu Softwareentwicklungsprojekten:
  hsc@MAXIMAGO.de
  Telefon **0231 / 58 69 67 - 12**
- Kontakt für Feedback zu diesem Buch:
  www.dotnet-doktor.de/Leserfeedback

5 Über dieses Buch

5.1 Bezugsquelle für Aktualisierungen

Sie können jederzeit Aktualisierungen des PDF-Buchs (gleiche Hauptversion!) kostenfrei bei Leanpub.com beziehen.

Käufer der Kindle- oder Druck-Version können die aktuelle PDF-Version zum Preis von 9,99 Dollar (zzgl. 7% Mehrwertsteuer) unter folgender Webadresse beziehen:

https://leanpub.com/CSharp10/c/Sloborn

Hinweise: Leider erlauben Amazon u.a. Buchhändler aufgrund der Buchpreisbindungsgesetze in Deutschland den Autoren grundsätzlich nicht, dass Leser eine Aktualisierung im Kindle-Format oder in gedruckter Form vergünstigt erhalten.

Bitte beachten Sie auch, dass die ISBN-Regularien erfordern, dass bei einer Titeländerung bei neuer Produktversion eine neue ISBN vergeben werden muss und damit auch ein neues Buchprojekt bei Amazon und Leanpub.com erstellt werden muss.

5.2 Versionsgeschichte dieses Buchs

Die Versionsgeschichte dieses Buch finden Sie in einem eigenen Kapitel am Ende des Buchs.

Hinweis: Die Versionsgeschichte ist eine wichtige Referenz für die Leser, die sich aktuelle Versionen des Buchs beschaffen (z.B. über Leanpub.com) und wissen wollen, was sich geändert hat. Wenn Sie das Buch erstmalig lesen, müssen Sie die Versionsgeschichte nicht lesen.

5.3 Hinweise zur Breite und Tiefe dieses Buch – Sie haben Einfluss!

Ein Fachbuch, das ein riesengroßes Themengebiet wie C# behandelt, kann nicht jedes Teilgebiet und jeden Aspekt der Programmiersprache behandeln, zumindest nicht in gleicher Tiefe. Dann würde solch ein Fachbuch über eintausend Seiten, in einigen Fällen sogar mehrere tausend Seiten umfassen.

Ich denke, dass ich nach aktuellem Stand der Technik und meinem Wissenstand etwa 1000 zur C#-Syntax und -Tools sowie 3000 Seiten zu den C#-Bibliotheken schreiben könnte. Würden Sie so ein dickes (und entsprechend teures) Buch kaufen und lesen wollen?

Wie jeder Fachautor lese auch ich immer wieder Kritik, dass ein(e) Leser*in ein bestimmtes Thema nicht oder nicht in ausreichender Tiefe behandelt sei in dem Buch. Das ist aus der Sicht der einzelnen Leser*in sicherlich gerechtfertigt, aber wie jeder Fachautor muss ich eben zwingend eine Auswahl der Themen treffen. Gerne dokumentiere ich hier, wie ich persönlich diese Auswahl für meine Bücher treffe:

- Ich behandele im Buch die Themen, die wir in unserer Firma selbst in der Praxis brauchen.
- Ich behandele zusätzlich die Themen, die unsere Kunden in Beratungsgesprächen behandelt haben möchten.

Folglich sind die Themen, die ich im Buch nicht oder nur kurz behandele für uns und unsere Kunden nicht relevant bzw. so selbsterklärend, dass es keine Fragen dazu gibt.

Natürlich kann das für Sie anders sein. Sie können mir immer gerne schreiben, wenn Sie ein Thema im Buch behandelt haben möchten. Ich sammele diese Anregungen und wenn es mehrere Zuschriften zu einem Thema gibt, dann kommt das Thema auf weit oben auf die Prioritätenliste. Ich denke, das ist ein faires Verfahren.

5.4 Geplante Themen

Folgende Themen sind für kommenden Ausgaben dieses Buchs geplant:

- Aliase für referenzierte Assemblies
- Clean Code-Programmierung mit C#
- Covariant Return Types (seit C# 9.0)
- Design Pattern in C#
- Dekompilierung mit ILSpy
- Extension Method GetEnumerator() (seit C# 9.0)
- Implicit Cast Operator [*docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/implicit*]
- Inkrementelle Source-Generatoren (seit C# 10)
- Nullable-Annotationen wie [AllowNull], [DisallowNull], [return: NotNullIfNotNull("xy")], [DoesNotReturn], [return: MaybeNull], MaybeNullWhen(bool), NotNullWhen(bool)
- Span<T> / Memory<T> (seit C# 7.2)
- Statische Codeanalyse
- Strukturen auf dem Stack (ref struct) seit C# 7.2
- Unmanaged Constructed Types (seit C# 8.0)

5.5 Programmcodebeispiele zu diesem Buch

Die Programmcodebeispiele zu diesem Buch können Sie auf der auf der von mir ehrenamtlich betriebenen Leserwebsite *www.IT-Visions.de/Leser* herunterladen. Dort müssen Sie sich registrieren. Bei der Registrierung wird ein Losungswort abgefragt. Bitte geben Sie dort das Losungswort **Sloborn** ein.

Alle Programmbeispiele aus diesem Buch sind in einer Visual Studio 2022-Projektmappe mit zwei Projekten enthalten. Es muss seit C# 8.0 zwei Projekte geben, weil einige Sprachfeatures von C# 8.0 nicht mehr im klassischen .NET Framework laufen und C# seit Version 9.0 gar nicht mehr dort läuft. Die beiden Projekte enthalten:

- CSharpSprachsyntax_NETClassic (.NET Framework 4.8): Alle Sprachfeatures von C# 1.0 bis 7.3 und solche von C# 8.0, die auch auf klassischen .NET Framework laufen
- CSharpSprachsyntax_NET (.NET 6.0): Alle Sprachfeatures von C# 8.0, die NICHT auf .NET Framework laufen sowie alle Sprachfeatures **ab** C# 9.0

Die Beispiele sind in Unterordnern nach Sprachversionen aufgeteilt. Dies heißt, dass Sie zum Beispiel Sprachfeatures von C# 9.0 im Ordner CS090 finden bzw. C# 10.0 in CS100.

Wie im Vorwort bereits erwähnt handelt es sich um den Anwendungstyp "Konsolenanwendung". So brauchen Sie als Leser kein Wissen über irgendeine GUI-Bibliothek und die Beispiele sind prägnant fokussiert auf die Syntax. Bitte beachten Sie das nächste Kapitel zum Hilfsklasse "CUI".

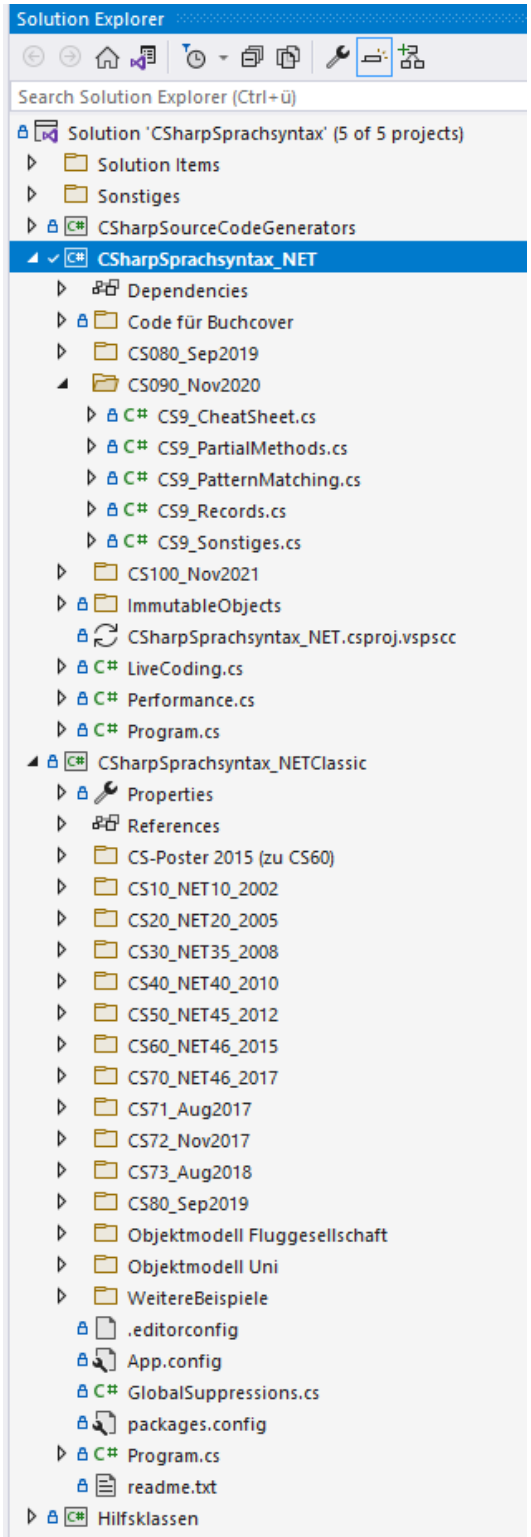

Solution Explorer
Search Solution Explorer (Ctrl+ü)
Solution 'CSharpSprachsyntax' (5 of 5 projects)
Solution Items
Sonstiges
CSharpSourceCodeGenerators
CSharpSprachsyntax_NET
Dependencies
Code für Buchcover
CS080_Sep2019
CS090_Nov2020
CS9_CheatSheet.cs
CS9_PartialMethods.cs
CS9_PatternMatching.cs
CS9_Records.cs
CS9_Sonstiges.cs
CS100_Nov2021
ImmutableObjects
CSharpSprachsyntax_NET.csproj.vspscc
LiveCoding.cs
Performance.cs
Program.cs
CSharpSprachsyntax_NETClassic
Properties
References
CS-Poster 2015 (zu CS60)
CS10_NET10_2002
CS20_NET20_2005
CS30_NET35_2008
CS40_NET40_2010
CS50_NET45_2012
CS60_NET46_2015
CS70_NET46_2017
CS71_Aug2017
CS72_Nov2017
CS73_Aug2018
CS80_Sep2019
Objektmodell Fluggesellschaft
Objektmodell Uni
WeitereBeispiele
.editorconfig
App.config
GlobalSuppressions.cs
packages.config
Program.cs
readme.txt
Hilfsklassen

Abbildung: Programmcodebeispiele zu diesem Buch in zwei Visual Studio-Konsolenanwendungen (EXE) plus Hilfsbibliotheken (DLLs)

5.6 Hilfsklasse zur Konsolenausgabe (CUI)

Für die Bildschirmausgabe an der Konsole wird in diesem Buch oft nicht nur Console.WriteLine() verwendet, sondern auch Hilfsroutinen kommen zur Anwendung, die farbige Bildschirmausgaben erzeugen. Diese Hilfsroutinen sind in der Klasse *ITVisions.CUI* (CUI besteht dabei für Commandline User Interface) implementiert. Diese Klasse ist Teil des NuGet-Pakets ITV.AppUtil…nupkg, welches bei den herunterladbaren Projekten zu diesem Buch in Form mitgeliefert und via <packageSource> in der Datei NuGet.config einbezogen wird.

Dies wichtigsten Hilfsroutinen in der Klasse *CUI* sind im Folgenden zum besseren Verständnis abgedruckt.

Listing: Klasse CUI mit Hilfsroutinen für die Bildschirmausgabe an der Konsole

```
using System;
using System.Runtime.InteropServices;
using System.Web;
using ITVisions.UI;
using System.Diagnostics;

namespace ITVisions
{
 /// <summary>
 /// Helper utilities for console UIs
 /// (C) Dr. Holger Schwichtenberg 2002-2018
 /// </summary>
 public static class CUI
 {
  public static bool IsDebug = false;
  public static bool IsVerbose = false;

  #region Print only under certain conditions
 public static void PrintDebug(object s)
  {
   PrintDebug(s, System.Console.ForegroundColor);
  }

  public static void PrintVerbose(object s)
  {
   PrintVerbose(s, System.Console.ForegroundColor);
  }
  #endregion

  #region Issues with predefined colors
  public static void MainHeadline(string s)
  {
   Print(s, ConsoleColor.Black, ConsoleColor.Yellow);

  }
  public static void Headline(string s)
  {
```

```
 Print(s, ConsoleColor.Yellow);
}
public static void HeaderFooter(string s)
{
 Console.ForegroundColor = ConsoleColor.Green;
 Console.WriteLine(s);
 Console.ForegroundColor = ConsoleColor.Gray;
}

public static void SubHeadline(string s)
{
 Print(s, ConsoleColor.White);
}

public static void PrintSuccess(object s)
{
 Print(s, ConsoleColor.Green);
}

public static void H1(string s)
{
 MainHeadline(s);
}

public static void H2(string s)
{
 Headline(s);
}

public static void H3(string s)
{
 SubHeadline(s);
}

public static void PrintGreen(string s)
{
 Print(s, ConsoleColor.Green);
}

public static void PrintYellow(string s)
{
 Print(s, ConsoleColor.Yellow);
}

public static void PrintRed(string s)
{
 Print(s, ConsoleColor.Red);
}

public static void PrintSuccess(object s)
{
 Print(s, ConsoleColor.Green);
}
```

```
public static void PrintStep(object s)
{
 Print(s, ConsoleColor.Cyan);
}

public static void PrintDebugSuccess(object s)
{
 PrintDebug(s, ConsoleColor.Green);
}

public static void PrintVerboseSuccess(object s)
{
 PrintVerbose(s, ConsoleColor.Green);
}

public static void PrintWarning(object s)
{
 Print(s, ConsoleColor.Cyan);
}

public static void PrintDebugWarning(object s)
{
 PrintDebug(s, ConsoleColor.Cyan);
}

public static void PrintVerboseWarning(object s)
{
 PrintVerbose(s, ConsoleColor.Cyan);
}

public static void PrintError(object s)
{
 Print(s, ConsoleColor.White, ConsoleColor.Red);
}

public static void PrintDebugError(object s)
{
 PrintDebug(s, ConsoleColor.White, ConsoleColor.Red);
}

public static void PrintVerboseError(object s)
{
 Print(s, ConsoleColor.White, ConsoleColor.Red);
}

public static void Print(object s)
{
 PrintInternal(s, null);
}
#endregion

#region Print with selectable color
```

```
  public static void Print(object s, ConsoleColor farbe, ConsoleColor?
hintergrundfarbe = null)
  {
   PrintInternal(s, farbe, hintergrundfarbe);
  }

  public static void PrintDebug(object s, ConsoleColor farbe, ConsoleColor?
hintergrundfarbe = null)
  {
   if (IsDebug || IsVerbose) PrintDebugOrVerbose(s, farbe, hintergrundfarbe);
  }

  public static void PrintVerbose(object s, ConsoleColor farbe)
  {
   if (!IsVerbose) return;
   PrintDebugOrVerbose(s, farbe);
  }
  #endregion

  #region Print with additional data

  /// <summary>
  /// Print with Thread-ID
  /// </summary>
  public static void PrintWithThreadID(string s, ConsoleColor c =
ConsoleColor.White)
  {
   var ausgabe = String.Format("Thread #{0:00} {1:}: {2}",
System.Threading.Thread.CurrentThread.ManagedThreadId,
DateTime.Now.ToLongTimeString(), s);
   CUI.Print(ausgabe, c);
  }

  /// <summary>
  ///  Print with time
  /// </summary>
  public static void PrintWithTime(object s, ConsoleColor c = ConsoleColor.White)
  {
   CUI.Print(DateTime.Now.Second + "." + DateTime.Now.Millisecond + ":" + s);
  }

  private static long count;
  /// <summary>
  /// Print with counter
  /// </summary>
  private static void PrintWithCounter(object s, ConsoleColor farbe,
ConsoleColor? hintergrundfarbe = null)
  {
   count += 1;
   s = $"{count:0000}: {s}";
   CUI.Print(s, farbe, hintergrundfarbe);
  }
```

```
  #endregion

  #region internal helper routines
  private static void PrintDebugOrVerbose(object s, ConsoleColor farbe,
ConsoleColor? hintergrundfarbe = null)
  {
   count += 1;
   s = $"{count:0000}: {s}";
   Print(s, farbe, hintergrundfarbe);
   Debug.WriteLine(s);
   Trace.WriteLine(s);
   Trace.Flush();
  }

  /// <summary>
  /// Output to console, trace and file
  /// </summary>
  /// <param name="s"></param>
  [DebuggerStepThrough()]
  private static void PrintInternal(object s, ConsoleColor? farbe = null,
ConsoleColor? hintergrundfarbe = null)
  {
   if (s == null) return;

   if (HttpContext.Current != null)
   {
    try
    {
 if (farbe != null)
    {
     HttpContext.Current.Response.Write("<span style='color:" +
farbe.Value.DrawingColor().Name + "'>");
    }
    if (!HttpContext.Current.Request.Url.ToString().ToLower().Contains(".asmx")
&& !HttpContext.Current.Request.Url.ToString().ToLower().Contains(".svc") &&
!HttpContext.Current.Request.Url.ToString().ToLower().Contains("/api/"))
HttpContext.Current.Response.Write(s.ToString() + "<br>");

    if (farbe != null)
    {
     HttpContext.Current.Response.Write("</span>");
    }
   }
   catch (Exception)
   {
   }
  }
  else
  {
   object x = 1;
   lock (x)
   {
    ConsoleColor alteFarbe = Console.ForegroundColor;
    ConsoleColor alteHFarbe = Console.BackgroundColor;
```

```
      if (farbe != null) Console.ForegroundColor = farbe.Value;
      if (hintergrundfarbe != null) Console.BackgroundColor =
hintergrundfarbe.Value;

      //if (farbe.ToString().Contains("Dark")) Console.BackgroundColor =
ConsoleColor.White;
      //else Console.BackgroundColor = ConsoleColor.Black;

      Console.WriteLine(s);
      Console.ForegroundColor = alteFarbe;
      Console.BackgroundColor = alteHFarbe;
     }
    }
   }
   #endregion

   #region Set the position of the console window
   [DllImport("kernel32.dll", ExactSpelling = true)]
   private static extern IntPtr GetConsoleWindow();
   private static IntPtr MyConsole = GetConsoleWindow();

   [DllImport("user32.dll", EntryPoint = "SetWindowPos")]
   public static extern IntPtr SetWindowPos(IntPtr hWnd, int hWndInsertAfter, int
x, int Y, int cx, int cy, int wFlags);

   // Set the position of the console window without size
   public static void SetConsolePos(int xpos, int ypos)
   {
    const int SWP_NOSIZE = 0x0001;
    SetWindowPos(MyConsole, 0, xpos, ypos, 0, 0, SWP_NOSIZE);
   }

   // Set the position of the console window with size
   public static void SetConsolePos(int xpos, int ypos, int w, int h)
   {
    SetWindowPos(MyConsole, 0, xpos, ypos, w, h, 0);
   }
   #endregion
  }
}
```

5.7 Qualitätssicherung der Programmcodebeispiele

Ich versichere Ihnen, dass die Programmcodebeispiele auf zwei meiner Entwicklungssysteme kompilierten und liefen, bevor ich sie per Kopieren & Einfügen in das Manuskript zu diesem Buch übernommen habe und auf der Leser-Website zum Download veröffentlicht habe.

Dennoch gibt es leider Gründe, warum die Beispiele bei Ihnen als Leser nicht laufen:

- Eine abweichende Systemkonfiguration (in der heutigen komplexen Welt der vielen Varianten und Versionen von Betriebssystemen und Anwendungen nicht unwahrscheinlich). Es ist einem Autor nicht möglich, alle Konfigurationen durchzutesten.

- Änderungen, die sich seit der Erstellung der Beispiele ergeben haben (von den vielen Breaking Changes, die die neueren .NET-Versionen immer wieder durch Microsoft erhalten, können auch Beispiele betroffen sein, was nicht immer leicht zu entdecken ist)
- Schließlich sind auch menschliche Fehler des Autors möglich. Bitte bedenken Sie, dass das Fachbuchschreiben – wie im Vorwort erwähnt – nur ein Hobby ist. Es gibt nur sehr wenige Menschen in Deutschland, die hauptberuflich als Fachbuchautor arbeiten und so professionell Programmcodebeispiele erstellen und testen können wie kommerziellen (bezahlten) Programmcode.

Falls dennoch Beispiele bei Ihnen nicht laufen, kontaktieren Sie mich bitte unter

www.dotnet-doktor.de/Leserfeedback

mit einer sehr genauen Fehlerbeschreibung. Ich bemühe mich, Ihnen binnen zwei Wochen zu antworten. Im Einzelfall kann es wegen dienstlicher oder privater Abwesenheit aber auch länger dauern.

5.8 Ihre Belohnung, wenn Sie helfen, dieses Buch zu verbessern!

Wenn Sie Fehler in diesem oder einem anderen selbstverlegten Fachbuch (siehe *www.IT-Visions.de/Verlag*) finden, bin ich Ihnen nicht nur wirklich sehr dankbar, sondern Sie bekommen auch eine Belohnung in Form von aktualisierten oder weiteren E-Books.

Fehlerart	E-Book-Guthaben
Inhaltlicher Fehler	Pro Fehler 20 Euro
Sprachlicher Fehler	Pro Fehler 4 Euro

Ein Beispiel: Wenn Sie einen inhaltlichen Fehler und fünf Rechtschreibfehler in diesem Buch finden, dann haben Sie bei mir 40 Euro gut. Dafür können Sie dann eins meiner selbstverlegten Bücher als E-Book bekommen.

Melden Sie die Fehler bitte per Webformular: *www.dotnet-doktor.de/Leserfeedback*

Bitte geben Sie dabei unbedingt nicht nur den Namen des Buchs, sondern auch die Versionsnummer (siehe Impressum) und die genaue Fundstelle (Kapitel, Seitenzahl, Absatz) an.

Schreiben Sie bitte dabei, welches E-Book Sie wünschen. Das Buch schicke ich Ihnen dann per E-Mail zu.

Tipp: Auch Fehler auf meiner persönlichen Website *www.dotnet-doktor.de* und der Firmenwebsite *www.IT-Visions.de* zählen mit!

Ich freue mich auf Ihre Fehlermeldung!

Holger Schwichtenberg

P.S. Falls Sie Ihre Fehlermeldung sich auf eine Ausgabe des Buchs bezieht, die älter als ein Jahr ist und der Fehler in der aktuellsten Ausgabe schon behoben ist, dann zählt das leider nicht.

6 Fakten zu C#

6.1 Der Name C#

C# wird gesprochen „C Sharp“. Das # könnte man auch in ein vierfaches Pluszeichen aufspalten (also C++++, eine Weiterentwicklung von C++). Ursprünglich sollte die Sprache "Cool" heißen. Eine Zeit lang wurde auch "C# .NET" verwendet; das ist heute aber nicht mehr üblich. Microsoft spricht aber gelegentlich noch von "Visual C#", z.B. meldet sich der Kommandozeilencompiler von C# auch in der aktuellen Version mit "Microsoft (R) Visual C# Compiler".

6.2 Ursprünge von C#

C# ist das Ergebnis eines Projektes bei Microsoft, welches im Dezember 1998 gestartet wurde, nachdem die Firma Sun Microsoft die Veränderung der von Sun entwickelten Programmiersprache Java verboten hatte. Vater von C# ist Anders Hejlsberg [*de.wikipedia.org/wiki/Anders_Hejlsberg*], der zuvor auch Turbo Pascal und Borland Delphi erschaffen hat. Er war früher bei Borland und arbeitet seit 1996 bei Microsoft. Heutzutage ist er auch verantwortlich für die Sprache TypeScript.

6.3 .NET als Basis für C#

Die Programmiersprache C# ist sehr eng verbunden mit der Softwareentwicklungsplattform Microsoft .NET. C#-Programmcode läuft immer auf Basis einer .NET-Laufzeitumgebung und benötigt Klassen aus der .NET-Basisklassenbibliothek. So besitzt C# selbst keine Datentypen: Alle Datentypen, die man in C# verwendet, z.B. string, sind in Wirklichkeit Klassen aus der .NET-Basisklassenbibliothek (string → System.String). Auch andere Sprachkonstrukte in C# basieren auf Schnittstellen und Klassen der .NET-Basisklassenbibliothek, z.B. foreach(…) { … } basiert auf der Schnittstelle System.Collections.IEnumerable und await foreach(…) { … } basiert auf System.Collections.Generic.IAsyncEnumerable<T>.

Im Laufe der Geschichte von .NET (seit dem Jahr 2001) gab es zahlreiche Implementierungen von .NET (.NET Framework, Mono, .NET Compact Framework, .NET Framework Client Profile, .NET Micro Framework, Silverlight, XNA, .NET Profile für Windows Runtime, .NET Core, Universal Windows Platform). Derzeit sind noch in signifikantem Umfang in Einsatz:

- .NET Framework
- .NET Core
- Universal Windows Platform (UWP)
- Mono/Xamarin
- .NET ab Version 5.0

Hinweis: Mit .NET 6.0 führt Microsoft diese Implementierungen zu einer einheitlichen Plattform zusammen. Alle anderen Implementierungen werden nicht mehr entwickelt.

Zumindest das ".NET Framework" wird aber noch viele Jahre eine Bedeutung im Markt haben, weil Microsoft dafür zumindest noch Updates im Bereich Fehlerbehebung, Zuverlässigkeit und Sicherheit liefert. Für alle anderen Implementierungen wird auch dieser Support bald enden.

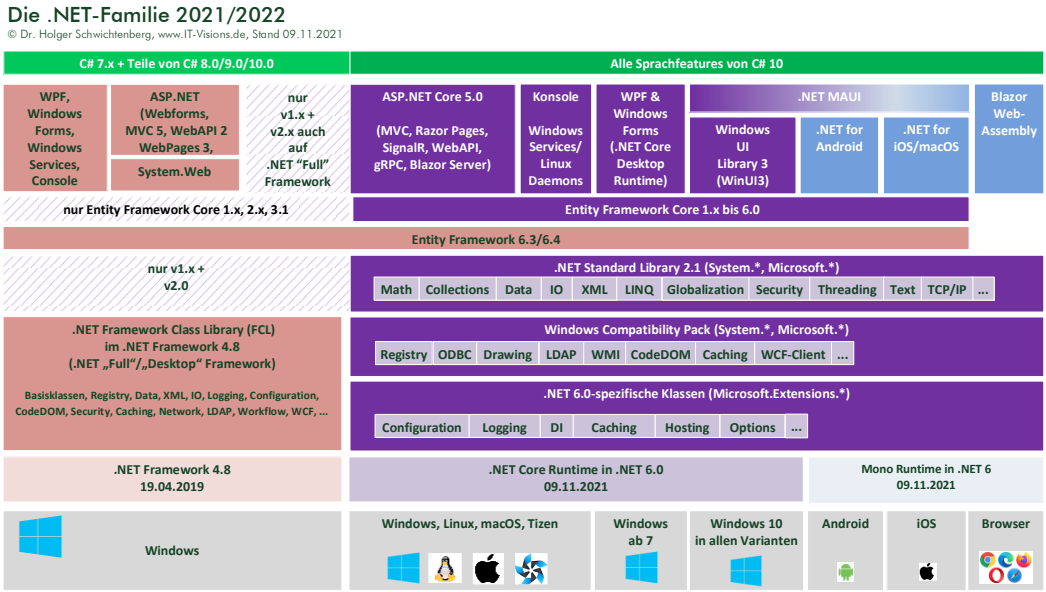


Abbildung: Die .NET-Familie mit .NET Framework 4.8 und .NET 6.0

6.4 Status der Programmiersprache C#

Früher gab es einen wahren Glaubenskrieg in der .NET-Entwicklergemeinde um die Wahl der »richtigen« Programmiersprache. C# oder Visual Basic .NET hieß die Frage, die viele Projektteams bewegt hat. Auch wenn Visual Basic .NET in allen wesentlichen Punkten syntaktisch ebenbürtig war, hat C# klar gewonnen.

C# ist heute nicht nur eine von vielen Programmiersprachen für .NET, es hat sich durchgesetzt als DIE Programmiersprache für .NET. Gegenwärtig gibt es nur noch wenige .NET-Projekte, die Visual Basic .NET, F# oder C++/CLI oder exotischere Sprachen verwenden.

Schaut man in die aktuelle Dokumentation der .NET-Klassen auf *docs.microsoft.com*, sieht man dort nur noch Beispiele für C#, während die alte MSDN-Dokumentation noch Beispiele in C#, Visual Basic .NET, und C++ enthielt.

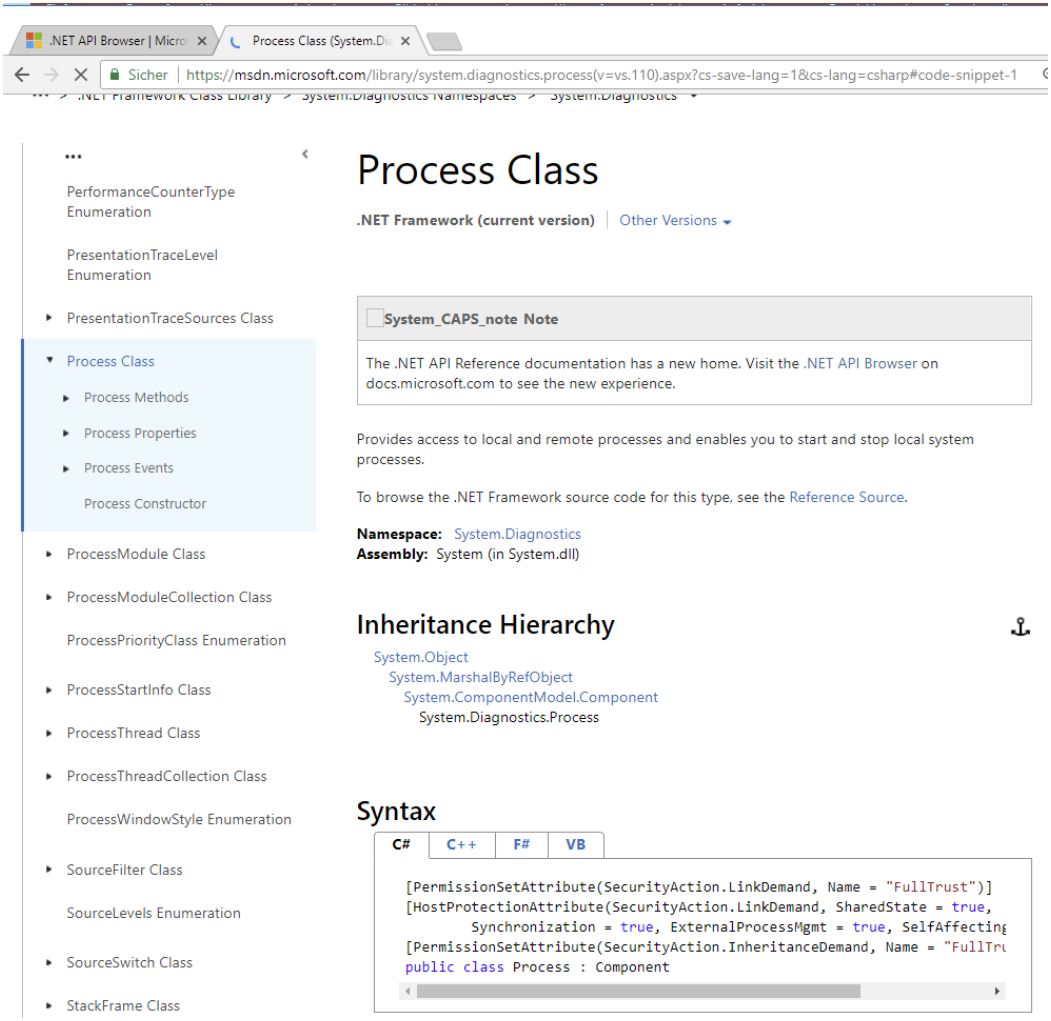


Abbildung: Beispiele in vier Sprachen in der alten MSDN-Dokumentation der .NET-Klassen

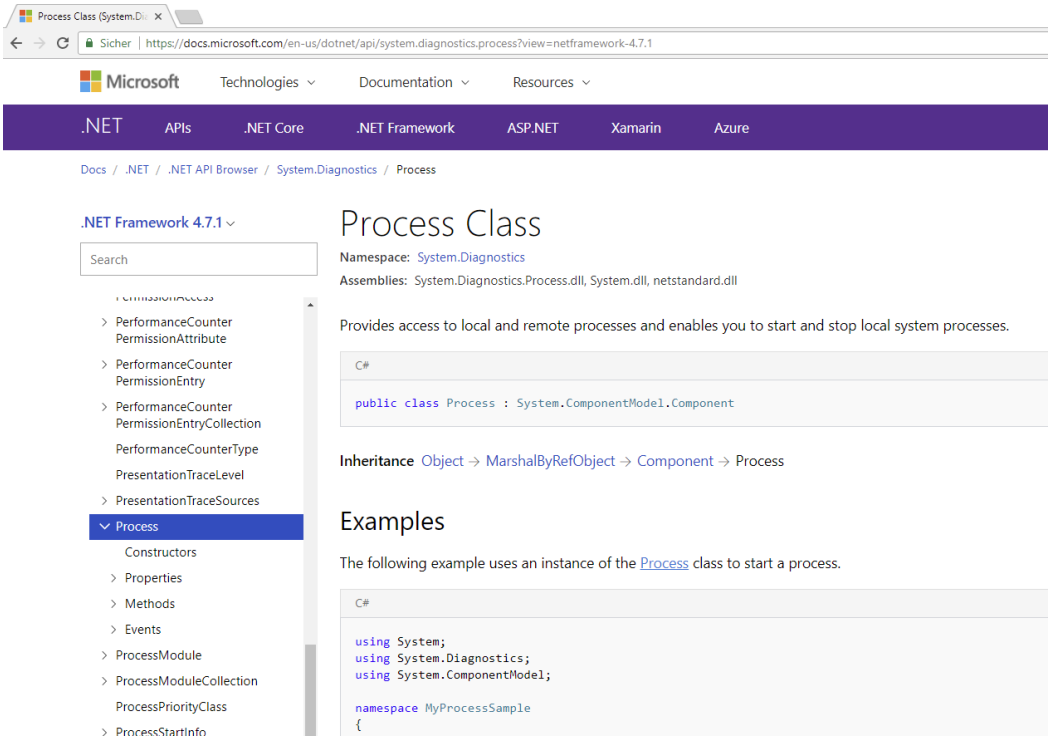


Abbildung: In der neuen .NET-Klassendokumentation gibt es nur noch Beispiele in C#

6.5 Versionsgeschichte

Hinsichtlich der Versionsnummern der Sprache C# herrschte früher etwas Verwirrung. Es gab einerseits eine offizielle Zählung mit Versionsnummer (parallel zum .NET Framework), andererseits mit Jahreszahlen (parallel zu Visual Studio). Intern wird eine dritte Zählung für den Compiler verwendet. Die erste Version von C# im Rahmen des .NET Framework 1.0 trug intern die Versionsnummer 7.0. Zu .NET 1.1 gab es dann C# 7.1, im .NET Framework 2.0 und 3.0 meldet sich der C#-Compiler mit Version 8.0. Ab .NET Framework 3.5 hat Microsoft dies aber bereinigt. Dort meldet sich der Compiler nun auch mit Version 3.5.

Die folgende Liste dokumentiert die Versionsgeschichte von C# einschließlich der verschiedenen Namen, die es jeweils gibt.

- C# 1.0 ist erschienen am 05.01.2002 (in Visual Studio.NET 2002+2003 / .NET Framework 1.0 und 1.1. Erste Version des ISO-Standards für C#.)
- C# 2.0 ist erschienen am 07.11.2005 (C# 2005 / in Visual Studio.NET 2005 / .NET Framework 2.0 und 3.0. Zweite Version des ISO-Standards für C#.)
- C# 3.0 ist erschienen am 15.08.2008 (C# 2008 / in Visual Studio.NET 2008 / .NET Framework 3.5)
- C# 4.0 ist erschienen am 12.04.2010 (C# 2010 / in Visual Studio.NET 2010 / .NET Framework 4.0)
- C# 5.0 ist erschienen am 12.08.2012 (C# 2012 / in Visual Studio.NET 2012 / .NET Framework 4.5)

- C# 6.0 ist erschienen am 20.07.2015 (C# 2015 / in Visual Studio.NET 2015 / .NET Framework 4.6)
- C# 7.0 ist erschienen am 05.03.2017 (C# 2017 / in Visual Studio 2017 v15.0)
- C# 7.1 ist erschienen am 14.08.2017 (in Visual Studio 2017 v15.3)
- C# 7.2 ist erschienen am 15.11.2017 (in Visual Studio 2017 v15.5)
- C# 7.3 ist erschienen am 02.08.2018 (in Visual Studio 2017 v15.7)
- C# 8.0 ist erschienen am 23.09.2019 (in Visual Studio 2019 v16.3)
- C# 9.0 ist erschienen am 10.11.2020 (in Visual Studio 2019 v16.8)
- C# 10.0 ist erschienen am 8.11.2021 (in Visual Studio 2022, v17.0)

Version der Sprachsyntax mit Versionsnummer	**Ausgeliefert mit**	**Version der Sprachsyntax mit Jahreszahl**	**Interne Versionsnummer des C#-Compilers**
C# 1.0	.NET Framework 1.0	Visual C# 2002	7.0 (alter Compiler)
C# 1.1	.NET Framework 1.1	Visual C# 2003	7.1 (alter Compiler)
C# 2.0	.NET Framework 2.0	Visual C# 2005	8.0 (alter Compiler)
C# 2.0	.NET Framework 3.0	Visual C# 2005	8.0 (alter Compiler)
C# 3.0	.NET Framework 3.5	Visual C# 2008	3.5 (alter Compiler)
C# 4.0	.NET Framework 4.0	Visual C# 2010	4.0 (alter Compiler)
C# 5.0	.NET Framework 4.5	Visual C# 2012	4.5 (alter Compiler)
C# 6.0	.NET Framework 4.6 / .NET Core 1.0	Visual C# 2015	1.x (Neuer Compiler)
C# 7.0	Visual Studio 2017 15.0 / .NET Core 2.0	Visual C# 2017	2.0 (Neuer Compiler)
C# 7.1	Visual Studio 2017 15.4 / .NET Core 2.0	Visual C# 2017	2.3 (Neuer Compiler)
C# 7.2	Visual Studio 2017 15.5 / .NET Core 2.0	Visual C# 2017	2.7 (Neuer Compiler)
C# 7.3	Visual Studio 2017 15.7 / .NET Core 2.1	Visual C# 2017	2.8 + 2.9 + 2.10 (Neuer Compiler)
C# 8.0 Preview	Visual Studio 2019 16.0 bis 16.2 / .NET Core 3.0 Preview	Visual C# 2018	3.0 + 3.1 + 3.2 (Neuer Compiler)
C# 8.0 RTM	Visual Studio 2019 16.3 / .NET Core 3.x	Visual C# 2018	3.3 bis 3.7 (Neuer Compiler)
C# 9.0	Visual Studio 2019 16.8 / .NET 5.0	Visual C# 2020	ab v3.8 (Neuer Compiler)

Version der Sprachsyntax mit Versionsnummer	Ausgeliefert mit	Version der Sprachsyntax mit Jahreszahl	Interne Versionsnummer des C#-Compilers
C# 10	Visual Studio 2022 17.0 / .NET 6.0	Visual C# 2022	ab v4.0 (Neuer Compiler)

Tabelle: Verschiedene Versionsnummernzählungen für die Sprache C#

6.6 Standardisierung

Microsoft hat einige Teile des .NET Framework unter dem Namen Common Language Infrastructure (CLI) standardisieren lassen. Die CLI wurde erstmals im Dezember 2001 von der European Computer Manufacturers Association (ECMA) standardisiert (ECMA-Standard 335, Arbeitsgruppe TC49 / TG3, früher: TC39 / TG3, siehe [ECMA01]); mit kleinen Änderungen wurde der Standard im Dezember 2002 von der weltweit wichtigsten Standardisierungsorganisation, der International Standardization Organization (ISO), übernommen als ISO / IEC 23271.

Die Begriffe lauten in den Standards zum Teil allerdings anders als bei Microsoft: Was im .NET Framework Microsoft Intermediate Language (MSIL) heißt, entspricht im Standard der Common Intermediate Language (CIL). Anstelle der Framework Class Library (FCL) spricht man von der CLI Class Library. Von der Standardisierung ausgenommen sind jedoch z.B. die Datenbankschnittstelle ADO.NET und die Benutzeroberflächen-Bibliotheken Windows Forms und ASP.NET Webforms. Auch die neueren .NET-Bibliotheken (WPF, WCF und WF) sind nicht standardisiert.

Auch die Programmiersprache C# ist von beiden Gremien akzeptiert (ECMA-334 bzw. ISO / IEC 23270). Die Standardisierung bezieht sich aber auf ältere Versionen. Die letzten C#-Versionen hat Microsoft nicht mehr standardisieren lassen. Die Standardisierung ist im März 2021 auf dem Stand C# 5.0 [*www.ecma-international.org/publications-and-standards/standards/ecma-334/*].

MICROSOFT VISUAL C# VERSION	CORRESPONDING ECMA STANDARD	CORRESPONDING ISO/IEC STANDARD
V1.0	ECMA-334:2003	ISO/IEC 23270:2003
V2.0	ECMA-334:2006	ISO/IEC 23270:2006
V3.0	none	none
V4.0	none	none
V5.0	ECMA-334:2017	ISO/IEC 23270:2018
V6.0	TBD	TBD
V7.0	TBD	TBD

Abbildung: Standard der C#-Standardisierung [Quelle: www.ecma-international.org/publications-and-standards/standards/ecma-334]

Ein weiterer, von Microsoft initiierter Standard ist von der ECMA im Dezember 2005 unter ECMA-372 (Arbeitsgruppe TC49 / TG5, früher: TC39 / TG5) verabschiedet worden: C++ / CLI ist eine Spracherweiterung für C++ (ISO / IEC 14882:2003), die eine elegantere Nutzung von C++ auf der CLI-Plattform ermöglicht, als dies bisher mit den Managed Extensions for C++ (alias Managed C++) möglich war.

6.7 Implementierung des C#-Compilers

Die ursprüngliche Version des C#-Compilers (csc.exe) wurde in C++ implementiert. Auch der C#-Compiler im Mono-Projekt ist in C++ geschrieben.

Mit dem Projekt "Roslyn" (alias: .NET Compiler Platform) hat Microsoft selbst den Compiler neu in C# implementiert. Die erste Version des neuen Compilers war C# 6.0.

6.8 Open Source

Bereits zu C# 1.0 gab es eine quelloffene Version im Projekt "Rotor" im Rahmen der Standardisierung von C#. Diese war jedoch nicht "Open Source", sondern nur "Shared Source", d.h. der Quellcode durfte betrachtet, aber nicht weiterverwendet werden. Seit C# 6.0 ist der neue Compiler im Rahmen der .NET Compiler Platform "Roslyn" ein Open Source-Projekt auf Github.

Projekt für das Design der Programmiersprache:

github.com/dotnet/csharplang

Projekt für die Implementierung der Programmiersprache:

github.com/dotnet/roslyn

6.9 Parität und Co-Evolution mit Visual Basic .NET

Im Jahr 2010 hatte Microsoft verkündet, die Programmiersprache C# und Visual Basic .NET hinsichtlich ihrer Funktionalität anzugleichen. »Die Sprachen sollen sich in Stil und Gefühl unterscheiden, nicht in ihrem Funktionsumfang«, schrieb Mads Torgersen, Produktmanager für C# damals. Scott Wiltamuth führt den Begriff "Co-Evolution" ein [*blogs.msdn.microsoft.com/scottwil/2010/03/09/vb-and-c-coevolution*].

Einige Jahre hat Microsoft diese Strategie tatsächlich umgesetzt und bestehende Sprachfeatures, die nur eine Sprache hatte, in der anderen Sprache nachgerüstet und neue Sprachfeatures gleichzeitig oder zumindest zeitnah in beiden Sprachen veröffentlicht.

Im Jahr 2017 hat Microsoft sich von Parität und Co-Evolution wieder verabschiedet. Die parallel zu C# 7.0 erschienene Version 15 von Visual Basic .NET bietet daher lediglich Tupel und binäre Literale als neue Sprachfeatures an. Zudem kann Visual Basic .NET 15 C#-Methoden nutzen, die Zeiger mit ref liefern, selbst aber solche Methoden nicht implementieren.

Im März 2020 hat Microsoft verkündet, die Programmiersprache Visual Basic .NET hinsichtlich der Syntax nicht mehr weiter zu entwickeln, die die Sprache aber zumindest bei einigen Projektarten in .NET weiterhin zu unterstützen [*devblogs.microsoft.com/vbteam/visual-basic-in-net-core-3-0/*]. Zentrale Aussagen darin waren:

- "Going forward, we do not plan to evolve Visual Basic as a language."
- "Future features of .NET Core that require language changes may not be supported in Visual Basic. "
- "Due to differences in the platform, there will be some differences between Visual Basic on .NET Framework and .NET Core."

Visual Basic .NET ist dennoch nach C# weiterhin die zweitwichtigste Programmiersprache in der .NET-Welt. Telemetriedaten [*blogs.msdn.microsoft.com/dotnet/2017/02/01/the-net-language-strategy*] von Microsoft zeigen einerseits, dass Visual Basic .NET hauptsächlich zur Programmierung mit älteren .NET-Techniken wie Windows Forms und ASP.NET Webforms zum Einsatz kommt. Andererseits beginnen viele neue .NET-Entwickler mit Visual Basic .NET, bevor sie sich an C# herantrauen.

6.10 Popularität von C#

Für die Beliebtheit von Programmiersprachen gibt es verschiedene Erhebungen. Sehr beliebt ist der Tiobe Index [*www.tiobe.com/tiobe-index*], der monatlich durch eine Auswertung von Internetseiten ermittelt wird. Hier liegt C# in der Regel in der Mitte der Top 10, hinter Java, C, C++ und Python. Knapp hinter C# liegt Visual Basic .NET.

Oct 2021	Oct 2020	Change	Programming Language	Ratings	Change
1	3	↑	Python	11.27%	-0.00%
2	1	↓	C	11.16%	-5.79%
3	2	↓	Java	10.46%	-2.11%
4	4		C++	7.50%	+0.57%
5	5		C#	5.26%	+1.10%
6	6		Visual Basic	5.24%	+1.27%
7	7		JavaScript	2.19%	+0.05%
8	10	↑	SQL	2.17%	+0.61%
9	8	↓	PHP	2.10%	+0.01%
10	17	⇈	Assembly language	2.06%	+0.99%
11	19	⇈	Classic Visual Basic	1.83%	+1.06%
12	14	↑	Go	1.28%	+0.13%
13	15	↑	MATLAB	1.20%	+0.08%
14	9	⇊	R	1.20%	-0.79%
15	12	↓	Groovy	1.18%	-0.05%
16	13	↓	Ruby	1.12%	-0.05%
17	16	↓	Swift	1.11%	+0.02%
18	37	⇈	Fortran	1.08%	+0.70%
19	11	⇊	Perl	0.93%	-0.49%
20	22	↑	Delphi/Object Pascal	0.93%	+0.22%

Abbildung: Beliebtheit der Programmiersprachen (Quelle: www.tiobe.com/tiobe-index)

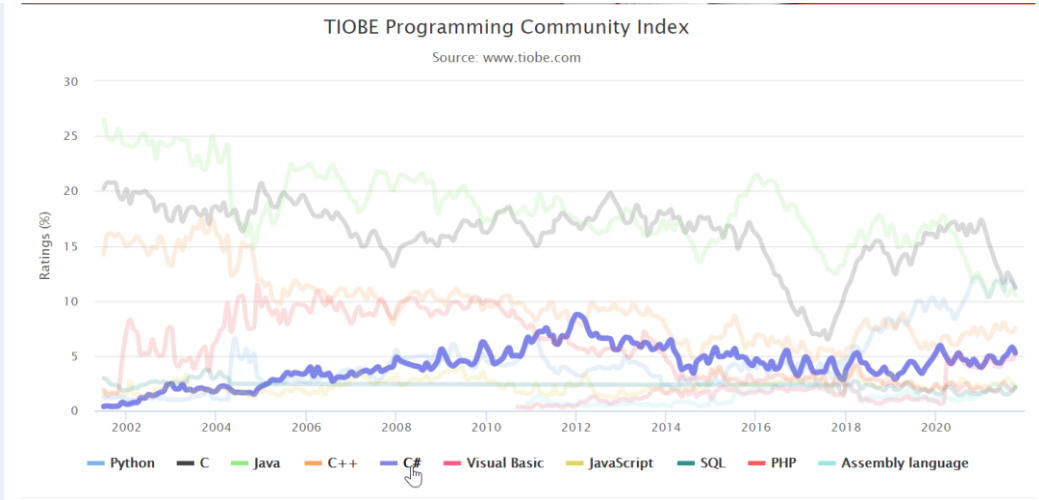


Abbildung: Beliebtheit von C# von 2002 bis 2021 (Quelle: www.tiobe.com/tiobe-index)

Das Ranking der IEEE (Institute of Electrical and Electronics Engineers) basiert auf der Auswertung mehrerer Datenquellen (CareerBuilder, GitHub, Google, Hacker News, IEEE, Reddit, Stack Overflow und Twitter).

Rank	Language	Type	Score
1	Python		100.0
2	Java		95.4
3	C		94.7
4	C++		92.4
5	JavaScript		88.1
6	C#		82.4
7	R		81.7
8	Go		77.7
9	HTML		75.4
10	Swift		70.4
11	Arduino		68.4
12	Matlab		68.3
13	PHP		68.0
14	Dart		67.7
15	SQL		65.0

Abbildung: IEEE Ranking 2021 [spectrum.ieee.org/top-programming-languages/#toggle-gdpr]

Beim IEEE-Ranking kann man nach Einsatzgebieten filtern. C# liegt so:

- Web: Platz 4 hinter Python, Java und JavaScript
- Enterprise: Platz 5 hinter Python, Java, C und C++
- Mobile: Platz 4 hinter Java, C und C++
- Embedded: Platz 4 hinter Python, C und C++
- Alle: Platz 6

Auch das IT-Marktforschungsunternehmen RedMonk liefert ein Programmiersprachenranking basierend auf GitHub und Stackoverflow.com. C# liegt dort zusammen mit C++ und CSS auf Platz 5. Davor sind JavaScript, Python, Java und PHP.

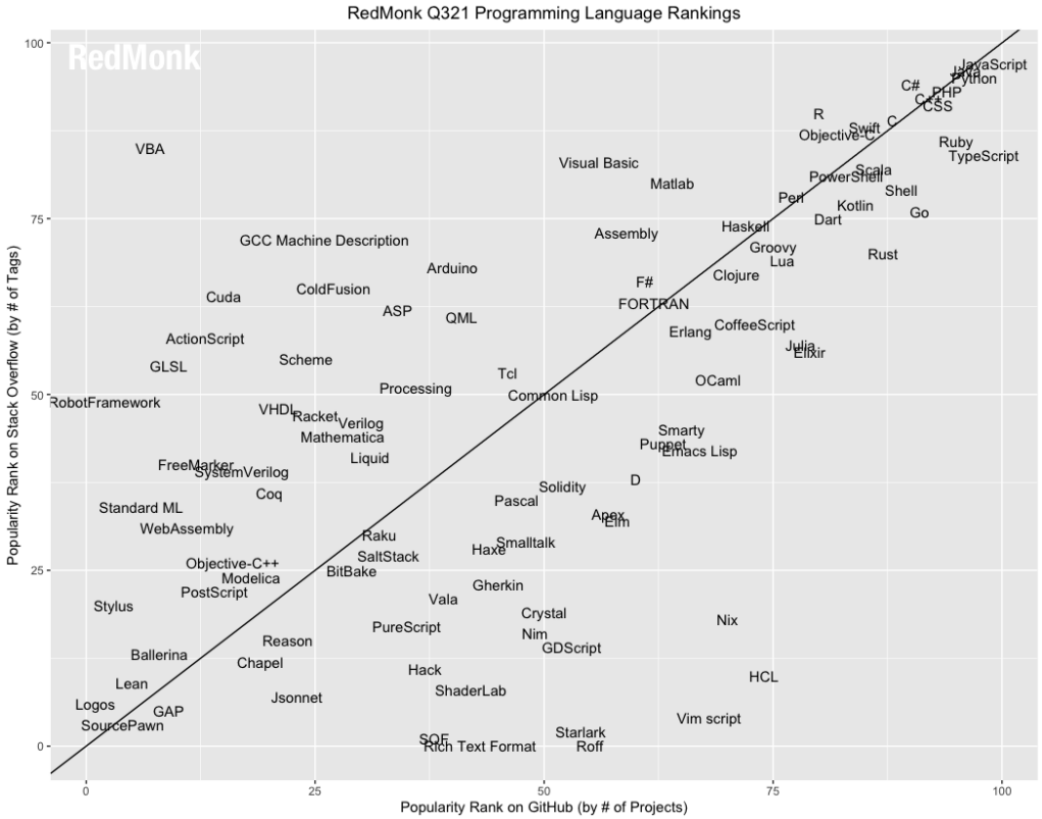


Abbildung: Programmiersprachen-Ranking von RedMonk, drittes Quartal 2021: Diagramm korreliert GitHub-Pull-Requests (x-Achse) zum Rang bei Stack Overflow (y-Achse) [redmonk.com/sogrady/2021/08/05/language-rankings-6-21]

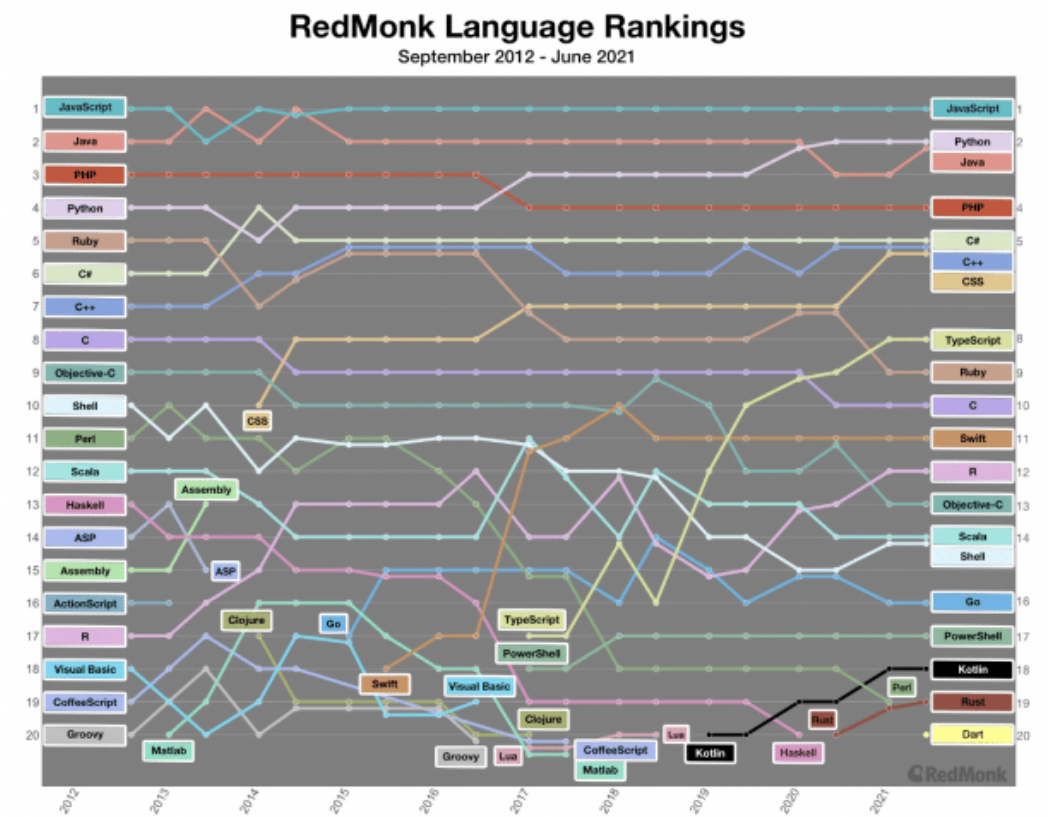


Abbildung: Jahresauswertungen von RedMonk 2012 bis 2021 [redmonk.com/sogrady/2021/08/05/language-rankings-6-21/]

Eine weitere viel beachtete Statistik ist die jährliche Umfrage von Stackoverflow.com. In der Jahresumfrage 2020 (2019, 2018) war C#

- Auf Platz 7 (7, 8) der am meisten eingesetzten Programmiersprachen mit 31,4 % (31,9%, 35,35%)
- Auf Platz 8 (10, 8) der beliebtesten Programmiersprachen mit 59,7% (67,0%, 60,4%)
- Auf Platz 18 (in 2019 und 2020) in der Liste der 25 gefürchtetsten Programmiersprachen (nicht unter den Top 25 in 2018)

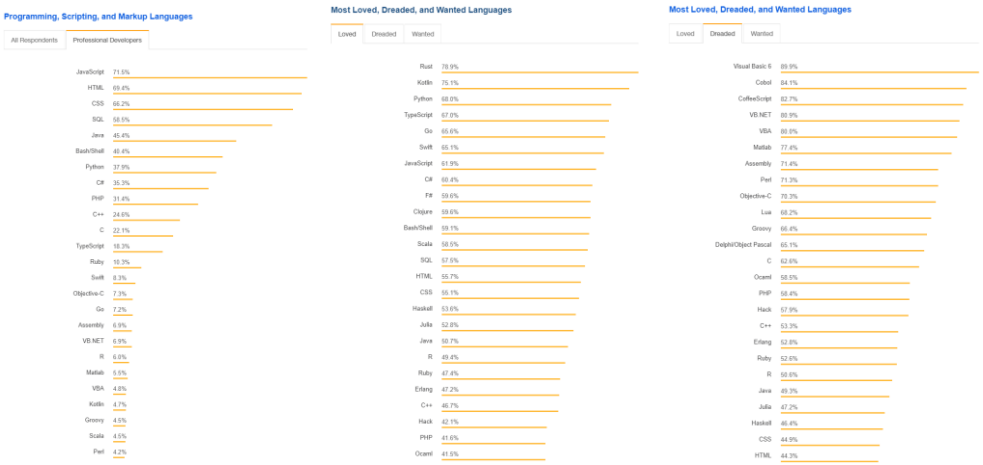


Abbildung: C# in der Jahresumfrage 2018 von stackoverflow.com [insights.stackoverflow.com/survey/2018]

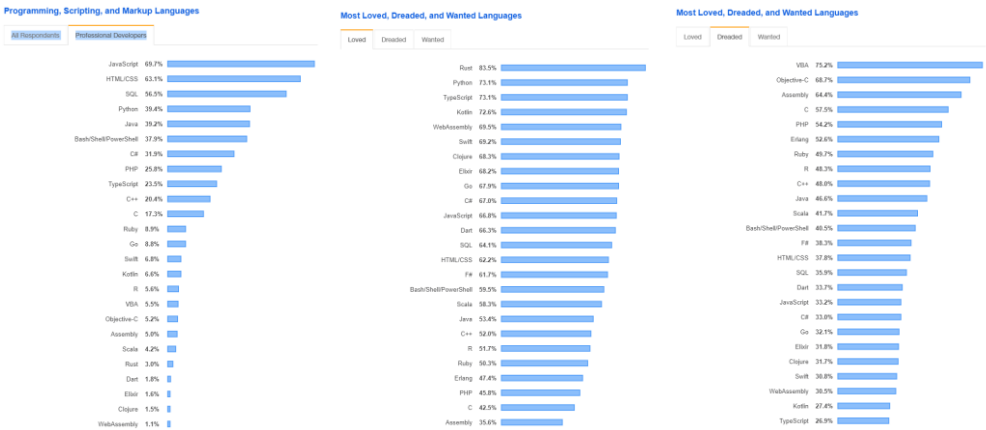


Abbildung: C# in der Jahresumfrage 2019 von stackoverflow.com [insights.stackoverflow.com/survey/2019]

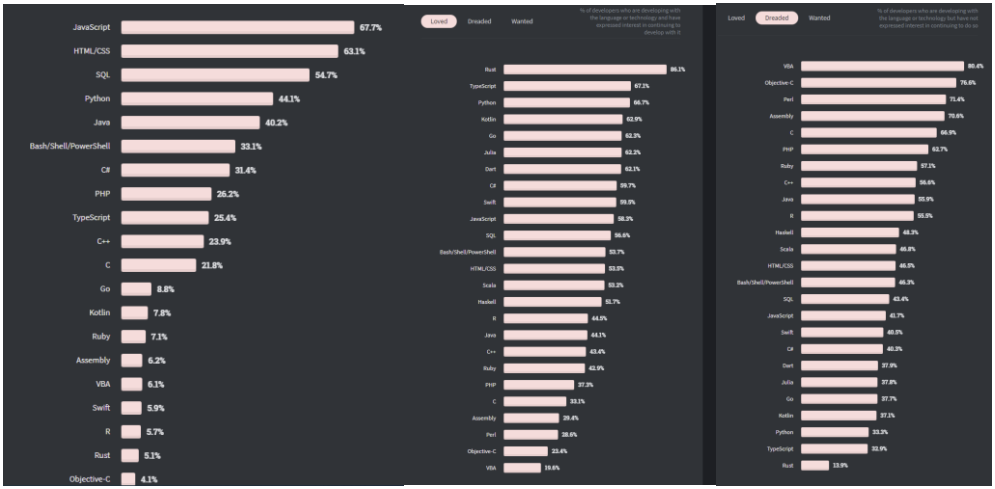


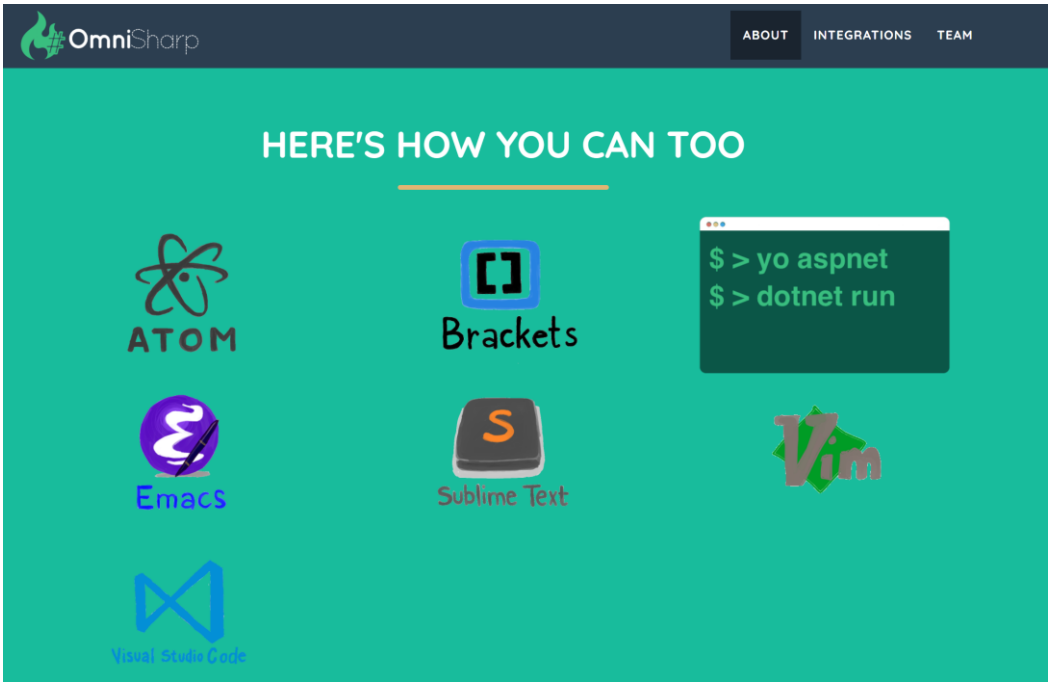
Abbildung: C# in der Jahresumfrage 2020 von stackoverflow.com [insights.stackoverflow.com/survey/2020]

6.11 Editoren für C#

Microsoft liefert für C# selbst drei Editoren:

- Visual Studio: nur für Windows. Kostenfreie Community-Version nur für Open Source-Projekte, Freiberufler und kleine Unternehmen.
  visualstudio.microsoft.com/de/downloads
- Visual Studio for Mac: kostenfrei (Nachfolger des früheren Xamarin Studio)
  visualstudio.microsoft.com/de/vs/mac
- Visual Studio Code: kostenfrei für Windows, macOS und Linux.
  code.visualstudio.com
  C+#-Erweiterungen muss installiert sein! Diese beinhaltet aber nicht alle Werkzeuge aus de, großen Visual Studio, z.B. keine grafischen UI-Designer
  marketplace.visualstudio.com/items?itemName=ms-dotnettools.csharp

Zudem liefert Microsoft mit OmniSharp [*www.omnisharp.net*] eine Basis für die Integration in anderen (plattformneutrale) Editoren wie ATOM, Brackets, Emacs, Sublime und Vim (siehe Abbildung). Hier wird nicht nur Syntax-Farbeinfärbung, sondern auch Eingabeunterstützung (IntelliSense) angeboten. Auch die Visual Studio Code-Erweiterung für C# basiert auf OmniSharp.



Es gibt weitere einfache Editoren, die für die C#-Syntax nur Einfärbung, aber keine Eingabeunterstützung bieten.

Einen weiteren professionellen C#-Editor mit viele Eingabeunterstützung und Refactoring-Funktionen liefert die Firma JetBrains mit ihrem Produkt "Rider" (kostenpflichtig, *www.jetbrains.com/rider*).

6.12 Neuerungen in C# 9.0

Dieses Kapitel bleibt auch in der C# 10-Version des Buchs erhalten, weil viele Unternehmen erst jetzt vom klassisches .NET Framework mit C# 8 auf die moderne .NET-Welt mit C# 10 umsteigen und daher die Neuerungen in C# 9.0 erst jetzt nutzen können.

Die fertige Version von C# 9.0 ist am 10.11.2020 im Rahmen von .NET 5.0 und Visual Studio 2019 v16.8 erschienen.

> **Hinweise:** C# 9.0 wird offiziell von Microsoft nur ab .NET 5.0 unterstützt ("C# 9.0 is supported only on .NET 5 and newer versions." [*docs.microsoft.com/en-us/dotnet/csharp/language-reference/configure-language-version*]. Man kann allerdings die meisten (aber nicht alle!) C# 9.0-Sprachfeatures auch in .NET Core, .NET Framework und Xamarin nutzen. Dazu muss man die <LangVersion> in der Projektdatei erhöhen. Dies wird im Kapitel "Erste C#-Schritte/Festlegen der Compilerversion" beschrieben.
>
> Notwendige Visual Studio-Version für C# 9.0 ist Visual Studio 2019 v16.8 oder höher.

Die wichtigsten Neuerungen in C# 9.0 sind:

- Record-Typen → siehe Kapitel "Record-Typen"
- Programme ohne Main() → Siehe Kapitel "Top-Level Statements"

- Properties, die nach Initialisierung unveränderlich sind (Init Only Properties mit Init Only Setters) → Siehe Kapitel "Attribute/Properties, die nach Initialisierung unveränderlich sind"
- Verwendung des Operators *new* ohne Typangabe (Target-Typed New Expression) → Siehe Kapitel "Klassendefinition/Instanzierung mit dem Operator new")
- Aufhebung der Restriktionen für partielle Methoden → Siehe Kapitel "Partielle Methoden"
- Statische anonyme Funktionen und Discard-Variablen in Lambdas → Siehe Kapitel "Lambda-Ausdrücke"
- Annotationen auf lokale Funktionen → Siehe Kapitel "Lokale Funktion"
- Erweiterung des Pattern Matching → Siehe Kapitel "Verzweigungen/Pattern Matching"
- Modul-Initialisierer → Siehe Kapitel "Modul-Initialisierer".
- Source Code-Generatoren: Mit diesen neuen Code-Generatoren kann ein Entwickler zusätzlichen Programmcode zur Kompilierungszeit erzeugen, der zusammen mit dem eigentlichen Programmcode kompiliert wird. Damit kann man z.B. Annotationen eine Bedeutung geben. → Siehe Kapitel "Source Code-Generatoren".

6.13 Neuerungen in C# 10.0

C# 10.0 ist zusammen mit Visual Studio 2022 und .NET 6 am 8.11.2021 erschienen.

Hinweise: C# 10.0 wird offiziell von Microsoft nur ab .NET 6.0 unterstützt ("C# 10.0 is supported only on .NET 6 and newer versions." [*docs.microsoft.com/en-us/dotnet/csharp/language-reference/configure-language-version*]. Man kann allerdings die meisten (aber nicht alle!) C# 10.0-Sprachfeatures auch in .NET Core, .NET Framework und Xamarin nutzen. Dazu muss man die <LangVersion> in der Projektdatei auf "10.0" erhöhen. Dies wird im Kapitel "Erste C#-Schritte/Festlegen der Compilerversion" beschrieben.

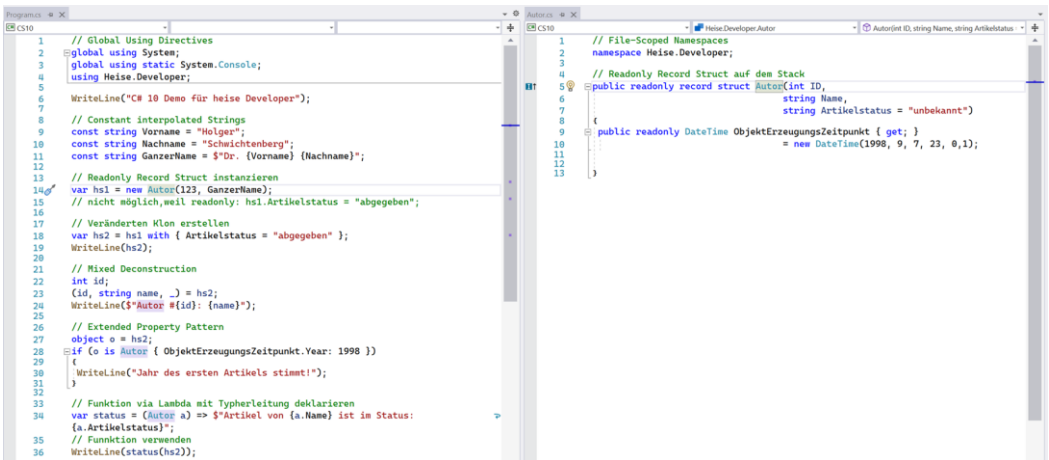
Notwendige Visual Studio-Version für C# 9.0 ist Visual Studio 2022 v17.0 oder höher. Eine Verwendung von C# 10.0 sowohl mit Visual Studio for Mac 2022 als auch einer aktuellen Version von Visual Studio Code und anderen OmniSharp-kompatiblen Editoren [*www.omnisharp.net*] ist möglich.

C# 10.0 - .NET 6 and Visual Studio 2022 version 17.0

- Record structs and `with` expressions on structs (`record struct Point(int X, int Y);` , `var newPoint = point with { X = 100 };`).
- Global using directives: `global using` directives avoid repeating the same `using` directives across many files in your program.
- Improved definite assignment: definite assignment and nullability analysis better handle common patterns such as `dictionary?.TryGetValue(key, out value) == true` .
- Constant interpolated strings: interpolated strings composed of constants are themselves constants.
- Extended property patterns: property patterns allow accessing nested members (`if (e is MethodCallExpression { Method.Name: "MethodName" })`).
- Sealed record ToString: a record can inherit a base record with a sealed `ToString` .
- Incremental source generators: improve the source generation experience in large projects by breaking down the source generation pipeline and caching intermediate results.
- Mixed deconstructions: deconstruction-assignments and deconstruction-declarations can be blended together (`(existingLocal, var declaredLocal) = expression`).
- Method-level AsyncMethodBuilder: the AsyncMethodBuilder used to compile an `async` method can be overridden locally.
- #line span directive: allow source generators like Razor fine-grained control of the line mapping with `#line` directives that specify the destination span (`#line (startLine, startChar) - (endLine, endChar) charOffset "fileName"`).
- Lambda improvements: attributes and return types are allowed on lambdas; lambdas and method groups have a natural delegate type (`var f = short () => 1;`).
- Interpolated string handlers: interpolated string handler types allow efficient formatting of interpolated strings in assignments and invocations.
- File-scoped namespaces: files with a single namespace don't need extra braces or indentation (`namespace X.Y.Z;`).
- Parameterless struct constructors: support parameterless constructors and instance field initializers for struct types.
- CallerArgumentExpression: this attribute allows capturing the expressions passed to a method as strings.

Abbildung: Übersicht über die Neuerungen in C# 10
Quelle: Microsoft
[github.com/dotnet/csharplang/blob/main/Language-Version-History.md]

Das folgende Bild realisiert das kleine Kunststück, fast alle neuen C# 10-Sprachfeatures in 2 überschaubare und kommentierte Listings unterzubringen, die zusammen auch noch Sinn machen. Verstehen Sie dies als Kurzreferenz. Natürlich finden Sie eine ausführliche Beschreibung in den verschiedenen Kapiteln dieses Buchs.



```
// Global Using Directives
global using System;
global using static System.Console;
using Heise.Developer;

WriteLine("C# 10 Demo für heise Developer");

// Constant interpolated Strings
const string Vorname = "Holger";
const string Nachname = "Schwichtenberg";
const string GanzerName = $"Dr. {Vorname} {Nachname}";

// Readonly Record Struct instanzieren
var hs1 = new Autor(123, GanzerName);
// nicht möglich,weil readonly: hs1.Artikelstatus = "abgegeben";

// Veränderten Klon erstellen
var hs2 = hs1 with { Artikelstatus = "abgegeben" };
WriteLine(hs2);

// Mixed Deconstruction
int id;
(id, string name, _) = hs2;
WriteLine($"Autor #{id}: {name}");

// Extended Property Pattern
object o = hs2;
if (o is Autor { ObjektErzeugungsZeitpunkt.Year: 1998 })
{
 WriteLine("Jahr des ersten Artikels stimmt!");
}

// Funktion via Lambda mit Typherleitung deklarieren
var status = (Autor a) => $"Artikel von {a.Name} ist im Status:
{a.Artikelstatus}";
// Funnktion verwenden
WriteLine(status(hs2));
```

```
// File-Scoped Namespaces
namespace Heise.Developer;

// Readonly Record Struct auf dem Stack
public readonly record struct Autor(int ID,
                                    string Name,
                                    string Artikelstatus = "unbekannt")
{
public readonly DateTime ObjektErzeugungsZeitpunkt { get; }
                                    = new DateTime(1998, 9, 7, 23, 0,1);


}
```

Abbildung: Fast alle neuen C# 10.0-Features auf einen Blick.

Sie finden in diesem Buch:

- Kapitel "Datentypen": Neuerungen zu Interpolated Strings

- Kapitel "Verzweigungen/ Pattern Matching": Neuerungen zum Pattern Matching
- Kapitel "Methoden": Caller Argument Expressions
- Kapitel "Namensräume": Alle Neuerungen zu den Namensräumen (File-Scoped Namespaces, Global Using Directives, Implicit Using Directives)
- Kapitel "Record-Typen": Alle Neuerungen zu Record-Typen (record class, record struct, sealed ToString())
- Kapitel "Strukturen/With-Ausdrücke": Einsatz von Klonen mit with bei Strukturen und anonymen Typen.
- Kapitel "Strukturen/Strukturen mit parameterlosem Konstruktor": Strukturen mit parameterlosem Konstruktor
- Kapitel "Tupel": Mixed Deconstruction
- Kapitel "Funktionale Programmierung/Lambda-Ausdrücke": Typherleitung, explizite Rückgabetypen und Annotationen/Attribute für Lambda-Ausdrücke

6.14 Blick in die Zukunft

Die kommende Version C# 11.0 soll im November 2022 zusammen mit .NET 7 erscheinen. Vorschläge für künftige Sprachversionen finden Sie unter

github.com/dotnet/csharplang/tree/main/proposals

Jedermann kann Vorschläge für neue Sprachfeatures einreichen; die Hürden zur Annahme sind aber recht hoch.

Die Liste der Sprachfeatures, an denen Microsoft aktiv arbeitet, findet man unter

github.com/dotnet/roslyn/blob/main/docs/Language%20Feature%20Status.md

C# Next

Feature	Branch	State	Developer	Reviewer	LDM Champ
nameof(parameter)	main	In Progress	jcouv	TBD	jcouv
Relax ordering of `ref` and `partial` modifiers	ref-partial	In Progress	alrz	gafter	jcouv
Parameter null-checking	param-nullchecking	In Progress	RikkiGibson, fayrose	cston, chsienki	jaredpar
Generic attributes	generic-attributes	Merged into 17.0p4 (preview langver)	AviAvni	RikkiGibson, jcouv	mattwar
Default in deconstruction	decon-default	Implemented	jcouv	gafter	jcouv
List patterns	list-patterns	In Progress	alrz	jcouv, 333fred	333fred
Raw string literals	RawStringLiterals	In Progress	CyrusNajmabadi	jcouv	CyrusNajmabadi
Semi-auto-properties	semi-auto-properties	In Progress	Youssef1313	TBD	CyrusNajmabadi
Required properties	required-properties	In Progress	333fred	TBD	333fred
Top Level statement attribute specifiers	main-attributes	In Progress	chsienki	TBD	jaredpar
Primary Constructors	primary-constructors	In Progress	TBD	TBD	MadsTorgersen
Params Span + Stackalloc any array type	params-span	In Progress	cston	TBD	jaredpar
Newlines in interpolations	main	In Progress	CyrusNajmabadi	jcouv, TBD	CyrusNajmabadi

Abbildung: Für C# 11.0 geplante Sprachfeatures

Sprachfeatures, die sich bereits in der Entwicklung befinden aber noch nicht Teil des Sprachcompilers sind, können Sie ausprobieren auf dieser Website:

sharplab.io

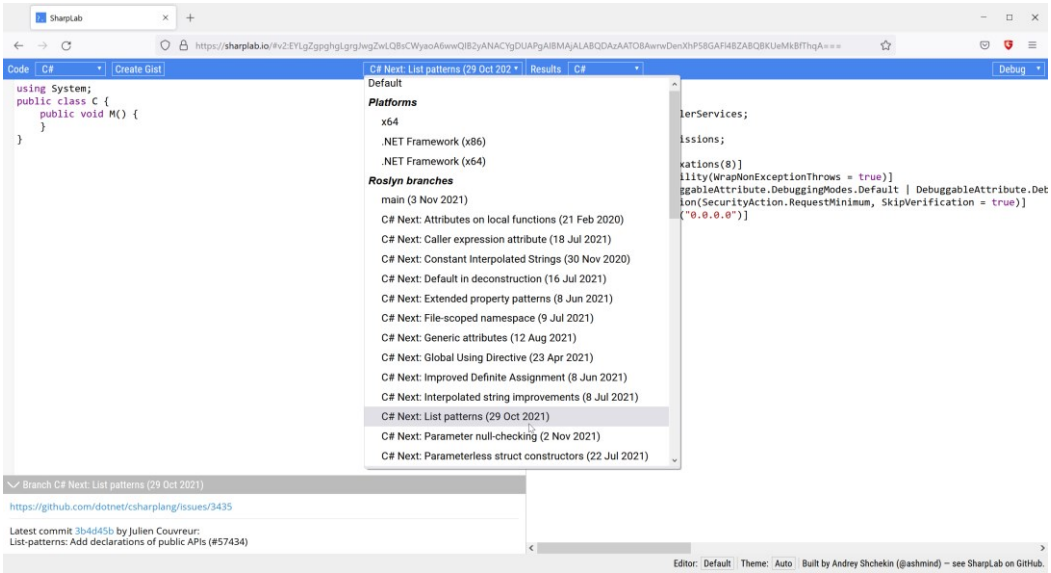


Abbildung: www.sharplab.io