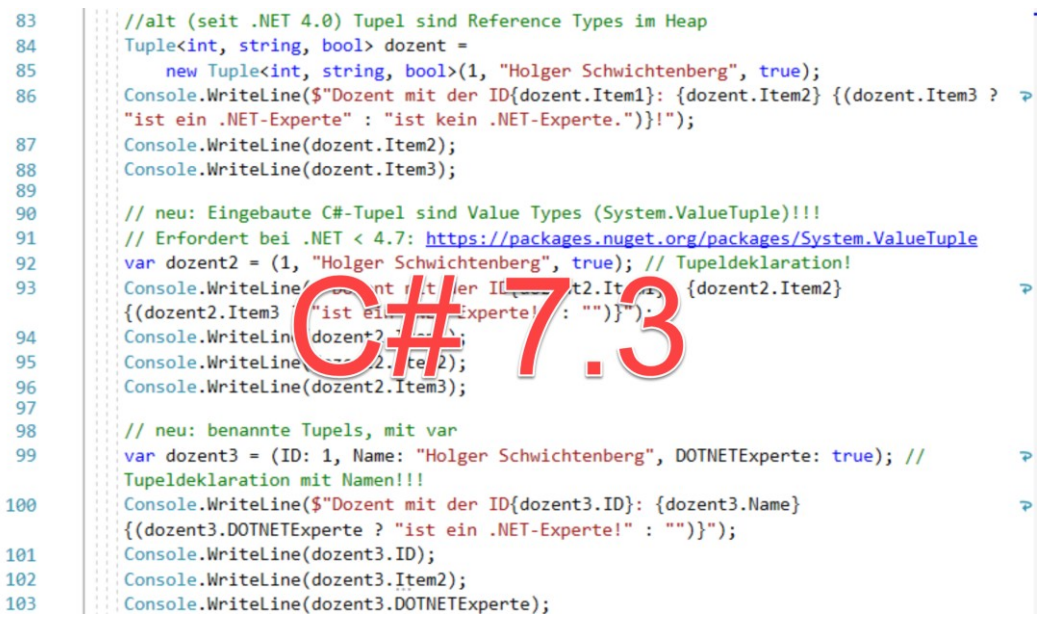
Dr. Holger Schwichtenberg

C# 7.3 Crashkurs

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




```
83      //alt (seit .NET 4.0) Tupel sind Reference Types im Heap
84      Tuple<int, string, bool> dozent =
85          new Tuple<int, string, bool>(1, "Holger Schwichtenberg", true);
86      Console.WriteLine($"Dozent mit der ID{dozent.Item1}: {dozent.Item2} {(dozent.Item3 ?
        "ist ein .NET-Experte" : "ist kein .NET-Experte.")}!");
87      Console.WriteLine(dozent.Item2);
88      Console.WriteLine(dozent.Item3);
89
90      // neu: Eingebaute C#-Tupel sind Value Types (System.ValueTuple)!!!
91      // Erfordert bei .NET < 4.7: https://packages.nuget.org/packages/System.ValueTuple
92      var dozent2 = (1, "Holger Schwichtenberg", true); // Tupeldeklaration!
93      Console.WriteLine(
        {(dozent2.Item3 ? "ist ein .NET-Experte!" : "")}");
94      Console.WriteLine(dozent2.Item2);
95      Console.WriteLine(dozent2.Item2);
96      Console.WriteLine(dozent2.Item3);
97
98      // neu: benannte Tupels, mit var
99      var dozent3 = (ID: 1, Name: "Holger Schwichtenberg", DOTNETExperte: true); //
        Tupeldeklaration mit Namen!!!
100     Console.WriteLine($"Dozent mit der ID{dozent3.ID}: {dozent3.Name}
        {(dozent3.DOTNETExperte ? "ist ein .NET-Experte!" : "")}");
101     Console.WriteLine(dozent3.ID);
102     Console.WriteLine(dozent3.Item2);
103     Console.WriteLine(dozent3.DOTNETExperte);
```

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

Verlag:	www.IT-Visions.de, Fahrenberg 40b, D-45257 Essen
ISBN:	3934279-31-7
Version:	2.1.2 / 29.08.2019
Sprachliche Korrektur:	Katrin Lettau und Matthias Bloch
Formatierung:	Katrin Lettau
Bezugsquelle:	https://www.amazon.de/exec/obidos/ASIN/3934279317/itvisions-21
Bezugsquelle:	https://leanpub.com/CSharp73Crashkurs

1 Inhaltsverzeichnis

2 Vorwort

Liebe Leserinnen und Leser,

der "C# Crashkurs" ist ein prägnanter Überblick über die Syntax der Programmiersprache C# in der aktuellen Version 7.3.

Dieses Büchlein ist geeignet für Softwareentwickler, die von einer anderen objektorientierten Programmiersprache (z.B. C++, Java, Visual Basic .NET 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 hervorgehoben.

Für Neueinsteiger, die mit C# erstmals eine objektorientiere Programmiersprache erlernen wollen, ist es nicht geeignet.

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

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

Dieses Büchlein wird vertrieben über Amazon.de

- Kindle-E-Book von Amazon.de für 9,99 Euro (der Autor erhält 5,56 Euro): *www.amazon.de/exec/obidos/ASIN/B07G2STYMH/itvisions-21*
- Gedruckt (Print-on-Demand) bei Amazon.de für 14,99 Euro (der Autor erhält 5,53 Euro): *www.amazon.de/exec/obidos/ASIN/3934279317/itvisions-21*
- PDF bei leanpub.com für 10,99 Dollar (der Autor erhält ca. 8,93 Euro): *www.leanpub.com/CSharp8*

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 Büchlein geben wird. Ich werde dann an diesem Büchlein arbeiten, wenn ich neben meinem Beruf als Softwarearchitekt, Berater und Dozent und meinen sportlichen Betätigungen noch etwas Zeit für das Fachbuchautorenhobby übrig habe.

Zudem möchte ich darauf hinweisen, dass ich natürlich keinen kostenfreien technischen Support zu den Inhalten dieses Büchleins geben kann. Ich freue mich aber immer über konstruktives Feedback und Verbesserungsvorschläge. Bitte verwenden Sie dazu das Kontaktformular auf *www.dotnet-doktor.de*.

Wenn Sie **technische Hilfe** zu C# und seinen Einsatzgebieten (.NET, Mono, Xamarin) oder anderen Themen rund um Visual Studio, Windows oder andere Microsoft-Produkte benötigen, stehe ich Ihnen im Rahmen meiner beruflichen Tätigkeit für die Firmen *www.IT-Visions.de* (Beratung, Schulung, Support) und 5Minds IT-Solutions GmbH & Co KG (Softwareentwicklung, siehe *www.5minds.de*) gerne zur Verfügung. Bitte wenden Sie sich für ein Angebot an das jeweilige Kundenteam.

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

Herzliche Grüße aus Essen, dem Herzen der Metropole Ruhrgebiet

Holger Schwichtenberg

3 Über den Autor

- Studienabschluss Diplom-Wirtschaftsinformatik an der Universität Essen
- Promotion an der Universität Essen im Gebiet komponentenbasierter Softwareentwicklung
- Seit 1996 selbstständig als unabhängiger Berater, Dozent, Softwarearchitekt und Fachjournalist
- Fachlicher Leiter des Berater- und Dozententeams bei *www.IT-Visions.de*
- Leitung der Softwareentwicklung im Bereich Microsoft/.NET bei der 5Minds IT-Solutions GmbH & Co. KG (*www.5Minds.de*)
- Über 65 Fachbücher beim Carl Hanser Verlag, bei O'Reilly, Microsoft Press, APress und Addison-Wesley sowie mehr als 1000 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. Microsoft TechEd, Microsoft Summit, Microsoft IT Forum, BASTA, BASTA-on-Tour, .NET Architecture Camp, Advanced Developers Conference, Developer Week, OOP, 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)
  - Microsoft Certified Solution Developer (MCSD)
- Thematische Schwerpunkte:
  - Softwarearchitektur, mehrschichtige Softwareentwicklung, Softwarekomponenten, SOA
  - Visual Studio, Continous Integration, Continous Delivery, Azure DevOps
  - Microsoft .NET Framework, C#, Visual Basic
  - .NET-Architektur/Auswahl von .NET-Technologien
  - Einführung von .NET Framework und Visual Studio/Migration auf .NET
  - Webanwendungsentwicklung und Cross-Plattform-Anwendungen mit HTML, ASP.NET, JavaScript/TypeScript und Webframeworks wie Angular
  - Enterprise .NET, verteilte Systeme/Webservices mit .NET, insbesondere Windows Communication Foundation und WebAPI
  - Relationale Datenbanken, XML, Datenzugriffsstrategien
  - Objektrelationales Mapping (ORM), insbesondere ADO.NET Entity Framework und EF Core
  - Windows PowerShell, PowerShell Core und Windows Management Instrumentation (WMI)
- Ehrenamtliche Community-Tätigkeiten:
  - Vortragender für die International .NET Association (INETA)
  - Betrieb diverser Community-Websites: www.dotnet-lexikon.de, www.dotnetframework.de, www.windows-scripting.de, www.aspnetdev.de u. a.
- Firmenwebsites: *http://www.IT-Visions.de* und *http://www.5Minds.de*
- Weblog: *http://www.dotnet-doktor.de*
- Kontakt für geschäftliche Anfragen via Kundenteam:
  E-Mail **kundenteam@IT-Visions.de** sowie Telefon **0201 / 64 95 90 - 0**
- Kontakt für Feedback zu diesem Buch: Kontaktformular auf *http://www.dotnet-doktor.de*

4 Über dieses Büchlein

4.1 Versionsgeschichte dieses Büchleins

Die folgende Tabelle zeigt die Versionen, die von diesem Büchlein erschienen sind, sowie die darin besprochenen C#-Versionen.

Ergänzungen der Versionsnummer an der dritten Stelle (z.B. 1.2.3) sind kleine Korrekturen im Büchlein, die nicht explizit in dieser Versionstabelle erscheinen.

Buchversion Datum	**Umfang**	**Preis Kindle-Ausgabe**	**Preis gedruckte Ausgabe**	**C#-Version**	**Bemerkung**
1.0 27.03.2018	166 Seiten	9,99 €	14,99 €	7.2	▪ Grundversion
1.1 20.07.2018	167 Seiten	9,99 €	14,99 €	7.2 (7.3)	▪ Ref Local Reassignment (C# 7.3) ▪ Ausblick auf C# 8.0
2.0 21.07.2018	172 Seiten	9,99 €	14,99 €	7.3 (8.0)	▪ Vergleich mit Tupeln (C# 7.3) ▪ Annotationen für Backing Field von Auto-Properties (C# 7.3) ▪ Verbesserungen für unsafe-Blöcke (C# 7.3) ▪ Ranges (C# 8.0)
2.1 27.11.2018	189 Seiten	9,99 €	14,99 €	7.3 (8.0)	▪ Kapitel "Grundkonzepte von C#" erweitert ▪ Kapitel "Attribute (Fields und Properties)" erweitert ▪ Kapitel "Ereignisse" überarbeitet ▪ Kapitel "Funktionale Programmierung in C#" erweitert

					▪ Kapitel "Behandlung von null" ergänzt

4.2 Geplante Themen

Folgende Themen sind für kommenden Ausgaben dieses Büchleins geplant:

- Span<T> / Memory<T> (C# 7.2)
- Aliase für referenzierte Assemblies
- IDisposable
- Indexer
- Design Pattern in C#
- Clean Code-Programmierung mit C#
- Weitere Neuerungen in C# 8.0

4.3 Programmcodebeispiele zu diesem Büchlein

Die Programmcodebeispiele zu diesem Büchlein 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 **The Orville** ein.

Alle Programmbeispiele aus diesem Büchlein sind in einem Visual Studio 2017-Projekt enthalten, organisiert und in Unterordnern nach Sprachversionen aufgeteilt. Dies heißt, dass Sie zum Beispiel Sprachfeatures von C# 7.0 im Ordner CS70 finden.

Wie im Vorwort bereits erwähnt handelt es sich um den Anwendungstyp "Konsolenanwendung". So brauchen Sie als Leser kein Wissen über irgendeine GUI-Bibliohek und die Beispiele sind prägnant fokussiert auf die Syntax.

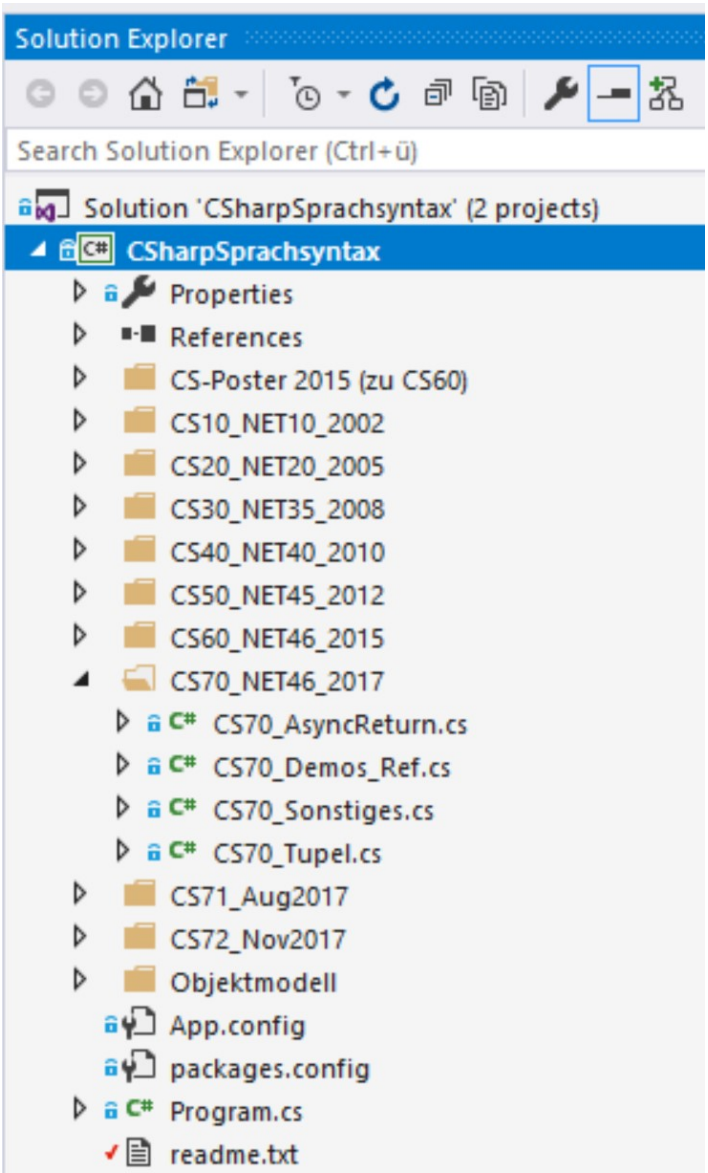


Abbildung: Programmcodebeispiele zu diesem Büchlein in einem Visual Studio-Projekt

5 Fakten zu C#

5.1 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".

5.2 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 Heljsberg [https://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.

5.3 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 *https://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.

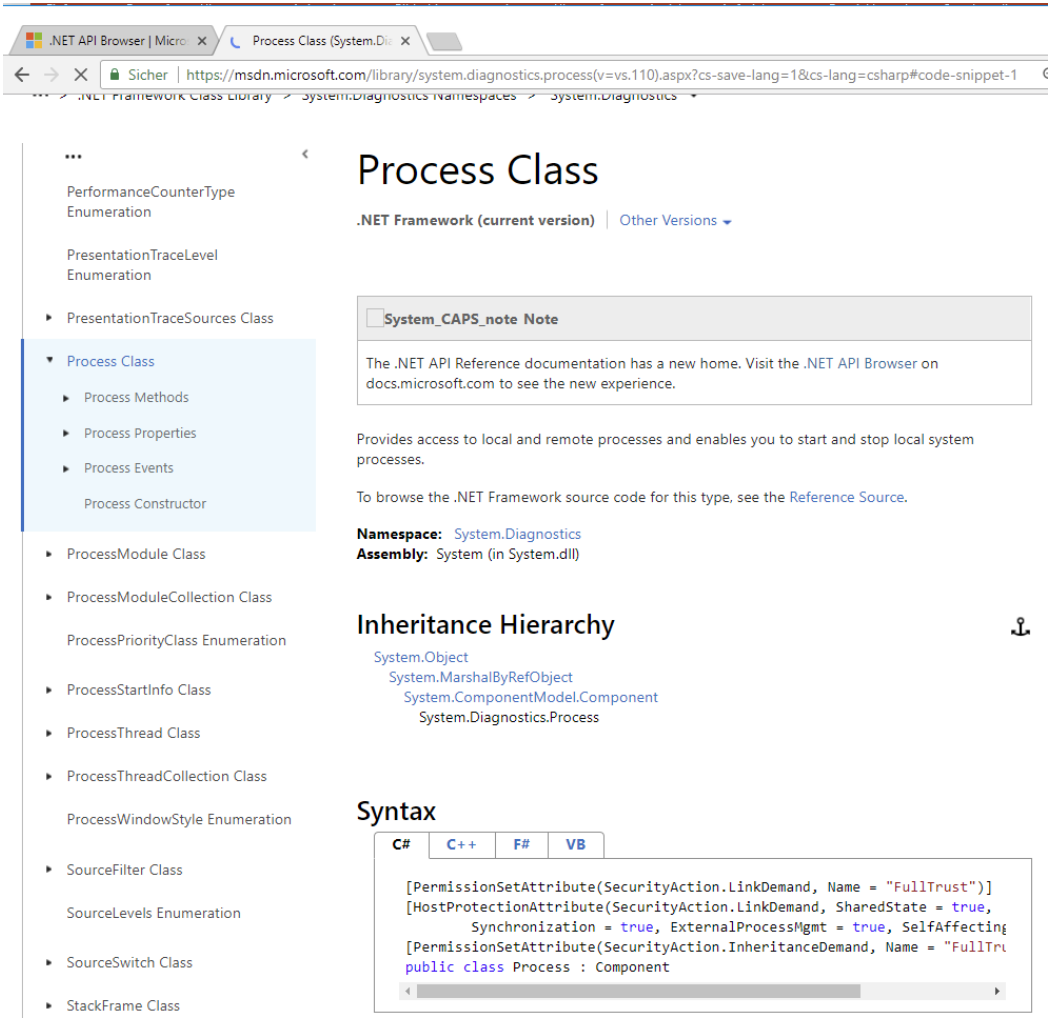


Abbildung: Beispiele in vier Sprachen in der alten MSDN-Dokumentation der .NET-Klassen

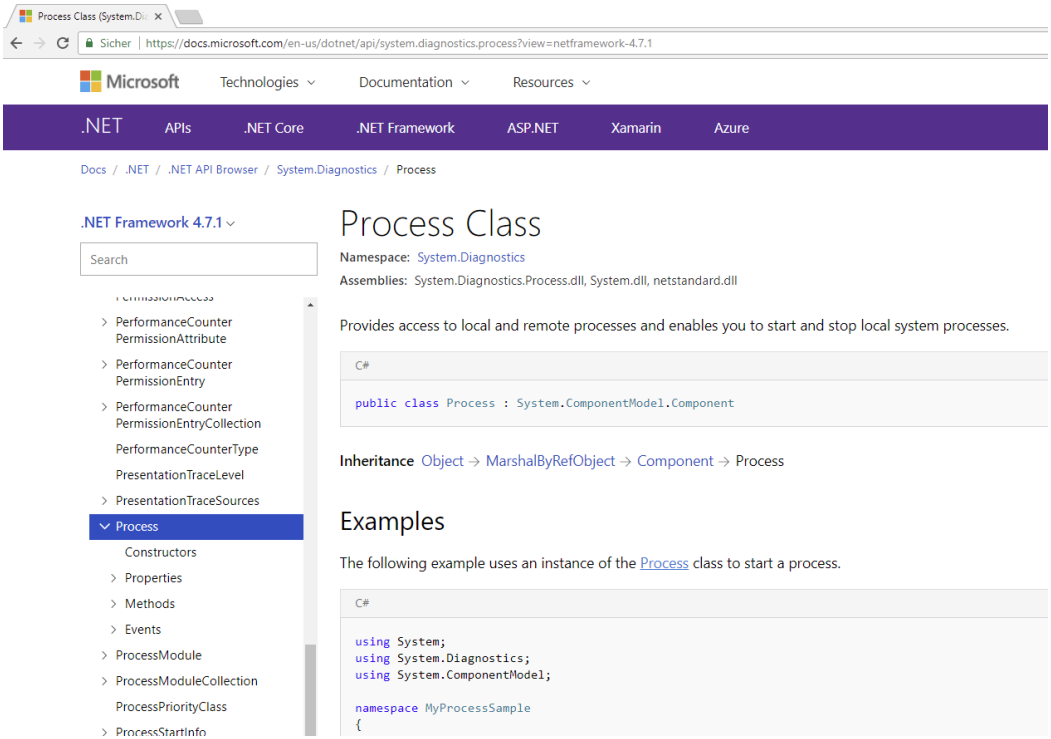


Abbildung: In der neuen .NET-Klassendokumentation gibt es nur noch Beispiele in C#

5.4 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)
- C# 7.1 ist erschienen am 14.08.2017 (in Visual Studio 15.3)
- C# 7.2 ist erschienen am 15.11.2017 (in Visual Studio 15.5)
- C# 8.0 Beta ist erschienen am 04.12.2018 (in Visual Studio 16.0 Preview 1)

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	Visual Studio 2019 16.0 / .NET Core 3.0	Visual C# 2018	2.11 (Neuer Compiler)

Tabelle: Verschiedene Versionsnummernzählungen für die Sprache C#

5.5 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 auf dem Stand C# 2.0

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.

5.6 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.

5.7 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:

https://github.com/dotnet/csharplang

Projekt für die Implementierung der Programmiersprache:

https://github.com/dotnet/roslyn

5.8 Kommende Versionen

Aktuell entwickelt Microsoft an der Version C# 8.0.

5.9 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 [https://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.

Visual Basic .NET ist nach C# die zweitwichtigste Programmiersprache in der .NET-Welt. Telemetriedaten [https://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. Microsoft nahm diese Erkenntnisse zum Anlass, von der im Jahr 2010 verkündigen Co-Evolutionsstrategie von C# und Visual Basic .NET abzurücken und zukünftig nicht mehr alle neuen C#-Features automatisch auf Visual Basic .NET zu übertragen. 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.

6 Grundkonzepte von C#

Konzeptionell wurde C# vor allem von C++ und Java beeinflusst; man kann aber auch Parallelen zu Visual Basic und Delphi finden.

6.1 Sprachtypus

Im Gegensatz zu C++, das eine hybride Sprache aus objektorientierten und nicht-objektorientierten Konzepten ist, ist C# ebenso wie Java eine rein objektorientierte Sprache, d.h., alle Datentypen basieren auf Klassen und alle Anweisungen erfolgen in Klassen.

C# unterstützt alle zentralen Konzepte der Objektorientierung einschließlich Schnittstellen, Vererbung und Polymorphismus. Schon in C# 2005 wurde auch die Unterstützung für generische Klassen und partielle Klassen hinzugefügt. Außerdem besitzt C# Konzepte der funktionalen Programmierung (Delegates und Lambda-Ausdrücke).

6.2 Groß- und Kleinschreibung

Ein wesentlicher Unterschied zwischen C# und Visual Basic .NET ist die Tatsache, dass C# im Gegensatz zu Visual Basic .NET zwischen Groß- und Kleinschreibung unterscheidet. Dies gilt sowohl für die Schlüsselwörter der Sprache als auch für alle Bezeichner (a und A sind verschiedene Variablen!). Die Schlüsselwörter der Sprache C# werden komplett in Kleinbuchstaben geschrieben.

6.3 Schlüsselwörter der Sprache

Die folgende Liste zeigt die vordefinierten Schlüsselwörter der Programmiersprache C#. Diese Namen dürfen in der gleichen Groß-/Kleinschreibung nicht als Bezeichner verwendet werden (Quelle: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/index).

abstract	as	base	bool	break
byte	case	catch	char	checked
class	const	continue	decimal	default
delegate	do	double	else	enum
event	explicit	extern	false	finally
fixed	float	for	foreach	goto
if	implicit	in	int	interface
internal	is	lock	long	namespace
new	null	object	operator	out
override	params	private	protected	public
readonly	ref	return	sbyte	sealed
short	sizeof	stackalloc	static	string
struct	switch	this	throw	true
try	typeof	uint	ulong	unchecked
unsafe	ushort	using	virtual	void
volatile	while			

6.4 Namensregeln und Namenskonventionen

Bei der Vergabe von eigenen Bezeichner (z.B. Variablenname, Parameternamen, Atributnamen und Methodennamen) gibt es verpflichtende Regeln und optionale Namenskonventionen.

Verpflichtende Regeln sind:

- Der Name darf nur Buchstaben (*), Zahlen und den Unterstrich enthalten.
- Der Name muss mit einem Buchstaben beginnen
- Die Groß- und Kleinschreibung ist relevant
- Es dürfen keine Namen von C#-Schlüsselwörtern verwendet werden.

Hinweis: (*) Umlaute sind erlaubt, aber sollten dennoch besser vermieden werden: Nicht alle Werkzeuge und alle Menschen kommen damit gut klar!

Optionale Regeln hat Microsoft in den ".NET Framework Design Guidelines" [https://docs.microsoft.com/en-us/dotnet/standard/design-guidelines] definiert. Die wichtigsten Regeln dort sind:

- Für die Groß-/Kleinschreibung gilt grundsätzlich **PascalCasing**, d.h. ein Bezeichner beginnt grundsätzlich mit einem Großbuchstaben und jedes weitere Wort innerhalb des Bezeichners beginnt ebenfalls wieder mit einem Großbuchstaben.

  Beispiel: KundenPortalBenutzer

- Ausnahmen gibt es für Abkürzungen, die nur aus zwei Buchstaben bestehen. Diese dürfen komplett in Großbuchstaben geschrieben sein (z.B. UI und IO). Alle anderen Abkürzungen werden entgegen ihrer normalen Schreibweise in Groß-/Kleinschreibung geschrieben (z.B. Xml, Xsd und W3c).

  Beispiele: System.IO.File, System.Xml.XmlDocument

- Lokale Variablen, versteckte Attribute (private/protected) und Parameternamen sollen in **camelCasing** (Bezeichner beginnt mit einem Kleinbuchstaben, aber jedes weitere Wort innerhalb des Bezeichners beginnt mit einem Großbuchstaben) geschrieben werden.

  Beispiel: Login(KundenPortalBenutzer kundenPortalBenutzer)

6.5 Blockbildung und Umbrüche

Blockbildung findet im C / C++-Stil statt, also mit geschweiften Klammern { }. Befehlstrenner ist das Semikolon (;).

Ein Zeilenumbruch kann zwischen den Elementen des Ausdrucks auftreten, ohne das besondere Vorkehrungen getroffen werden müssen. Zahlen können seit C# 7.0 mit einem Unterstrich gegliedert werden; aber man darf innerhalb von Zahlen keinen Zeilenumbruch haben.

```
// Formel ohne Umbrüche
double Ergebnis1 = (2 + 3) * ( 5 + 6) * (7 * 8) + 3.141_592_653_59;

// Formel mit Umbrüchen
double Ergebnis2 = (2 + 3) *
                   (5 + 6) *
                   (7 * 8)
                   + 3.141_592_653_59;
```

6.6 Hello World

Das folgende Listing zeigt das Hello World-Beispiel in C#, das man in jeder Programmiersprache zuerst schreibt.

```
using System;

namespace HalloWelt
{
 class Program
 {
  static void Main(string[] args)
  {
   Console.WriteLine("Hallo Welt!");
  }
 }
}
```

Mariginal komplexer ist diese Variante, die – sofern vorhanden – den ersten übergebenen Kommandozeilenparameter als Name auffasst und die Person mit Namen grüßt.

```
namespace HalloWelt
{
 class Program
 {
  static void Main(string[] args)
  {
   if (args.Length > 0)
   {
    var name = args[0];
    // Ausgabe mit String Interpolation
    Console.WriteLine($"Hallo {name}!");
    Console.ReadLine();
   }
   else
   {
    Console.WriteLine("Hallo Welt!");
   }
  }
 }
}
```

6.7 Eingebaute Funktionen

Anders als in Visual Basic existieren in C# keine eingebauten Funktionen zur Typumwandlung (z.B. CBool(), CInt(), CLng(), CType()), Zeichenkettenverarbeitung (z.B. InStr(), Trim(), LCase()) und Ausgabe (z.B. MsgBox()). Auch die My-Klassenbibliothek ist nicht vorhanden. Grundsätzlich ist es möglich, die in Visual Basic eingebauten Funktionen und die My-Bibliothek durch Referenzierung der Microsoft.VisualBasic.dll auch in C# zu nutzen. Dies sollte jedoch vermieden werden, um sprachunabhängig zu bleiben. Alle Visual Basic-Funktionen und -Objekte sind auch in der .NET-Klassenbibliothek enthalten, z.B. String.IndexOf() statt InStr() und Convert.ToInt32() statt CInt().

7 Der C#-Compiler

Es gibt zwei Varianten des C#-Compilers: eine alte, in C++ geschriebene, und neue, in C# geschriebene Implementierung.

7.1 Der ursprüngliche (alte) C#-Compiler

Der Kommandozeilencompiler für C# im .NET Framework Redistributable ist csc.exe. Er wird installiert im Verzeichnis C:\Windows\Microsoft.NET\Framework64\v4.0.30319. Alternativ kann er in der .NET Framework-Klassenbibliothek im sogenannten "CodeDOM" durch die Klasse Microsoft.CSharp.CSCodeProvider angesprochen werden.

Wenn Sie heute ein aktuelles Microsoft .NET Framework (z.B. 4.7.2) verwenden, so ist dort der ursprüngliche C#-Compiler immer noch in der Version 5.0 enthalten.

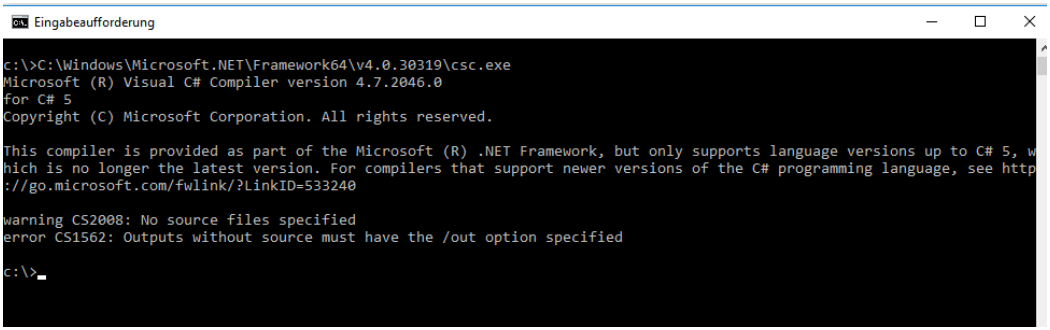


Abbildung: In .NET Framework 4.7.1 ist der C#-Compiler für C# 5.0 enthalten.

7.1.1 Kompilierung mit csc.exe

Der Befehl

```
csc.exe Dateiname1.cs Dateiname2.cs DateinameX.cs
```

oder

```
csc Dateiname1.cs Dateiname2.cs DateinameX.cs
```

übersetzt die angegebenen Dateien in eine Konsolenanwendung. Eine Datei, die als Konsolenanwendung oder Windows-Anwendung kompiliert wird, muss genau eine Klasse mit folgendem Einstiegspunkt besitzen: public static void Main().

Listing: »Hello World« in C#

```
class Hauptprogramm
{
        public static void Main()
        {
                System.Console.WriteLine("Hello World!");
        }
}
```

7.1.2 Kommandozeilenparameter

Der Kommandozeilencompiler bietet zahlreiche Optionen. Die wichtigsten davon sind:

- /target:winexe Der Compiler erzeugt eine Windows-Anwendung
- /target:library Der Compiler erzeugt eine DLL (kein Main() notwendig)

- /r:Dateiliste Die angegebenen Assemblys werden referenziert
- /out:Dateiname Name der Ausgabedatei
- /doc:Dateiname Der Compiler erzeugt zusätzlich eine XML-Dokumentationsdatei
- /help Anzeige der Hilfe zu den Compiler-Optionen
- Anders als beim Visual Basic .NET-Compiler vbc.exe müssen die Optionen /target und /out bei csc.exe vor den Namen der Quelldateien in der Parameterliste erscheinen.

Es folgt die komplette Liste der Kommandozeilenparameter des alten C#-Compilers

```
                        Visual C# Compiler Options

                        - OUTPUT FILES -
/out:<file>                     Specify output file name (default: base name of
file with main class or first file)
/target:exe                     Build a console executable (default) (Short form:
/t:exe)
/target:winexe                  Build a Windows executable (Short form: /t:winexe)
/target:library                 Build a library (Short form: /t:library)
/target:module                  Build a module that can be added to another
assembly (Short form: /t:module)
/target:appcontainerexe         Build an Appcontainer executable (Short form:
/t:appcontainerexe)
/target:winmdobj                Build a Windows Runtime intermediate file that is
consumed by WinMDExp (Short form: /t:winmdobj)
/doc:<file>                     XML Documentation file to generate
/platform:<string>              Limit which platforms this code can run on: x86,
Itanium, x64, arm, anycpu32bitpreferred, or anycpu. The default is anycpu.

                        - INPUT FILES -
/recurse:<wildcard>             Include all files in the current directory and
subdirectories according to the wildcard specifications
/reference:<alias>=<file>       Reference metadata from the specified assembly
file using the given alias (Short form: /r)
/reference:<file list>          Reference metadata from the specified assembly
files (Short form: /r)
/addmodule:<file list>          Link the specified modules into this assembly
/link:<file list>               Embed metadata from the specified interop assembly
files (Short form: /l)

                        - RESOURCES -
/win32res:<file>                Specify a Win32 resource file (.res)
/win32icon:<file>               Use this icon for the output
/win32manifest:<file>           Specify a Win32 manifest file (.xml)
/nowin32manifest                Do not include the default Win32 manifest
/resource:<resinfo>             Embed the specified resource (Short form: /res)
/linkresource:<resinfo>         Link the specified resource to this assembly
(Short form: /linkres)
                                Where the resinfo format is <file>[,<string
name>[,public|private]]

                        - CODE GENERATION -
/debug[+|-]                     Emit debugging information
/debug:{full|pdbonly}           Specify debugging type ('full' is default, and
enables attaching a debugger to a running program)
```

```
/optimize[+|-]                  Enable optimizations (Short form: /o)

                        - ERRORS AND WARNINGS -
/warnaserror[+|-]               Report all warnings as errors
/warnaserror[+|-]:<warn list>   Report specific warnings as errors
/warn:<n>                       Set warning level (0-4) (Short form: /w)
/nowarn:<warn list>             Disable specific warning messages

                        - LANGUAGE -
/checked[+|-]                   Generate overflow checks
/unsafe[+|-]                    Allow 'unsafe' code
/define:<symbol list>           Define conditional compilation symbol(s) (Short
form: /d)
/langversion:<string>           Specify language version mode: ISO-1, ISO-2, 3, 4,
5, or Default

                        - SECURITY -
/delaysign[+|-]                 Delay-sign the assembly using only the public
portion of the strong name key
/keyfile:<file>                 Specify a strong name key file
/keycontainer:<string>          Specify a strong name key container
/highentropyva[+|-]             Enable high-entropy ASLR

                        - MISCELLANEOUS -
@<file>                         Read response file for more options
/help                           Display this usage message (Short form: /?)
/nologo                         Suppress compiler copyright message
/noconfig                       Do not auto include CSC.RSP file

                        - ADVANCED -
/baseaddress:<address>          Base address for the library to be built
/bugreport:<file>               Create a 'Bug Report' file
/codepage:<n>                   Specify the codepage to use when opening source
files
/utf8output                     Output compiler messages in UTF-8 encoding
/main:<type>                    Specify the type that contains the entry point
(ignore all other possible entry points) (Short form: /m)
/fullpaths                      Compiler generates fully qualified paths
/filealign:<n>                  Specify the alignment used for output file
sections
/pdb:<file>                     Specify debug information file name (default:
output file name with .pdb extension)
/errorendlocation               Output line and column of the end location of each
error
/preferreduilang                Specify the preferred output language name.
/nostdlib[+|-]                  Do not reference standard library (mscorlib.dll)
/subsystemversion:<string>      Specify subsystem version of this assembly
/lib:<file list>                Specify additional directories to search in for
references
/errorreport:<string>           Specify how to handle internal compiler errors:
prompt, send, queue, or none. The default is queue.
/appconfig:<file>               Specify an application configuration file
containing assembly binding settings
/moduleassemblyname:<string>    Name of the assembly which this module will be a
part of
```

7.2 Der aktuelle (neue) C#-Compiler

Der im Projekt "Roslyn" neu implementierte C#-Compiler heißt auch csc.exe; er ist aber nicht mehr Teil des .NET Framework Redistributable. Er wird auf diesen Wegen verbreitet:

- Visual Studio 2017 bzw. Visual Studio 2017 Build Tools
- .NET Core SDK
- Nuget-Paket https://www.nuget.org/packages/Microsoft.Net.Compilers

Visual Studio installiert den Compiler in C:\Program Files (x86)\Microsoft Visual Studio\2017\Enterprise\MSBuild\15.0\Bin\Roslyn. Das Nuget-Paket *https://www.nuget.org/packages/Microsoft.Net.Compilers* enthält den csc.exe im Ordner /Tools. Im .NET Core SDK wird der C#-Compiler nicht als csc.exe mitgeliefert, sondern über die .NET CLI-Werkzeuge angesprochen (z.B. dotnet build).

Die folgende Abbildung zeigt die Installation des C#-Compilers per Nuget.exe mit dem Befehl:

```
nuget install Microsoft.Net.Compilers
```

Das Programm Nuget.exe bekommt man *https://www.nuget.org/downloads*.

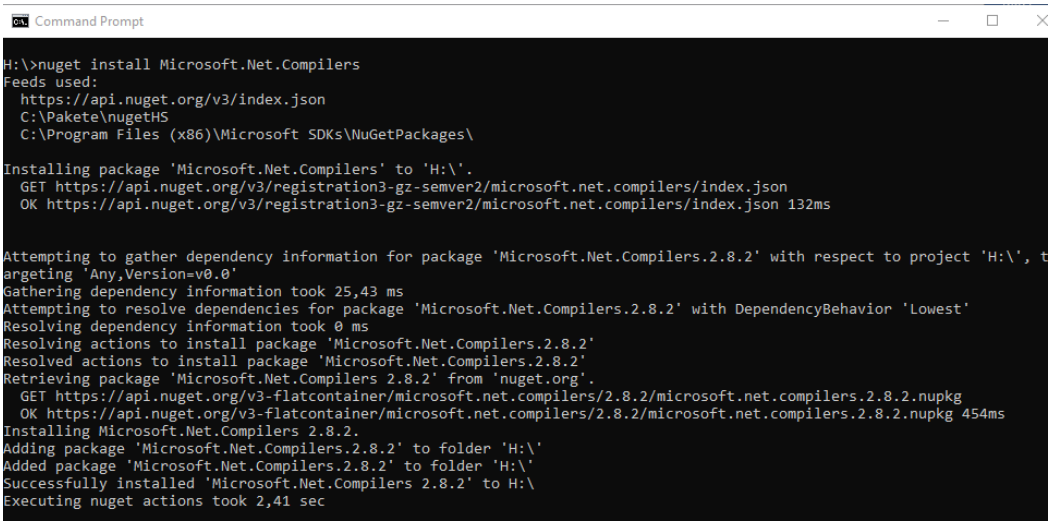


Abbildung: Installation des neuen C#-Compilers via Nuget

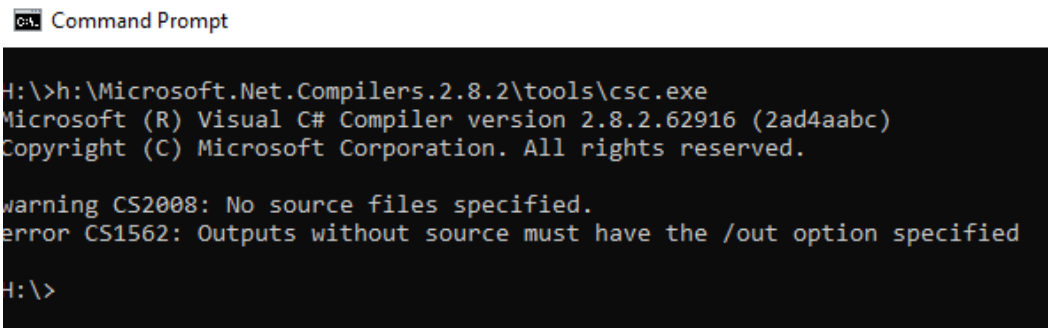


Abbildung: Start des neuen C#-Compiler aus der Nuget-Installation

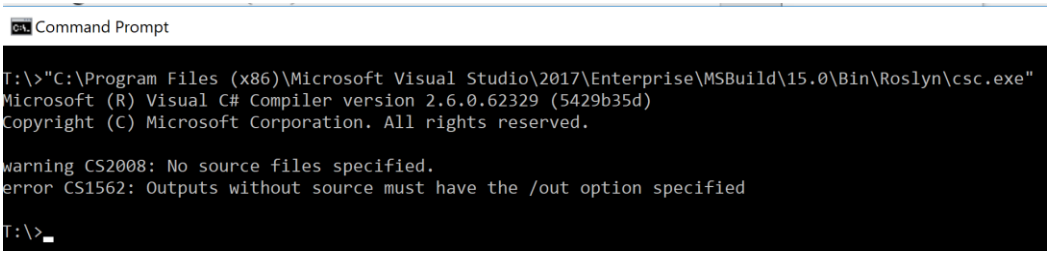


Abbildung: Start des neue C#-Compiler aus der Visual Studio-Installation

Die Neufassung des CodeDOM-APIs mit dem neuen Compiler erhält man über das Nuget-Paket *www.nuget.org/packages/Microsoft.CodeDom.Providers.DotNetCompilerPlatform*.

7.2.1 Versionsnummern des Compilers

Die Versionsummer des neuen C#-Compilers richtet sich nach dem Funktionsumfang des Compilers, nicht nach den Sprachfeatures (siehe folgende Abbildung).

- Versions `1.x` mean C# 6.0 and VB 14 (Visual Studio 2015 and updates). For instance, `1.3.2` corresponds to the most recent update (update 3) of Visual Studio 2015.
- Version `2.0` means C# 7.0 and VB 15 (Visual Studio 2017 version 15.0).
- Version `2.1` is still C# 7.0, but with a couple fixes (Visual Studio 2017 version 15.1).
- Version `2.2` is still C# 7.0, but with a couple more fixes (Visual Studio 2017 version 15.2). Language version "default" was updated to mean "7.0".
- Version `2.3` means C# 7.1 and VB 15.3 (Visual Studio 2017 version 15.3). For instance, `2.3.0-beta1` corresponds to Visual Studio 2017 version 15.3 (Preview 1).
- Version `2.4` is still C# 7.1 and VB 15.3, but with a couple fixes (Visual Studio 2017 version 15.4).
- Version `2.6` means C# 7.2 and VB 15.5 (Visual Studio 2017 version 15.5).
- Version `2.7` means C# 7.2 and VB 15.5, but with a number of fixes (Visual Studio 2017 version 15.6).
- Version `2.8` means C# 7.3 (Visual Studio 2017 version 15.7)
- Version `2.9` is still C# 7.3 and VB 15.5, but with more fixes (Visual Studio 2017 version 15.8)

Abbildung: Versionierung des neuen C#-Compilers [https://github.com/dotnet/roslyn/wiki/NuGet-packages]

7.2.2 Kommandozeilenparameter

Es folgen die Kommandozeilenparameter des neuen C#-Compilers

```
                    Visual C# Compiler Options

                        - OUTPUT FILES -
```

```
 /out:<file>                   Specify output file name (default: base name of
                               file with main class or first file)
 /target:exe                   Build a console executable (default) (Short
                               form: /t:exe)
 /target:winexe                Build a Windows executable (Short form:
                               /t:winexe)
 /target:library               Build a library (Short form: /t:library)
 /target:module                Build a module that can be added to another
                               assembly (Short form: /t:module)
 /target:appcontainerexe       Build an Appcontainer executable (Short form:
                               /t:appcontainerexe)
 /target:winmdobj              Build a Windows Runtime intermediate file that
                               is consumed by WinMDExp (Short form: /t:winmdobj)
 /doc:<file>                   XML Documentation file to generate
 /refout:<file>                Reference assembly output to generate
 /platform:<string>            Limit which platforms this code can run on: x86,
                               Itanium, x64, arm, anycpu32bitpreferred, or
                               anycpu. The default is anycpu.

                         - INPUT FILES -
 /recurse:<wildcard>           Include all files in the current directory and
                               subdirectories according to the wildcard
                               specifications
 /reference:<alias>=<file>     Reference metadata from the specified assembly
                               file using the given alias (Short form: /r)
 /reference:<file list>        Reference metadata from the specified assembly
                               files (Short form: /r)
 /addmodule:<file list>        Link the specified modules into this assembly
 /link:<file list>             Embed metadata from the specified interop
                               assembly files (Short form: /l)
 /analyzer:<file list>         Run the analyzers from this assembly
                               (Short form: /a)
 /additionalfile:<file list>   Additional files that don't directly affect code
                               generation but may be used by analyzers for
producing
                               errors or warnings.
 /embed                        Embed all source files in the PDB.
 /embed:<file list>            Embed specific files in the PDB

                         - RESOURCES -
 /win32res:<file>              Specify a Win32 resource file (.res)
 /win32icon:<file>             Use this icon for the output
 /win32manifest:<file>         Specify a Win32 manifest file (.xml)
 /nowin32manifest              Do not include the default Win32 manifest
 /resource:<resinfo>           Embed the specified resource (Short form: /res)
 /linkresource:<resinfo>       Link the specified resource to this assembly
                               (Short form: /linkres) Where the resinfo format
                               is <file>[,<string name>[,public|private]]

                         - CODE GENERATION -
 /debug[+|-]                   Emit debugging information
 /debug:{full|pdbonly|portable|embedded}
                               Specify debugging type ('full' is default,
```

```
                                'portable' is a cross-platform format,
                                'embedded' is a cross-platform format embedded
into
                                the target .dll or .exe)
 /optimize[+|-]                 Enable optimizations (Short form: /o)
 /deterministic                 Produce a deterministic assembly
                                (including module version GUID and timestamp)
 /refonly                       Produce a reference assembly in place of the main
output
 /instrument:TestCoverage       Produce an assembly instrumented to collect
                                coverage information
 /sourcelink:<file>             Source link info to embed into PDB.

                        - ERRORS AND WARNINGS -
 /warnaserror[+|-]              Report all warnings as errors
 /warnaserror[+|-]:<warn list> Report specific warnings as errors
 /warn:<n>                      Set warning level (0-4) (Short form: /w)
 /nowarn:<warn list>            Disable specific warning messages
 /ruleset:<file>                Specify a ruleset file that disables specific
                                diagnostics.
 /errorlog:<file>               Specify a file to log all compiler and analyzer
                                diagnostics.
 /reportanalyzer                Report additional analyzer information, such as
                                execution time.

                        - LANGUAGE -
 /checked[+|-]                  Generate overflow checks
 /unsafe[+|-]                   Allow 'unsafe' code
 /define:<symbol list>          Define conditional compilation symbol(s) (Short
                                form: /d)
 /langversion:?                 Display the allowed values for language version
 /langversion:<string>          Specify language version such as
                                `default` (latest major version), or
                                `latest` (latest version, including minor
versions),
                                or specific versions like `6` or `7.1`

                        - SECURITY -
 /delaysign[+|-]                Delay-sign the assembly using only the public
                                portion of the strong name key
 /publicsign[+|-]               Public-sign the assembly using only the public
                                portion of the strong name key
 /keyfile:<file>                Specify a strong name key file
 /keycontainer:<string>         Specify a strong name key container
 /highentropyva[+|-]            Enable high-entropy ASLR

                        - MISCELLANEOUS -
 @<file>                        Read response file for more options
 /help                          Display this usage message (Short form: /?)
 /nologo                        Suppress compiler copyright message
 /noconfig                      Do not auto include CSC.RSP file
 /parallel[+|-]                 Concurrent build.
 /version                       Display the compiler version number and exit.
```

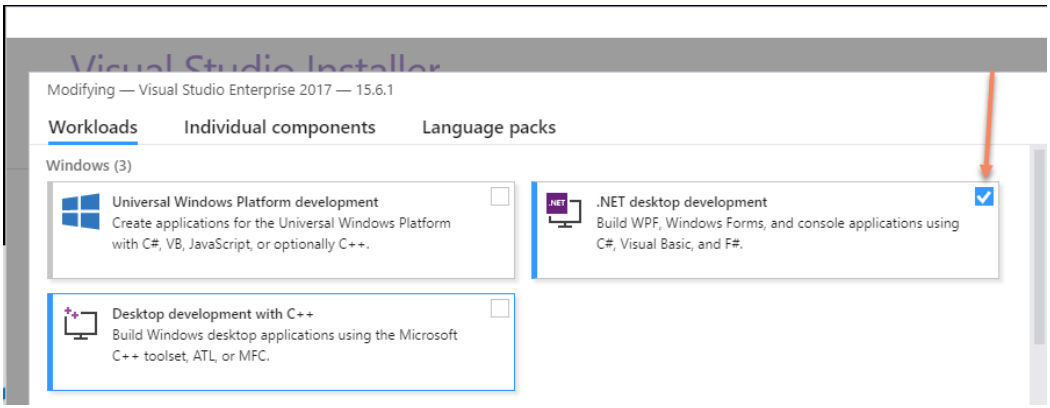
```
                        - ADVANCED -
/baseaddress:<address>        Base address for the library to be built
/checksumalgorithm:<alg>      Specify algorithm for calculating source file
                              checksum stored in PDB. Supported values are:
                              SHA1 (default) or SHA256.
/codepage:<n>                 Specify the codepage to use when opening source
                              files
/utf8output                   Output compiler messages in UTF-8 encoding
/main:<type>                  Specify the type that contains the entry point
                              (ignore all other possible entry points) (Short
                              form: /m)
/fullpaths                    Compiler generates fully qualified paths
/filealign:<n>                Specify the alignment used for output file
                              sections
/pathmap:<K1>=<V1>,<K2>=<V2>,...
                              Specify a mapping for source path names output by
                              the compiler.
/pdb:<file>                   Specify debug information file name (default:
                              output file name with .pdb extension)
/errorendlocation             Output line and column of the end location of
                              each error
/preferreduilang              Specify the preferred output language name.
/nostdlib[+|-]                Do not reference standard library (mscorlib.dll)
/subsystemversion:<string>    Specify subsystem version of this assembly
/lib:<file list>              Specify additional directories to search in for
                              references
/errorreport:<string>         Specify how to handle internal compiler errors:
                              prompt, send, queue, or none. The default is
                              queue.
/appconfig:<file>             Specify an application configuration file
                              containing assembly binding settings
/moduleassemblyname:<string>  Name of the assembly which this module will be
                              a part of
/modulename:<string>          Specify the name of the source module
```

8 Erste Schritte Visual Studio

Dieses Büchlein ist kein Handbuch für Visual Studio. Für Leser, die neu in Visual Studio sind, folgt jedoch hier eine kurze Einführung in das Anlegen und übersetzen eines Projekts am Beispiel von Konsolenanwendungsprojekten für .NET Framework und .NET Core. Zum Einsatz kommt Visual Studio 2017.

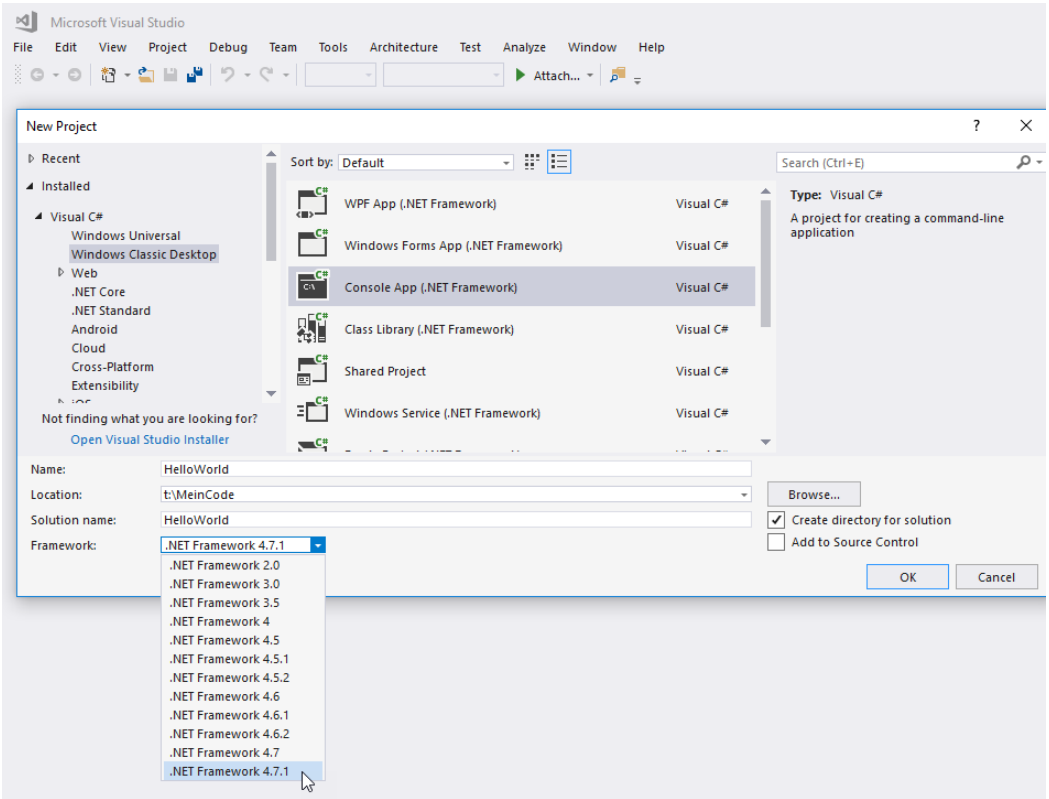
8.1 Hello World mit dem .NET Framework

Wählen Sie bei der Installation von Visual Studio den Workload ".NET Desktop Development" aus.

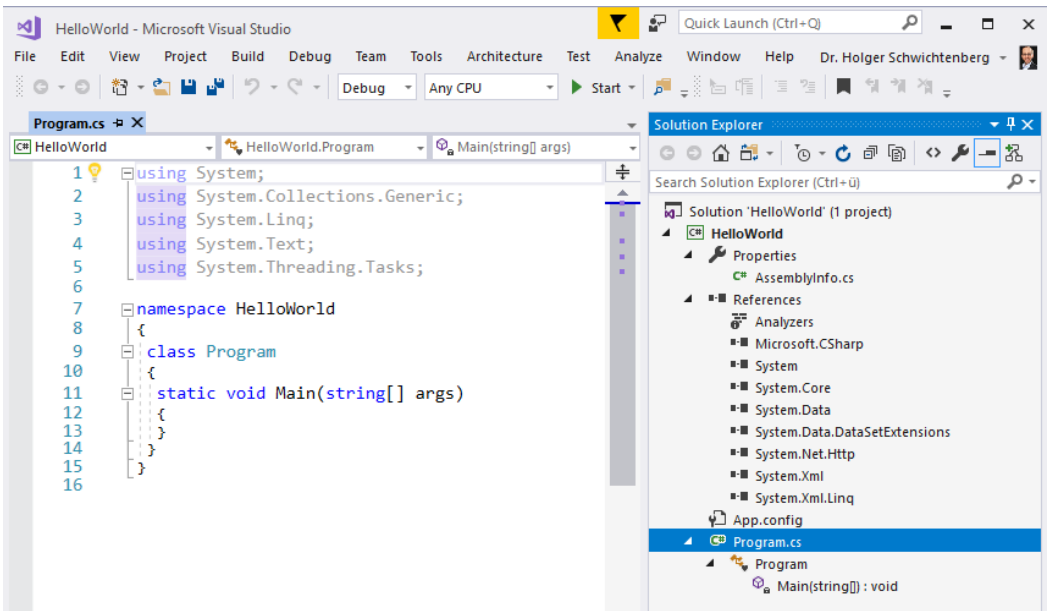


Starten Sie Visual Studio

Wählen Sie File/New Project und dann im dem Dialog "Visual C#/Windows Classic Desktop/Console App". Geben Sie unten den gewünschten Standort ein (wählen Sie am besten einen Verzeichnisnamen ohne Leerzeichen!) und wählen Sie die aktuellste .NET Framework-Version aus.



Sie erhalten dann eine Projektmappe (.sln-Datei im Dateisystem) mit einem Projekt (.csproj-Datei). In dem Projekt gibt es eine Datei program.cs mit der Grundstruktur der Konsolenanwendung.

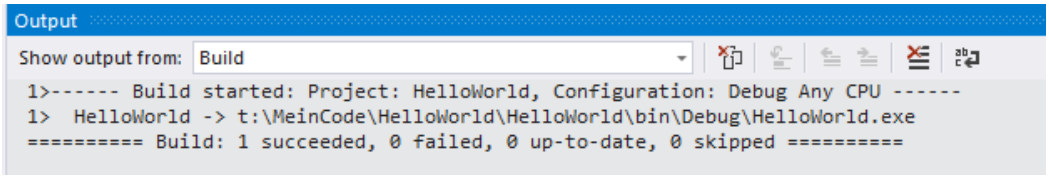


Ergänzen Sie in Main() den folgenden Programmcode:

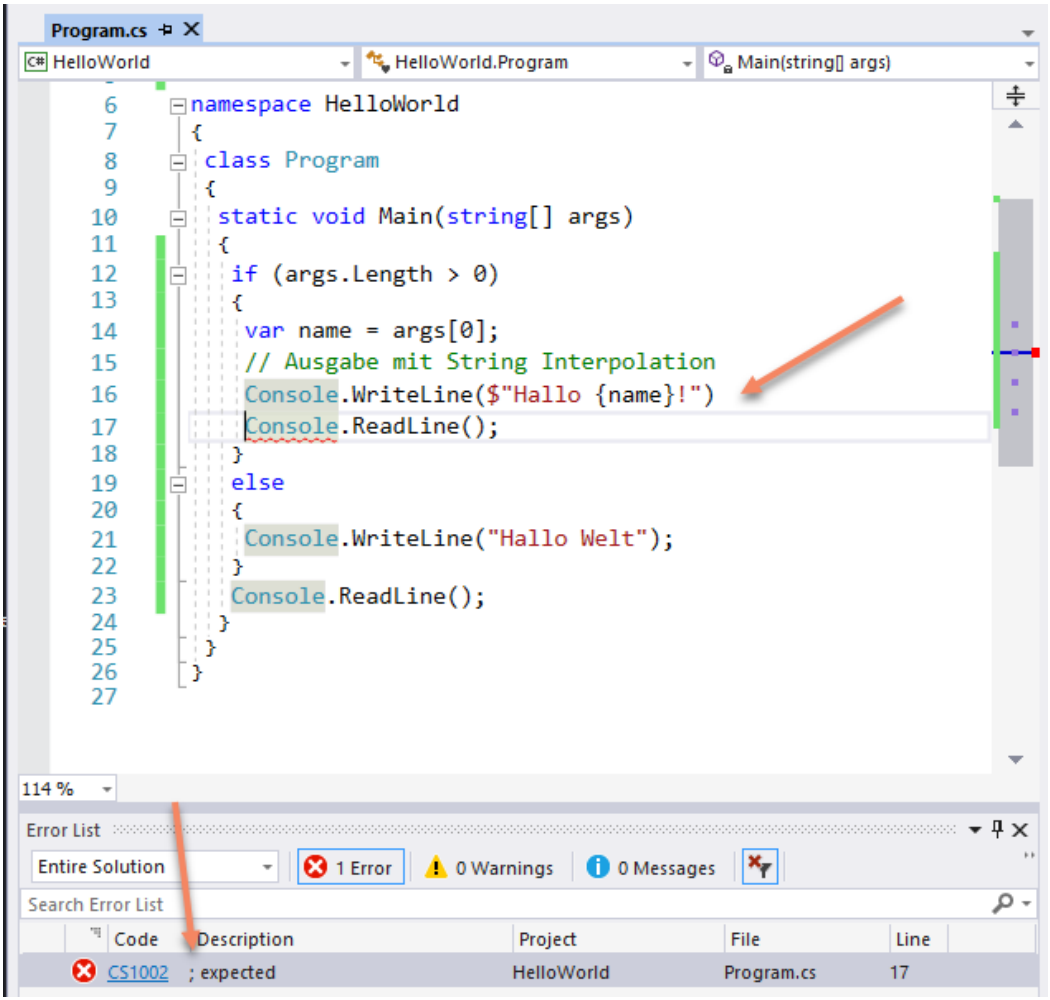
```
namespace HalloWelt
{
 class Program
 {
  static void Main(string[] args)
  {
   if (args.Length > 0)
   {
    var name = args[0];
    // Ausgabe mit String Interpolation
    Console.WriteLine($"Hallo {name}!");
    Console.ReadLine();
   }
   else
   {
    Console.WriteLine("Hallo Welt!");
   }
   Console.ReadLine();
  }
 }
}
```

Wählen Sie Build/Build Solution (Alternativ die Tastenkombination STRG+SHIFT+B), um den Programmcode zu übersetzen.

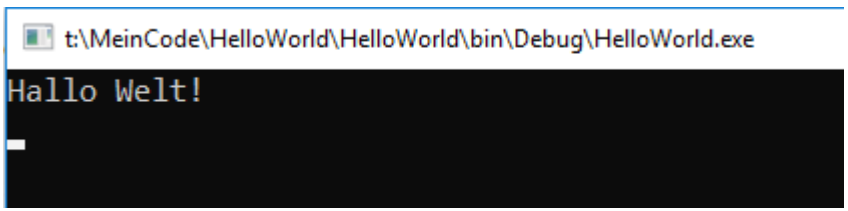
Sie sollten nun im Ausgabefenster (Einblenden über View/Output) dies sehen:



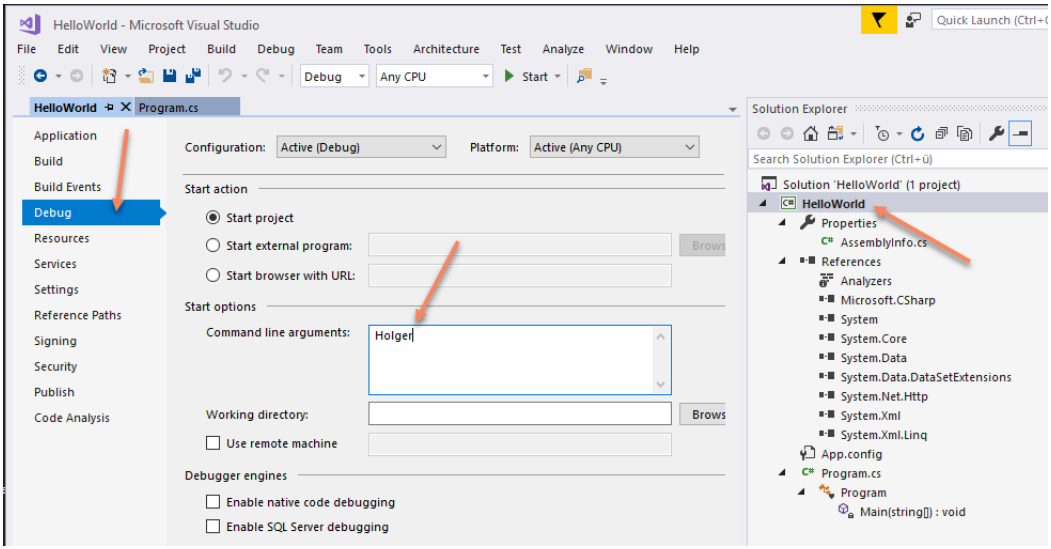
Falls Sie Eingabefehler gemacht haben, sehen Sie dies im Fenster "Error List".



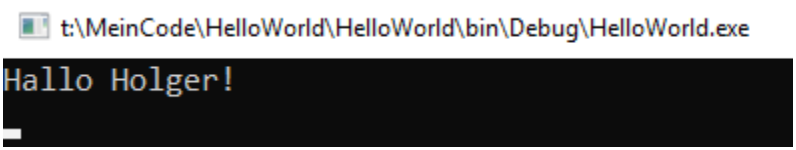
Wenn Ihr Programm erfolgreich übersetzt, starten Sie es im Debugger mit Debug/Start Debugging oder der Taste F5.



Um dem Programm beim Start einen Kommandozeilenparameter zu übergeben, wählen Sie im Solution Explorer im Kontextmenü des Projekts (nicht der Projektmappe, so "Soultion" davor steht) den Eintrag "Properties" und tragen Sie in der Registerkarte "Debug" bei "Command Line Arguments" Ihren Namen ein.

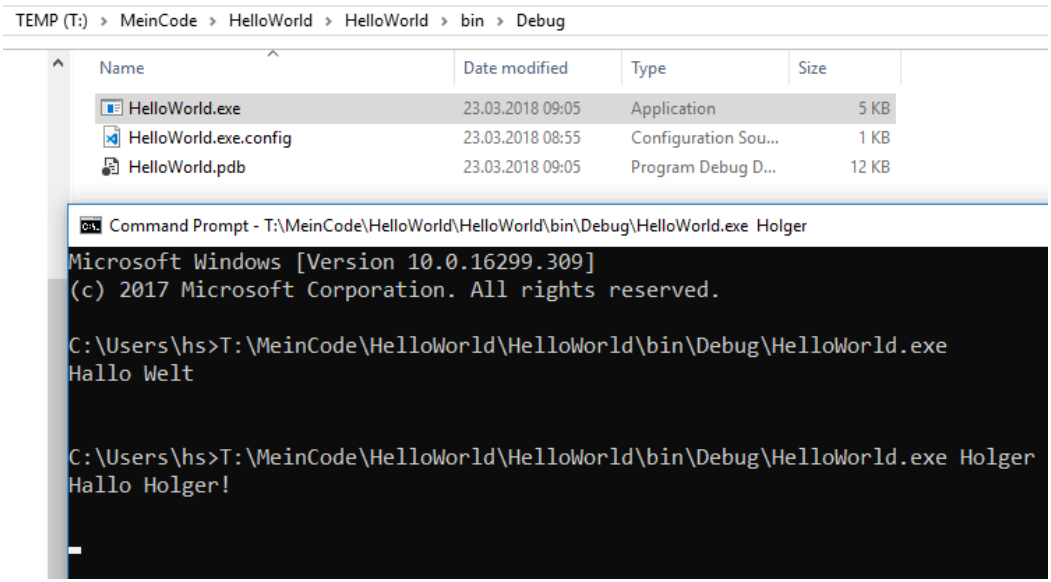


Drücken Sie wieder F5.



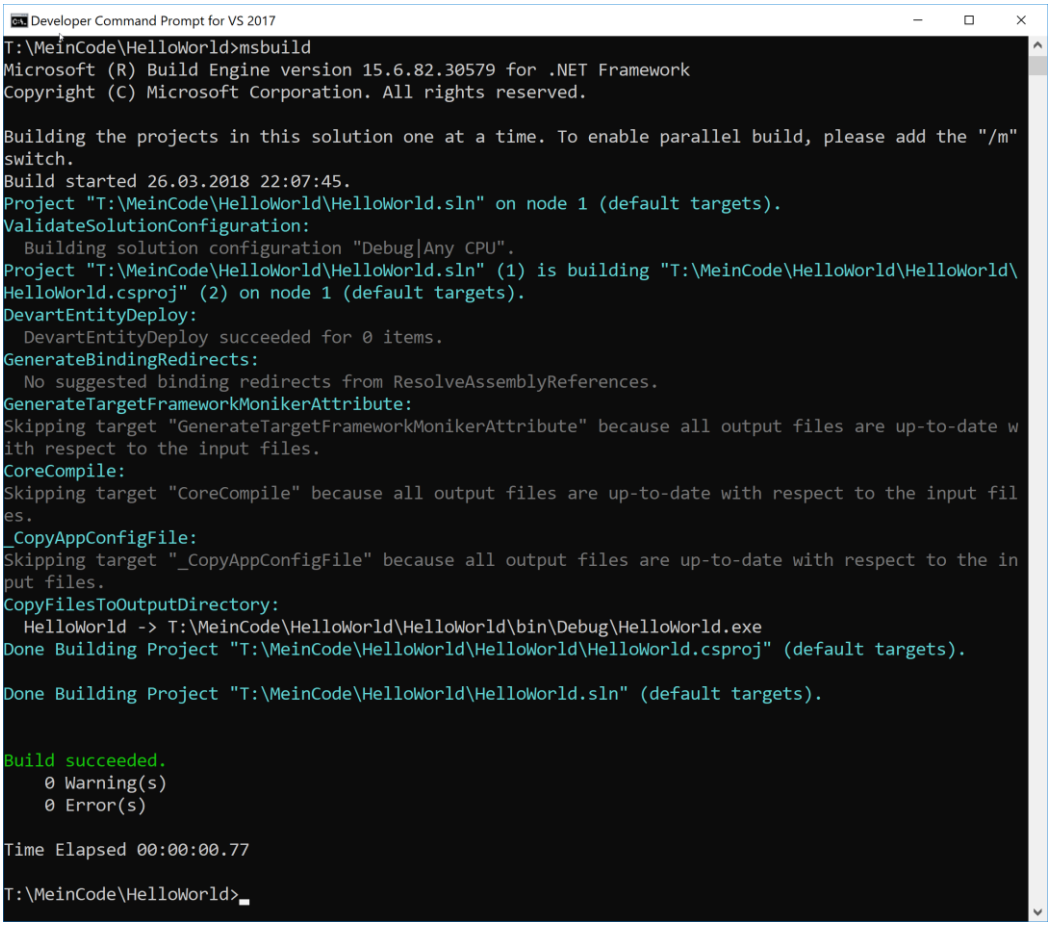
Schauen Sie sich das Projekt auf der Festplatte im Windows Explorer an. Sie erkennen ein Ausgabeverzeichnis bin/Debug in dem das kompilierte Programm als .EXE-Datei liegt, die man direkt starten kann.

Das Kompilat in .NET nennt man eine Assembly. Die Assembly ist in diesem Fall eine .EXE-Datei.



Sie können ein in Visual Studio erzeugtes .NET-Projekt auch an der Kommandozeile übersetzen. Theoretisch kann man dazu den C#-Compiler csc.exe direkt einsetzen, aber dann muss man alle Quellcodedateien sowie benötigte Referenzen auf andere Assemblies dort als Parameter angeben. Da diese Abhängigkeiten alle bereits in den Projektdateien definiert sind, bietet sich der Einsatz von msbuild.exe an, dass die .csproj-Dateien auswertet. Öffnen Sie dazu den "Developer Command Prompt", der mit Visual Studio installiert wird, gehen Sie in das Verzeichnis mit der .sln-Datei und rufen Sie msbuild.exe auf.

Hinweis: Andere .NET-Anwendungsarten (z.B. Webanwendungen mit ASP.NET, Desktop-Anwendungen mit Windows Forms oder Windows Presentation Foundation, Mobile Apps mit Xamarin) erstellen und übersetzen Sie mit den gleichen Funktionen und Werkzeugen. Sie müssen nur entsprechende Workloads im Setup von Visual Studio installieren und dann die entsprechende Projektvorlage wählen.



8.2 Hello World mit .NET Core

Hier werden die Schritte beschrieben, die anders sind, wenn Sie .NET Core verwenden wollen statt .NET Framework.

Wichtig ist, dass Sie in Visual Studio 2017 nicht nur den Workload ".NET Core Cross-Platform Development" wählen, sondern das .NET Core SDK in der aktuellen Version zusätzlich von [https://www.microsoft.com/net/download/windows] installieren.

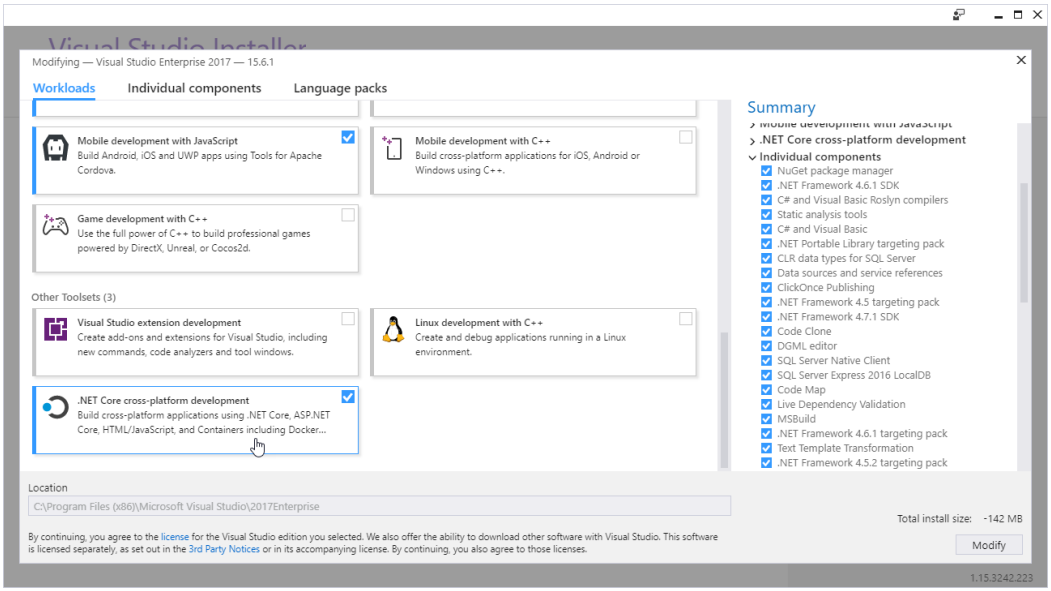
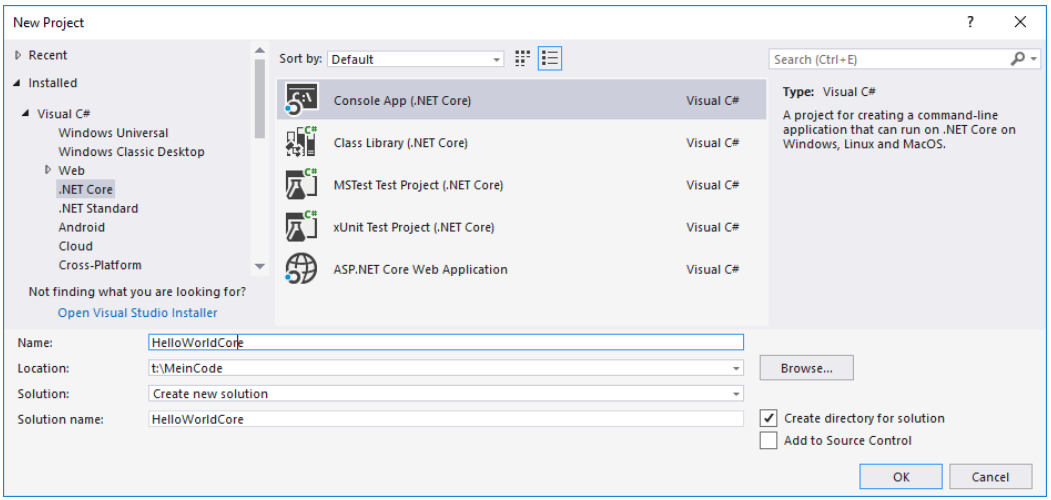
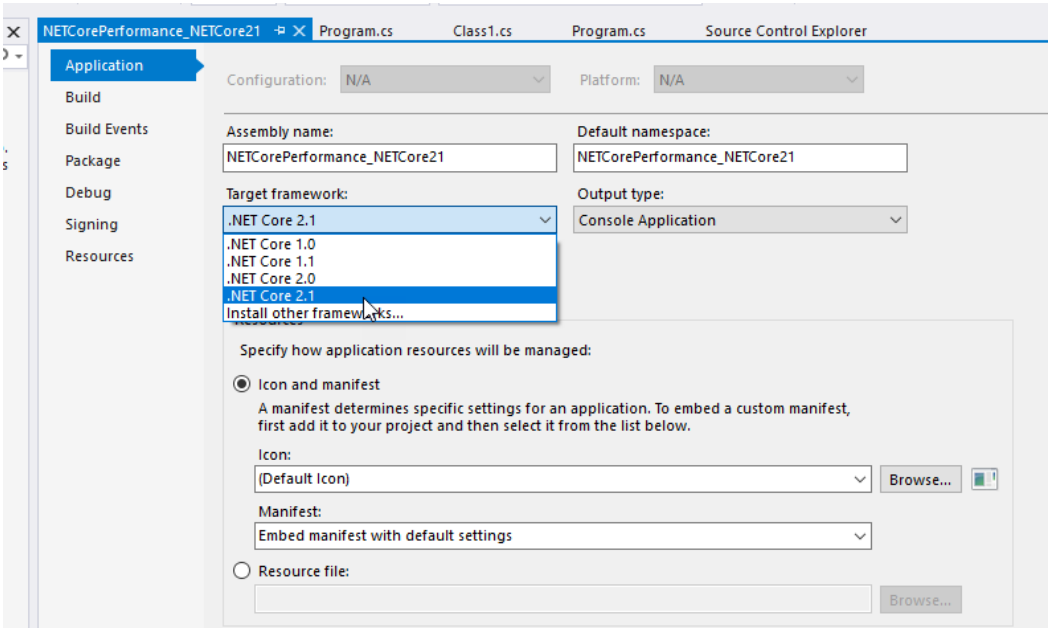


Abbildung: Installation des Workloads ".NET Core Cross-Platform Development" in Visual Studio 2017

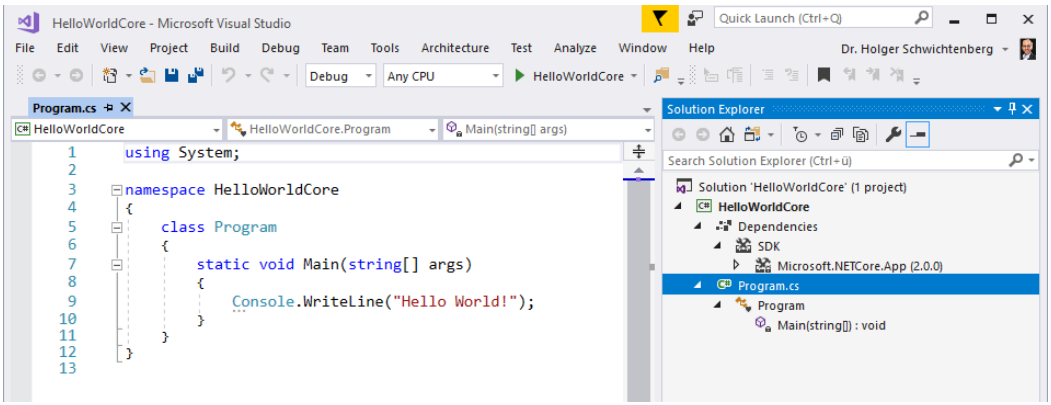
Wählen Sie bei File/New Project jetzt "Visual C#/.NET Core/Console App (.NET Core).



Man kann in dieser Maske nicht die .NET Core-Version festlegen. Dies geht erst nach dem Anlegen in den Projekteigenschaften "Application/Target Framework".

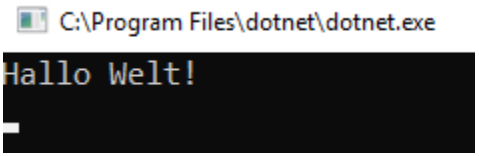


Der Projektaufbau eines .NET Core-Projekts ist etwas anders als bei einem klassischen .NET-Projekt (z.B. Ast "Dependencies" statt "References"), die Bedienung bezüglich übersetzen und Debugging aber gleich.



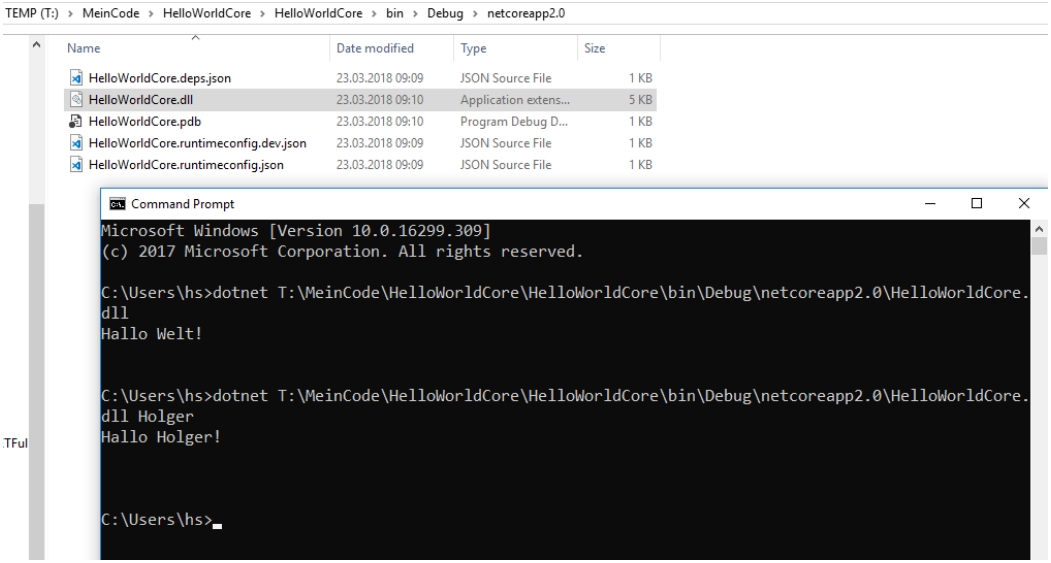
Der gleiche Programmcode kann hier eingetragen werden.

Bei Start der Anwendung sieht man in der Titelzeile dotnet.exe, was das universelle Kommandozeilenwerkzeug von .NET Core ist, dass auch zum Start einer .NET Core-Anwendung verwendet wird.

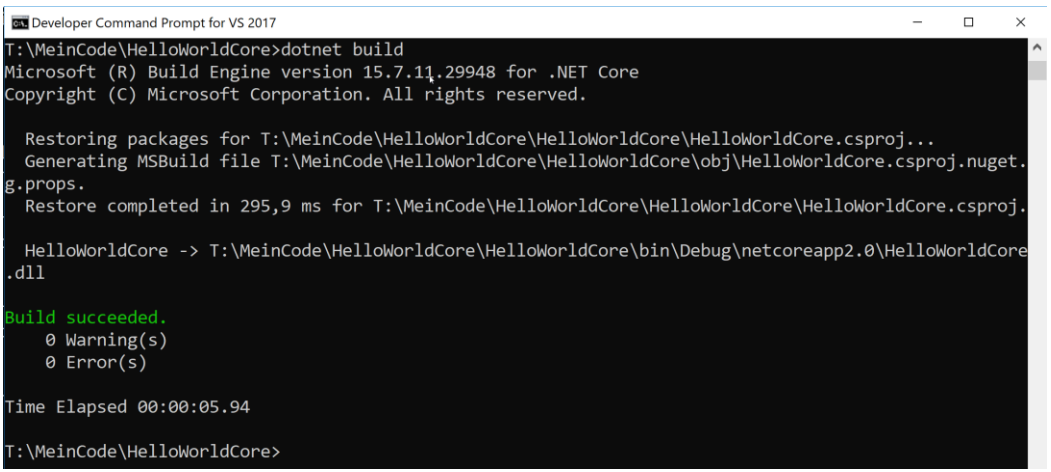


Während man beim .NET Framework im Ausgabeverzeichnis eine .EXE-Datei erhält, bekommt man bei .NET Core nur eine .DLL. Daher muss man dotnet.exe (oder abgekürzt dotnet) beim Start voranstellen.

Das Kompilat nennt man auch in .NET Core eine Assembly. Die Assembly ist in diesem Fall eine .DLL-Datei.



Ein .NET Core-Projekt können Sie an der Kommandozeile mit msbuild.exe oder dotnet.exe build übersetzen.



Hinweis: Andere .NET Core-Anwendungsarten (z.B. Webanwendungen mit ASP.NET Core, Universal Windows Platform Apps) erstellen und übersetzen Sie mit den gleichen Funktionen und Werkzeugen. Sie müssen nur entsprechende Workloads im Setup von Visual Studio installieren und dann die entsprechende Projektvorlage wählen.

8.3 Festlegung der Compilerversion in Visual Studio

Während früher die verwendete Viusal Studio-Version auch die verwendete Version des Sprachcompilers von C# festlegte, kann man seit Visual Studio 2017 Update 3 (Version 15.3) die Sprachversion pro Projekt in den Projekteigenschaften (Build/Avanced) festlegen.

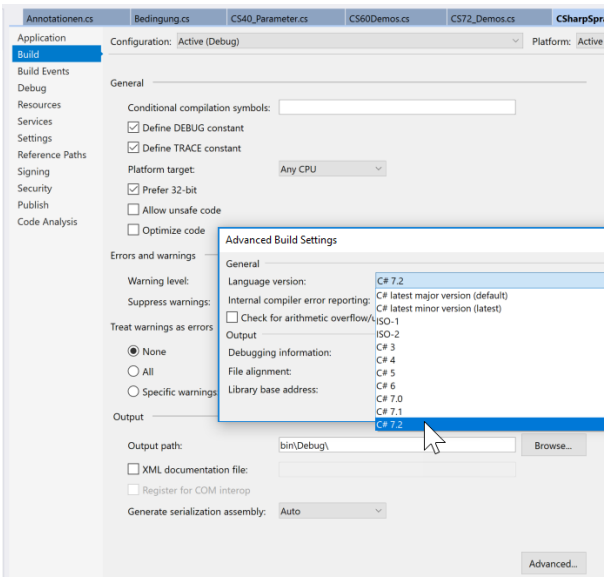


Abbildung: Einstellen der Sprachversion

Zudem warnt Visual Studio, wenn Sie ein Sprachfeature verwenden, welches es in der eingestellten Version noch nicht gibt.