Spickzettel („Cheat Sheet“) C# 11.0 und .NET 7.0 – neue Sprachfeatures und neue APIs

Autor: Dr. Holger Schwichtenberg (www.IT-Visions.de)

www.IT-Visions.de
Dr. Holger Schwichtenberg

Neue .NET SDK CLI-Befehle in .NET 7.0

Online-Suche auf NuGet.org nach Projektvorlagen

Bisher: *dotnet new blazor --search --language C# --author xy*

Nun auch: *dotnet new search blazor --language C# --author xy*

Installieren einer Projektvorlage von NuGet.org

Bisher: *dotnet new --install PACKAGE_ID*

Nun auch: *dotnet new install PACKAGE_ID*

Liste aller installierten Projektvorlagen

Bisher: *dotnet new --list*

Nun auch: *dotnet new list*

Deployment direkt in Container ohne Dockerfile

dotnet add package Microsoft.NET.Build.Containers

dotnet publish --os linux --arch x64 -c Release
-p:PublishProfile=DefaultContainer

Native Ahead-of-Time-Compilation (AOT)

In .csproj-Datei, leider nur für Konsolenanwendungen und DLLs!

```
<PropertyGroup>
    <PublishAot>true</PublishAot>
</PropertyGroup>
```

Kompilierung:

```
dotnet publish -r win-x64 -c Release
```

C# 11: Neue .NET-Basisdatentypen für Zahlen

System.Int128	Int128	Ganzzahl	16 Bytes
System.UInt128	UInt128	Ganzzahl	16 Bytes

C# 11: Neue Typaliase für bestehende Basisdatentypen

System.IntPtr	nint	Zeiger	4 oder 8 Bytes
System.UIntPtr	nuint	Zeiger	4 oder 8 Bytes

C# 11: nameof() in Annotationen für Parameternamen

```
[Description($"Diese Methode besitzt einen generischen Typparameter mit Namen {nameof(T)} und erwartet
eine Instanz dieses Types im Parameter {nameof(obj)}.")]
static void NameOfErweiterungen<T>(T obj)  {  …  }
```

C# 11: UTF8-Zeichenketten mit Zusatz u8 oder U8

```
ReadOnlySpan<byte> s1 = "Hallo Holger!"u8;
var s2 = "Hallo Holger!"u8;
byte[] s3 = "Hallo Holger!"u8.ToArray();
```

liefert jeweils Bytefolge *0x48 0x61 … 0x6C 0x67 0x65 0x72 0x21*

UTF8-Zeichenketten können nicht verwendet werden mit String Interpolation und in Standardwerten für Parameter!

C# 11: Zeilenumbrüche in Interpolationsausdrücken

```
string ganzerName = "Dr. Holger Schwichtenberg";
var t = $"Vorname: { // Kommentare nun möglich
                   ganzerName // Vor- und Nachname
                   .Split(" ") // aufteilen
                   .ElementAt(1)}";
```

C# 11: file = Typen, die nur in einer Datei sichtbar sind

```
file class Person { … }
```

Interner Name wird automatisch vom Compiler vergeben!

C# 11: Raw Literal Strings

- Drei oder mehr Anführungszeichen am Anfang und Ende
- Interpolation mit zwei oder mehr $$ und Klammerpaaren {{ x }}
- Keine Steuerzeichen definiert
- Umbrüche landen in der Zeichenkette
- Einrückungen bleiben erhalten, aber alles vor Ende (""") entfällt

```
var name = "Dr. Holger Schwichtenberg";
var website = "www.dotnet-doktor.de";
var json = $$"""
  {
   "Person": {
   "Name": "{{name}}",
   "Webseite": "{{website}}"
   }
  }
  """;
```

```
{
 "Person": {
  "Name": "Dr. Holger Schwichtenberg",
  "Webseite": "www.dotnet-doktor.de"
  }
}
```

C# 11: Auto-default Structs

```
struct Trainer {
 public int ID;
 public string? Name { get; set; }
 public Trainer() { }
 public override string ToString() {
  return $"#{this.ID} Name: {this.Name ?? "(NULL)"}"; }
}
```

Vor C# 11 beschwerte sich hier der Compiler:

```
CS0843  Auto-implemented property 'Trainer.Name' must be fully assigned before control is returned to the caller.
CS0171  Field 'Trainer.ID' must be fully assigned before control is returned to the caller
```

Ab C# 11 werden alle nicht explizit initialisierten Felder und Properties mit Standardwerten initialisiert, z.B. "*#0 Name: (NULL)*"

C# 11: Required Members

- Required ist erlaubt in Klassen, Strukturen und Record-Typen, aber nicht in Schnittstellen
- Required ist erlaubt bei Fields und Properties
- Konstruktor mit [SetsRequiredMembers] annotierbar. Compiler prüft aber nicht, ob wirklich alle Pflichtmitglieder dort gesetzt!

```
public class Consultant {
 public Consultant() { }
 [SetsRequiredMembers]
 public Consultant(int id, string name)
              => (ID, Name) = (id, name);
 public required int ID; // Field
 public required string Name { get; init; } // Property
 public string? City { get; set; } // nicht "required"!
}
```

Erlaubte Nutzung via Konstruktor oder Objekt-Initialisierer:

```
var p1 = new Consultant(1, "Dr. Holger Schwichtenberg");
var p2 = new Consultant() {ID = 2, Name = "Marc Müller"};
```

NICHT erlaubte Instanzierung, weil ID und Name nicht gesetzt:

```
var p3 = new Consultant();
```

C# 11: Static Abstract in Schnittstellen

```
interface IAbc {
 abstract string GetA();
 static string GetB() => "B";
 static abstract int ID { get; set; } // NEU
 static abstract string GetC(); // NEU
}
```

```
class Abc : IAbc {
 public static int ID { get; set; } = 1;
 public string GetA() => "A";
 public static string GetC() => "C";
}
```

Nutzung:

```
var obj = new Abc();
Console.WriteLine(obj.GetA());
Console.WriteLine(IAbc.GetB());
Console.WriteLine(Abc.ID);
Console.WriteLine(Abc.GetC());
```

C# 11: Generic Attributes / Generic Annotations

Deklaration:

```
public class GenericAttribute<T> : System.Attribute { }
```

Als Typparameter nicht erlaubt: dynamic, Nullable Reference Types, Tupel in C#-Syntax (ValueTupel<T,T> ist aber erlaubt!)

Nutzung:

```
[GenericAttribute<Person>()]
class PersonManager {
 public Person p { get; set; }
 [GenericAttribute<Person>()]
 public Person CreatePerson(int ID, string Name) {
  return new Person(ID, Name); }
}
```

C# 11: List Pattern

```
public string CheckList(int[] values)
=> values switch {
 [1, 2, .., 10] => "beginnt mit 1, 2 sowie endet mit 10",
 [1, 2] => "besteht aus 1 und 2",
 [1, _] => "beginnt mit 1, danach noch ein Element",
 [1, ..] => "beginnt mit 1, danach mehrere Elemente",
 [..] => "mehrere Elemente, beginnt nicht mit 1"
};
```

Einsatzbeispiele von CheckList():

```
Console.WriteLine(CheckList(new[] { 1, 2, 10 }));
```

→ "beginnt mit 1, 2 sowie endet mit 10"

```
Console.WriteLine(CheckList(new[] { 1, 3 }));
```

→ "beginnt mit 1, danach noch ein Element"

```
Console.WriteLine(CheckList(new[] { 1, 2, 5 }));
```

→ "beginnt mit 1, noch mehrere Elemente"

```
Console.WriteLine(CheckList(new[] { 3, 5, 6, 7 }));
```

→ "mehrere Elementen, beginnt nicht mit 1"

Spickzettel („Cheat Sheet“) C# 11.0 und .NET 7.0 – neue Sprachfeatures und neue APIs

Autor: Dr. Holger Schwichtenberg (www.IT-Visions.de)

www.IT-Visions.de
Dr. Holger Schwichtenberg

C# 11: Slice Pattern

```
public string ExtractValue(int[] values)
=> values switch {
 [1, var middle, _] => $"Mittlere Zahl von 3 Zahlen (Beginn 1): {String.Join(", ", middle)}",
 [_, var middle, _] => $"Mittlere Zahl von 3 Zahlen (Beginn beliebig): {String.Join(", ", middle)}",
 [.. var all] => $"Alle Zahlen: {String.Join(", ", all)}"
};
public string ExtractValues(int[] values)
=> values switch {
 [1, .. var mittlereZahlen, _] => $"Mittlere Zahlen (Beginn 1): {String.Join(", ", mittlereZahlen)}",
 [.. var alleZahlen] => $"Alle Zahlen: {String.Join(", ", alleZahlen)}"
};
```

Einsatzbeispiele von ExtractValue():

```
Console.WriteLine(ExtractValue(new[] { 1, 2, 6 }));
```

→ "Mittlere Zahl von 3 Zahlen (Beginn 1): 2"

```
Console.WriteLine(ExtractValue(new[] { 2, 5, 6, 7 }));
```

→ "Alle Zahlen: 2, 5, 6, 7"

Einsatzbeispiele von ExtractValues():

```
Console.WriteLine(ExtractValues(new[] { 1, 2, 5, 6 }));
```

→ "Mittlere Zahlen (Beginn 1): 2, 5"

```
Console.WriteLine(ExtractValues(new[] { 2, 5, 6, 7 }));
```

→ "Alle Zahlen: 2, 5, 6, 7"

.NET 7.0: Generische Mathematik mit INumber<T> u.a.

```
T Calc<T>(T x, T y) where T : INumber<T> {
 Trace.WriteLine($"Calc {x.GetType().ToString()}/{y.GetType().ToString()}");
 if (x == T.Zero || y <= T.Zero) return T.One;
 return (x + y) * T.CreateChecked(42.24);
}
```

Diese generische Funktion kann mit beliebigen numerischen Typen aufgerufen werden:

```
Console.WriteLine(Calc(1, 2)); // Byte->126
Console.WriteLine(Calc((Int128)1, (Int128)2)); // ->126
Console.WriteLine(Calc(1.0d, 2.0d)); // Double->126,72
Console.WriteLine(Calc(1.0m, 2.0m)); // Decimal->126,720
```

.NET 7.0 LINQ: Order() und OrderDescending()

```
var data = new List<int>() { 2, 1, 3 };
var sortedAlt = data.OrderBy(e => e); //alt
var sortedNeu = data.Order();  //neu
var sortedDescAlt = data.OrderByDescending(e => e); //alt
var sortedDescNeu = data.OrderDescending(); //neu
```

.NET 7.0: Microseconds und Nanoseconds

Neue Eigenschaften *Microseconds* und *Nanoseconds* bei den Klassen *TimeStamp*, *DateTime*, *DateTimeOffset* und *TimeOnly*

```
TimeSpan ts = endeZeitunkt - startZeitunkt;
Console.WriteLine(ts.TotalMilliseconds + " ms");
Console.WriteLine(ts.TotalMicroseconds + " µs");
Console.WriteLine(ts.Ticks + " ticks"); //1 Tick = 100ns
Console.WriteLine(ts.TotalNanoseconds + " ns");
```

.NET 7.0: Regex Source Generator

Partielle Methode mit partieller Klasse; wird von SCG implementiert

```
public partial class Checker {
  // Quelle für regulären Ausdruck für ~99,99% Abdeckung von RFC 5322: http://emailregex.com/
  [RegexGenerator(@"\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*", RegexOptions.IgnoreCase)]
  public static partial Regex EMailRegEx();
  [RegexGenerator(@"(?im)^[{(]?[0-9A-F]{8}[-]?(?:[0-9A-F]{4}[-]?){3}[0-9A-F]{12}[)}]?$", RegexOptions.IgnoreCase)]
  public static partial Regex GUIDRegEx();
  public bool IsEMail(string input)  {
   return EMailRegEx().IsMatch(input);  }
  public string ExtractEMail(string input) {
   return EMailRegEx().Match(input).Value;  }
}
```

.NET 7.0: Platform Invoke Source Code Generator

```
public static partial class Win32API {
 [LibraryImport("User32.dll")]
 [return: MarshalAs(UnmanagedType.Bool)]
 public static partial bool MessageBeep(UInt32 beepType);
}
```

Aufruf dann: Win32API.MessageBeep(10);

.NET 7.0: Tarball-Archive (TAR)

TAR-Archiv mit GZip-Komprimierung erstellen aus Verzeichnis

```
string folder = @"T:\Dokumente";
string tar = @"T:\ArchivKomprimiert.tar.gz";
using (MemoryStream ms = new()) {
  using (TarWriter writer = new(ms, TarEntryFormat.Pax, leaveOpen: true))  {
    foreach (var file in new System.IO.DirectoryInfo(folder).GetFiles())  {
     writer.WriteEntry(fileName: file.FullName, entryName: file.Name);   }  }
  using FileStream tarstream = File.Create(tar);
  using GZipStream compressor = new(tarstream, CompressionMode.Compress);
  ms.Seek(0, SeekOrigin.Begin);
  ms.CopyTo(compressor);
  ms.Close();
}
```

Komprimiertes TAR extrahieren in ein Verzeichnis

```
using (FileStream fs = File.OpenRead(tar)) {
  using GZipStream decompressor = new(fs, CompressionMode.Decompress);
  using TarReader reader = new(decompressor, leaveOpen: false);
  TarEntry? entry;
  while ((entry = reader.GetNextEntry(copyData: true)) != null)  {
   string destFileName = Path.Join(folder, entry.Name);
   entry.ExtractToFile(destinationFileName, overwrite: true);   }
}
```

.NET 7.0: JSON-Polymorphismus

```
[JsonDerivedType(typeof(Person), typeDiscriminator: "P")]
[JsonDerivedType(typeof(Consultant), typeDiscriminator: "C")]
public class Person {
   public string Name { get; set; } }
public class Consultant : Person {
   public string Company { get; set; } }
```

Serialisierung:

```
Person value = new Consultant() { Name= "Holger Schwichtenberg", Company="www.IT-Visions.de" };
var json = JsonSerializer.Serialize<Person>(value);
Console.WriteLine(json); // → {"$type":"C","Company":"www.IT-Visions.de","Name":"Holger Schwichtenberg"}
```

Deserialisierung:

```
var ohneTyp = @"{""Company"":""www.IT-Visions.de"",""Name"":""Holger Schwichtenberg""}";
var mitTyp = @"{""$type"":""C"",""Company"":""www.IT-Visions.de"",""Name"":""Holger Schwichtenberg""}";
Person? p1 = JsonSerializer.Deserialize<Person>(ohneTyp);
Console.WriteLine(p1); // → Person
Person? p2 = JsonSerializer.Deserialize<Person>(mitTyp);
Console.WriteLine(p2); // → Consultant
```

Webadressen

Website des Autors zu .NET 7.0
www.dotnet7.de
Neuerungen in .NET 7.0
learn.microsoft.com/en-us/dotnet/core/whats-new/dotnet-7
Neuerungen in C# 11.0
learn.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-11
Neuerungen in ASP.NET Core 7.0
learn.microsoft.com/en-us/aspnet/core/release-notes/aspnetcore-7.0
Neuerungen in Entity Framework Core 7.0
learn.microsoft.com/ef/core/what-is-new/ef-core-7.0/whatsnew
Breaking Changes in .NET 7.0
learn.microsoft.com/en-us/dotnet/core/compatibility/7.0

Über den Autor

Dr. Holger Schwichtenberg gehört zu den bekanntesten Experten für Softwareentwicklung in Deutschland. Er hat zahlreiche Bücher und Fachartikel zu .NET und Webtechniken veröffentlicht und spricht regelmäßig auf Fachkonferenzen. Sie können ihn und seine Kollegen für Schulungen, Beratungen, Coaching und Softwareentwicklung buchen.
E-Mail: **anfragen@IT-Visions.de** Website: **www.IT-Visions.de**
Weblog: **www.dotnet-doktor.de** Twitter: **@dotnetdoktor**