Spickzettel („Cheat Sheet“) C# 12.0 und .NET 8.0 – neue Sprachfeatures und neue APIs

Autor: Dr. Holger Schwichtenberg

www.IT-Visions.de
Dr. Holger Schwichtenberg

Neue und geänderte .NET SDK CLI-Befehle in .NET 8.0

Neuer Workload + neue Projektvorlage für WASI

```
dotnet workload install wasi-experimental
dotnet net wasiconsole
```

Löschen von Workloads

```
dotnet workload clean → Löscht nur verwaiste Workloads
dotnet workload clean --all → Löscht alle Workloads
```

Projektvorlagen mit AOT-Kompilierung anlegen

```
dotnet new console –aot
  → Vorlagen, die AOT unterstützen: api, grpc, worker
```

Kompaktere und farbige Ausgabe bei Build: Terminal Logger statt Console Logger (in Windows Terminal, nicht in alter Konsole!)

```
dotnet build /tl oder msbuild /tl
```

Warnungen vor NuGet-Paketen beim Build aktivieren in Projektdatei

```
<NuGetAudit>true</NuGetAudit>
<NuGetAuditLevel>moderate</NuGetAuditLevel>
  → erlaubte Werte: low, moderate, high, critical
```

Nun automatische **Release**-Kompilierung bei dotnet publish und dotnet pack. Debug-Kompilierung weiterhin möglich auf Wunsch

```
dotnet publish -c debug
```

/artifacts/bin, /artifacts/obj und /artifacts/publish statt /bin, /obj und /publish per Eintrag in Directory.Build.props

```
<ArtifactsPath>$(MSBuildThisFileDirectory)artifacts</ArtifactsPath>
```

Erstellen einer Container-Datei ohne Veröffentlichung bei Registry

```
dotnet publish -p PublishProfile=DefaultContainer
-p ContainerArchiveOutputPath=t:\meinimage.tar.gz
```

C# 12.0: Primärkonstruktoren

Angabe von Konstruktorparametern direkt nach Klassenname: Anders als bei den in C# 9.0 eingeführten Record-Typen erstellt der Primärkonstruktor aber keine öffentlichen Properties in der Klasse, sondern nur private Fields. Um öffentlich auf die im Primärkonstruktor übergebenen Daten zugreifen zu können, muss man die Konstruktorparameter für Zuweisungen verwenden!

```
public class Person(Guid id, string name) {
 // Verwendung der Primärkonstruktorparameter zur Initialisierung
 public Guid ID { get; init; } = id;
 public string Name { get; set; } = name;
 // Eigene Konstruktoren müssen den Primärkonstruktor rufen!
 public Person(string name) : this(Guid.NewGuid(), name) { }
 public override string ToString() {
  // Hier Properties statt Primärkonstruktorparameter verwenden!
  // Man würde sonst Namensänderungen nicht sehen!
  return $"Person {ID}: {Name}"; }
}
```

Erbende Klassen können ebenfalls Primärkonstruktoren verwenden

```
public class Autor(Guid id, string name, string website) :
            Person(id, name) {
 public string Website { get; set; } = website;
}
```

Verwendung wie bei Klassen ohne Primärkonstruktor

```
var a = new Autor(Guid.NewGuid(), "Holger Schwichtenberg",
"www.IT-Visions.de");
Console.WriteLine(a.ToString());
a.Name = "Dr. " + a.Name; // Schreibzugriff auf Property möglich
```

Console.WriteLine(a.Name);C# 12.0: Typaliase

Aliase für Typen mit using, z.B. Tupel, Arrays, Nullable<T>

```
using Author = (int ID, string Name, string Website);
using Author = (int, string, string); // Alternative ohne Namen
using Numbers = int[];
using DbInt = int?;
using AuthorSet = Author[]; // Aliase in Aliasen nicht erlaubt!
```

Für alle Dateien im Projekt mit global

```
global using Author = (int ID, string Name, string);
```

Verwendung im Code

```
Author hs = (42, "Dr. Holger Schwichtenberg", "www.IT-Visions.de");
Numbers numbers = new int[10];
DbInt n = LoadNumberFromDatabase(SQL, connectionString);
```

Nicht erlaubt bei Instanzierung von Arrays

```
Numbers numbers = new Numbers;
```

C# 12.0: Erweiterung für nameof()

Namensabruf von Instanzmitgliedern von Klassenmitgliedern war vor C# 12.0 nicht möglich in einigen Fällen (z.B. statische Mitglieder, Annotationen). Compiler-Fehler war: "CS0120: An object reference is required for the non-static field, method, or property". Beispiel: Name.Length funktioniert jetzt überall

```
public struct Person {
 public string Name;
// nameof() in folgenden bisher schon möglich:
 public string MemberName1() => nameof(Name);
 public string MemberName2() => nameof(Name.Length);
 public static string MemberName3() => nameof(Name);
 // bisher Fehler CS0120 bei nameof(Name.Length):
 public static string MemberName4() => nameof(Name.Length);
 [Description($"{nameof(StringLength)} liefert von {nameof(Name)}
die Eigenschaft {nameof(Name.Length)}")]
 public int StringLength() {
   return Name.Length;  }
}
```

C# 12.0: Vereinfachte Mengeninitialisierungen

Neue Syntax mit eckigen Klammern wie in JavaScript:

```
int[] a2 = [1, 2, 3];
ImmutableArray<int> c2 = [1, 2, 3];
Span<int> b2 = [1, 2, 3];
List<int> d2 = [1, 2, 3];
IEnumerable<int> e2 = [1, 2, 3];
```

Auch bei späterer Zuweisung

```
List<string> sites1, sites2, sites3 = ["dotnettraining.de"];
sites1 = ["www.IT-Visions.de", "www.dotnet-doktor.de"];
sites2 = []; // leere Liste
```

Verwendung mit Spread-Operator (Der Spread-Operator sorgt dafür, dass keine verschachtelte, sondern eine flache Liste entsteht!)

```
List<string> allList = [.. sites1, " www.dotnet8.de", .. sites2, sites3];
allList = [.. allList, "powershell-schulungen.de"]; // nun 5 Elemente
```

Eckige Klammern direkt als Methodenparameter

```
Print(["angular-schulungen.de", " powershell-schulungen.de"]);
```

C# 12.0: Optionale Lambda-Parameter

Optionale Parameter in Lambdas sind nun erlaubt:

```
var calc = (decimal x, decimal y, decimal z = 1) => (x + y) * z;
```

Das geht auch mit Statement Lambdas

```
var Print = (object text, ConsoleColor? color = null) => {
 if (color != null) Console.ForegroundColor = color.Value;
 Console.WriteLine(text);
 if (color != null) Console.ResetColor();
};
```

Aufruf

```
Print(calc(20, 1, 2), ConsoleColor.Green); // 42
Print(calc(40, 2), ConsoleColor.Green); // 42
```

.NET 8.0: Neue Zufallsfunktionen in System.Random

```
Span<string> Websites = ["www.IT-Visions.de", "angular-
schulungen.de", "dotnettraining.de", "www.dotnetframework.de",
"www.dotnet8.de", "dotnet-lexikon.de", "www.dotnet-doktor.de"];
```

Zufallsreihenfolge mit Shuffle()

```
Random.Shared.Shuffle(Websites);
```

Zufallsauswahl mit GetItems()

```
ReadOnlySpan<string> WebsitesReadOnly = Websites;
ReadOnlySpan<string> Zufallsauswahl3 =
Random.Shared.GetItems(WebsitesReadOnly, 3);
```

.NET 8.0: IsPrivilegedProcess()

System.Environment.IsPrivilegedProcess liefert true in diesen Fällen:

- **Windows:** NT AUTHORITY\LocalSystem, BUILTIN\Administrator, andere administrative Konten mit erhöhten Rechten
- **LINUX und macOS:** geteuid() == 0

Spickzettel („Cheat Sheet“) C# 12.0 und .NET 8.0 – neue Sprachfeatures und neue APIs

Autor: Dr. Holger Schwichtenberg

.NET 8.0: Neue/erweiterte Validierungsannotationen

```
public class SoftwareDeveloper {
 [AllowedValues("", "Dr.", "Prof. Dr.")] // nur diese Anreden erlaubt
 public string? Anrede { get; set; }
 [Length(0, 3)] // [Length] nun auch für Mengen: bis zu 3 Websites
 public List<string> Websites { get; set; } = new();
 [Base64String] // muss Base64 sein. Kein Schutz, nur Beispiel!
 public string? Kennwort { get; set; }
 [Range(0d, 120d, MinimumIsExclusive = true, MaximumIsExclusive
= true)] // Neue Parameter: > 0 und < 120 statt >= und <=
 public double Alter { get; set; }
}
```

.NET 8.0: Razor-Rendering in beliebigen Anwendungen

Beispiel: HTML-E-Mail-Versand per Razor-Template in Konsole:

EMailTemplate.razor

```
<html><body> <h2>Bestätigung</h2> <hr />
<p>Guten Tag @Name,</p> <p>.NET 8.0 ist am
<b>@Datum.ToLongDateString()</b> erschienen.</p>
<Footer/> @* Einbinden einer weiteren Razor-Komponente *@
</body></html>
@code {
 [Parameter]
 public string Name { get; set; }
 [Parameter]
 public DateTime Datum { get; set; } = new(2023, 11, 14);
}
```

Footer.razor

```
@{ var url = "https://www.IT-Visions.de"; }
<p>Mit freundlichen Grüßen</p>
<p><a href="@url">@url</a></p>
```

Dependency Injection zur Vorbereitung des Renderns

```
IServiceCollection services = new ServiceCollection();
services.AddLogging((b) => loggingBuilder
      .SetMinimumLevel(LogLevel.Trace).AddConsole()  );
IServiceProvider sp = services.BuildServiceProvider();
ILoggerFactory loggerFactory =
sp.GetRequiredService<ILoggerFactory>();
```

HTML-Rendering mit Razor Templates

```
await using var renderer = new HtmlRenderer(sp, loggerFactory);
var html = await renderer.Dispatcher.InvokeAsync(async () => {
  // Parameter zusammenstellen
  var name = "Dr. Holger Schwichtenberg";
  var pdic = new Dictionary<string, object>() {{ "Name", name }};
  var pv = ParameterView.FromDictionary(pdic);
  // Nun EMailTemplate.razor rendern
  var output = await
renderer.RenderComponentAsync<EMailTemplate>(pv);
  return output.ToHtmlString();  });
```

HTML-E-Mail senden

```
new ITVisions.Network.MailUtil().SendMail("dotnet@Microsoft.com",
"buero@IT-Visions.de", "Terminbestätigung", html, HTML: true);
```

Bestätigung

Guten Tag Dr. Holger Schwichtenberg,

.NET 8.0 ist am **Dienstag, 14. November 2023** erschienen.

Mit freundlichen Grüßen

www.IT-Visions.de

.NET 8.0: Dependency Injection mit Schlüsseln

Beispiel:

```
interface IDataProvider {  public string Name { get; set; } }
class LocalData : IDataProvider {
 public string Name { get; set; } = "Lokale Daten"; }
class RemoteData : IDataProvider {
 public string Name { get; set; } = "Daten von WebAPI"; }
class RemoteDataConsumer([FromKeyedServices("remote")]
IDataProvider data) {
 public override string ToString() {  return data.Name; } }
```

Aufbau des Dependency Injection-Containers

```
var services = new ServiceCollection();
services.AddKeyedSingleton<IDataProvider, LocalData>("local");
services.AddKeyedSingleton<IDataProvider,
RemoteData>("remote");
services.AddSingleton<RemoteDataConsumer>();
var serviceProvider = services.BuildServiceProvider();
```

Nutzung

```
var service1 =
serviceProvider.GetKeyedService<IDataProvider>("remote");
Console.WriteLine(service1.Name); // Daten von WebAPI
var service2 =
serviceProvider.GetRequiredService<RemoteDataConsumer>();
Console.WriteLine(service2.ToString()); // Daten von WebAPI
```

Nicht möglich, da IAppService ein Service mit Schlüssel ist ("No service for type 'IAppService' has been registered."):

```
serviceProvider.GetRequiredService<IAppService>();
```

.NET 8.0: Zeitabstraktion mit System.TimeProvider

```
private class FakeTimeProvider : TimeProvider {
  private readonly DateTimeOffset _date;
  public FakeTimeProvider(DateTimeOffset date) {  _date = date;  }
  public override DateTimeOffset GetUtcNow() {  return _date;  }
  public override ITimer CreateTimer(TimerCallback callback, object
state, TimeSpan dueTime, TimeSpan period) =>
base.CreateTimer(callback, state, new TimeSpan(0, 0, 0), new
TimeSpan(0, 0, 1));
}
```

Nutzung

```
void PrintTime(TimeProvider timeProvider) {
   Console.WriteLine("Lokale Zeit: " + timeProvider.GetLocalNow());
   Console.WriteLine("UTC: " + timeProvider.GetUtcNow());  }
// Reguläre Systemzeit übergeben
PrintTime(TimeProvider.System);
// Vorgetäuschte Zeit übergeben
FakeTimeProvider fakeTimeProvider =
new(DateTime.Parse("14.11.2023 18:00:00"));
PrintTime(fakeTimeProvider); // 14.11.2023 18 Uhr
// Keine Verzögerung, da Fake!
await fakeTimeProvider.Delay(new TimeSpan(42, 21, 0));
// FakeTimer verkürzt das Intervall von 10 auf 1 Sekunde
fakeTimeProvider.CreateTimer((x) => callback(fakeTimeProvider),
null, new TimeSpan(0, 0, 0), new TimeSpan(0, 0, 10));
```

.NET 8.0: Parse() und TryParse() in Klasse IPNetwork

Neue statische Methode Parse()

```
IPNetwork network1 = IPNetwork.Parse("192.168.0.1/32");
Console.WriteLine(network1.BaseAddress); // 192.168.0.1
Console.WriteLine(network1.PrefixLength); // 32
```

Neue statische Methode TryParse()

```
bool b = IPNetwork.TryParse("192.168.0.1/32", out var network2);
Console.WriteLine(b); // true
Console.WriteLine(network2.BaseAddress); // 192.168.0.1
Console.WriteLine(network2.PrefixLength); // 32
```

Webadressen

Website des Autors zu .NET 8.0
www.dotnet8.de
Neuerungen in .NET 8.0
learn.microsoft.com/en-us/dotnet/core/whats-new/dotnet-8
Neuerungen in C# 12.0
learn.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-12
Neuerungen in ASP.NET Core 8.0
learn.microsoft.com/en-us/aspnet/core/release-notes/aspnetcore-8.0
Neuerungen in Entity Framework Core 8.0
learn.microsoft.com/ef/core/what-is-new/ef-core-8.0/whatsnew
Breaking Changes in .NET 8.0
learn.microsoft.com/en-us/dotnet/core/compatibility/8.0

Über den Autor

Dr. Holger Schwichtenberg (MVP seit 20 Jahren) 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 Beratungen, Schulungen, Coaching und Softwareentwicklung buchen.
E-Mail: **anfragen@IT-Visions.de** Website: **www.IT-Visions.de**
Weblog: **www.dotnet-doktor.de** Twitter: **@dotnetdoktor**