5Minds

.NET 6 und C# 10

Top Features der neuen LTS Version

Über mich

Martin Pöpel

- Senior Software Developer
- Certified Professional for Software Architecture (CPSA-F)
- Seit über 15 Jahren im .NET Framework zu Hause
- Interessiert an: Software Architektur, Cloud Development, Machine Learning, .NET

@MaddinDev

github.com/M4ddinPoe

Ab Heute verfügbar!!!

.NET 6

https://dotnet.microsoft.com/download/dotnet/6.0

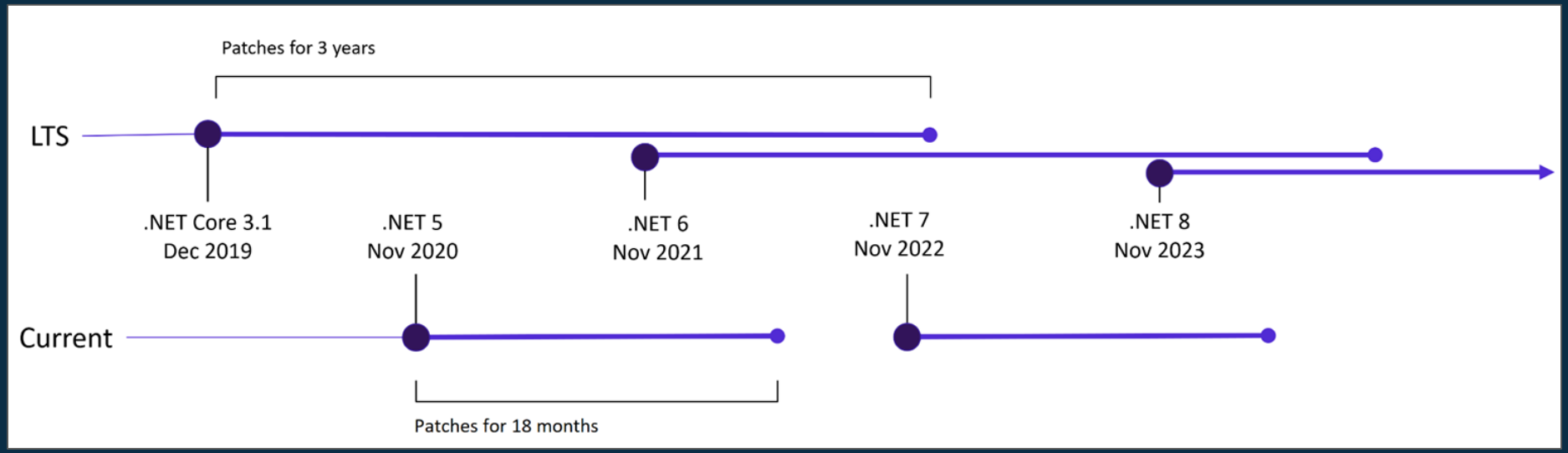
Visual Studio 22

https://visualstudio.microsoft.com/de/vs/

Verfügbarkeit und Kompatibilität

- Visual Studio 2022
- Visual Studio 2019 for Mac (v8.10)
- Rider (2021.3 EAP)
- IIS runtime support (ASP.NET Core Module v2) v16.0.21273.0

.NET 6 (LTS) Lifetime



https://dotnet.microsoft.com/platform/support/policy/dotnet-core

C# 10

Null Parameter Checking {

```
1
2       public bool DoCheck(string text)
3       {
4           ArgumentNullException.ThrowIfNull(text);
5
6           return text == "Test";
7       }
8
9
10
11
12
13
14
15
16
11
17
18
19
```

}

Record Structs {

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
11
17
18
19
```

```
    Aktuell gibt es Structs und Records in C#

    public struct Rectangle
    {
       public int Height { get; init; }

       public int Width { get; init; }
    }

    public record Person(string FirstName, string LastName);
```

}

Record Structs {

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
11
17
18
19
```

```
    Aktuell gibt es Structs und Records in C#

    public struct Rectangle
    {
       public int Height { get; init; }

       public int Width { get; init; }
    }

    public record Person(string FirstName, string LastName);

    C#10 gibt uns Record Structs
    public record struct Point(double X, double Y, double Z);
```

}

Record Types can seal ToString {

```
1
2       public record Person(string FirstName, string LastName)
3       {
4           public sealed override string ToString()
5           {
6               return $"Hi my name is: {FirstName} {LastName}";
7           }
8       }
9
10      public record Client(string FirstName, string LastName, string EMail)
11          : Person(FirstName, LastName)
12      {
13          public override string ToString()
14          {
15              return $"Hi my name is: {FirstName} {LastName}";
16          }
11      }
17
18
19
```

CS0239 'Client.ToString()': cannot override inherited member 'Person.ToString()' because it is sealed

}

Record Types can seal ToString {

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
11
17
18
19
```

```
    public record Person(string FirstName, string LastName)
    {
        public virtual string ToFullName()
        {
            return string.Empty;
        }
    }

    public record Client(string FirstName, string LastName, string EMail)
        : Person(FirstName, LastName)
    {
        public sealed override string ToFullName()
        {
            return base.ToString();
        }
    }

    public record Customer(string FirstName, string LastName, string EMail) : Client(FirstName, LastName, EMail)
    {
        public override string ToFullName()
        {
            return $"Hello, I'am {FirstName} {LastName} and you can reach me through: {EMail}";
        }
    }
```

}

Contant Interpolated Strings {

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
11
17
18
19
```

```
    Aktuell gibt es keine Interpolation für konstante Strings

    private const string BaseUrl = "https://www.5minds.de/";

    private const string TeamUrl = BaseUrl + "/team";

    private const string CareerUrl = BaseUrl + "/karriere";
```

}

Contant Interpolated Strings {

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
11
17
18
19
```

```
    C#10 bringt diese Möglichkeit mit

    private const string BaseUrl = "https://www.5minds.de/";

    private const string TeamUrl = $"{BaseUrl}/team";

    private const string CareerUrl = $"{BaseUrl}/karriere";
```

}

Attributes support generics {

Vor C#10:

```
public class OldAttribute : Attribute
{
   public OldAttribute(Type type)
   {
   }
}
```

Mit C#10:

```
public class NewAttribute<T> : Attribute
{
   public T MyType { get; set; }
}
```

}

Lambda Improvements {

Declaration vor C#10:

```
Func<string> hello = () => "Hello!";
```

Mit C#10 ist es möglich var zu benutzen:

```
var hello = () => "Hello!";
```

Ebenfall ist eine null Rückgabe in c# 10 möglich

```
var hello = string () => null;
```

}

Extended Property Patterns {

```
var subCompany = new Company("Google");
var company = new Company("Alphabet", subCompany);

Vor C#10 komplizierte Abfrage:

if (company is { SubCompany: { Name: "Google" } })
{
    // ...
}

Mit C#10 deutlich einfacher:

if (company is { SubCompany.Name: "Google"  })
{
    // ...
}
```

}

Structure Type Improvements {

```
public struct Rectangle
{
    Vor C#10 waren Parameterlose Konstruktoren in Stucts nicht erlaubt:
    public Rectangle()
    {
        this.Height = 0;
        this.Width = 0;
    }

    public Rectangle(int height, int width)
    {
        Height = height;
        Width = width;
    }

    public int Height { get; init; }

    public int Width { get; init; }
}
```

}

Structure Type Improvements {

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
11
17
18
19
```

Auch das neu Erzeugen mit with war nicht erlaubt:

```
var rectangle = new Rectangle(100, 100);
var rectangle2 = rectangle with { Width = 200 };
```

}

Assignment and declaration in same deconstruction {

```
internal class Point
{
    public Point(int x, int y, int z)
    {
        X=x; Y=y; Z=z;
    }

    public int X { get; init; }
    public int Y { get; init; }
    public int Z { get; init; }

    internal void Deconstruct(out int x, out int y, out int z)
    {
        x=X; y=Y; z=Z;
    }
}
Ist vor C# 10 nicht möglich:
var point = new Point(2, 3, 5);
(int height, int width) = point;
```

}

Global usings {

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
11
17
18
19
```

```
global using System;
```

In einer *.cs Datei gespeichert ist dieses using für das gesamte Projekt gültig

}

File Scoped Namespace {

```
namespace FileScopedNamespace;

internal class Point
{
    public Point(int x, int y, int z)
    {
        X=x; Y=y; Z=z;
    }

    public int X { get; init; }
    public int Y { get; init; }
    public int Z { get; init; }

    internal void Deconstruct(out int x, out int y, out int z)
    {
        x=X; y=Y; z=Z;
    }
}
```

}

.NET 6

Minimal Api {

Mit dem Kommando:

```
dotnet new web -o MinApi
```

Wird folgende einfache API erzeugt.

```
using Microsoft.AspNetCore.Builder;

var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.MapGet("/hello", () => "Hello, World!");

app.Run();
```

}

Hot Reload - Was ist das?

- Mit Hot Reload können Anwendungen geändert werden während sie laufen

- Funktioniert mit:
  - WPF,
  - Windows Forms,
  - .NET MAUI previews,
  - ASP.NET Core apps code-behind,
  - Console applications,
  - WinUI 3 (managed debugger required)
  - und vielen mehr

```
Neue Api's {
  // - Prority Queue
  // - Parallel.ForEachAsync
  // - Confguration.GetRequiredSection
  // - PeriodicalTimer
  // - IEnumerableChunks
  // - *ByMethods
  // - OverrideDefaultInFirstorDefault
}
```

Priority Queue {

```
using System.Collections.Generic;

PriorityQueue<string, int> queue = new PriorityQueue<string, int>();
queue.Enqueue("Item A", 0);
queue.Enqueue("Item B", 60);
queue.Enqueue("Item C", 2);
queue.Enqueue("Item D", 1);

while (queue.TryDequeue(out string item, out int priority))
{
   Console.WriteLine($"Popped Item : {item}. Priority Was : {priority}");
}
```

}

Priority Queue {

```
Popped Item : Item A. Priority Was : 0
Popped Item : Item D. Priority Was : 1
Popped Item : Item C. Priority Was : 2
Popped Item : Item B. Priority Was : 60
```

}

Parallel.ForEach {

```
1  var userHandlers = new []
2  {
3     "users/okyrylchuk",
4     "users/davidfowl"
5  };
6
7  using HttpClient client = new()
8  {
9     BaseAddress = new Uri("https://api.github.com"),
10 };
11 client.DefaultRequestHeaders.UserAgent.Add(new ProductInfoHeaderValue("DotNet", "6"));
12
13 ParallelOptions parallelOptions = new()
14 {
15    MaxDegreeOfParallelism = 3
16 };
17
11 await Parallel.ForEachAsync(userHandlers, parallelOptions, async (uri, token) =>
18 {
19    var user = await client.GetFromJsonAsync<GitHubUser>(uri, token);

      Console.WriteLine($"Name: {user.Name}\nBio: {user.Bio}\n");
   });
}
```

Configuration.GetRequiredSection {

We added a helper to make it easier to throw if a required section of configuration is missing:

```
var configuration = new ConfigurationManager();
var options = new MyOptions();

// This will throw if the section isn't configured
configuration.GetRequiredSection("MyOptions").Bind(options);

class MyOptions
{
   public string? SettingValue { get; set;}
}
```

}

https://devblogs.microsoft.com/dotnet/new-dotnet-6-apis-driven-by-the-developer-community/

PeriodicTimer {

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
11
17
```

Eine moderne timer API.

Die komplett Asynchron ist und damit viele Probleme anderer Timer API’s löst.

```
var timer = new PeriodicTimer(TimeSpan.FromSeconds(1));


while (await timer.WaitForNextTickAsync())
{
    Console.WriteLine(DateTime.UtcNow);
}
```

}

https://devblogs.microsoft.com/dotnet/new-dotnet-6-apis-driven-by-the-developer-community/

IEnumerable Chunks {

```
1
2
3       A new helper to chunk any IEnumerable into batches
4
5
6       int chunkNumber = 1;
7       foreach (int[] chunk in Enumerable.Range(0, 9).Chunk(3))
8       {
9          Console.WriteLine($"Chunk {chunkNumber++}");
10         foreach (var item in chunk)
11         {
12             Console.WriteLine(item);
13         }
14      }
15
16
11
17
```

}

https://devblogs.microsoft.com/dotnet/new-dotnet-6-apis-driven-by-the-developer-community/

*ByMethods {

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
11
17
```

```
    In der alten Zeit:


    var people = GetPeople();

    var oldestAge = people.Max(person => person.Age);
    var youngestAge = people.Min(person => person.Age);

    var oldest = people.FirstOrDefault(p => p.Age == oldestAge);
    var youngest = people.FirstOrDefault(p => p.Age == youngestAge);

    Console.WriteLine($"The oldest person is {oldest.Age}");
    Console.WriteLine($"The youngest person is {youngest.Age}");
```

}

https://devblogs.microsoft.com/dotnet/new-dotnet-6-apis-driven-by-the-developer-community/

*ByMethods {

```
1
2
3   Heute: A new helper to chunk any IEnumerable into batches
4
5   var people = GetPeople();
6
7
8   var oldest = people.MaxBy(person => person.Age);
9   var youngest = people.MinBy(person => person.Age);
10
11  Console.WriteLine($"The oldest person is {oldest.Age}");
12  Console.WriteLine($"The youngest person is {youngest.Age}");
13
14
15
16
11
17
```

}

https://devblogs.microsoft.com/dotnet/new-dotnet-6-apis-driven-by-the-developer-community/

OverrideDefaultInFirstOrDefault {

Damals:

```
var names = new List<string> {"Max", "Lea", "Tom", "Isa"};
string longNames = names.FirstOrDefault(name => name.Length > 3) ?? string.Empty;
```

}

https://devblogs.microsoft.com/dotnet/new-dotnet-6-apis-driven-by-the-developer-community/

OverrideDefaultInFirstOrDefault {

Heute:

```
var names = new List<string> {"Max", "Lea", "Tom", "Isa"};
string name = names.FirstOrDefault(name => name = "Maddin", " -NA- ");
```

}

https://devblogs.microsoft.com/dotnet/new-dotnet-6-apis-driven-by-the-developer-community/

Blazor {

```
// - Ahead of Time Compilation (Blazor Wasm)
// - Error Boundaries
// - Improved Prerendering With Preserved State
```

}

Ahead of time compilation

- Blazor Wasm Applikationen werden im Browser interpretiert
- Hierfür wurde ein IL Interpreter in WASM implementiert
- Ahead-Of-Time (AOT) kompiliert direkt nach Wasm

Error Boundaries

- Bei einer Exception ist bisher die komplette Applikation abgestürzt
- Mit .NET 6 lassen sich Teile der App in Error Boundaries einteilen
- Abstürze betreffen dadurch nur die Boundary

Error Boundaries {

```
<h1>My Application</h1>
<div>
   <ErrorBoundary>
       <ChildContent>
           <SomeComponent />
       </ChildContent>
       <ErrorContent>
           <p class="bad-error">
               Hier ist was schief gegangen...
           </p>
       </ErrorContent>
   </ErrorBoundary>
</div>
```

}

https://devblogs.microsoft.com/dotnet/new-dotnet-6-apis-driven-by-the-developer-community/

Improved Prerendering with preserved state

- Blazor-Anwendungen können auf dem Server vorgerendert werden, um die Ladezeit zu optimieren
- Der “state” der Seite geht dabei im Browser verloren und muss wieder nachgeladen werden
- Dies kann zu einem “flackern” der UI führen
- .NET 6 führt hierfür den `<preserve-component-state />` Tag-Helper ein

Improved Prerendering with preserved state

Der State wir übertragen in dem ein HTML Kommentar der den verschlüsselten State enthält eingefügt wird

```
<!--Blazor-Component-State:CfDJ8IZEzFk/KP1DoDRucCE
6nSjBxhfV8XW7LAhH9nkG90KnWp6A83ylBVm+Fkac8gozf2hBP
DSQHeh/jejDrmtDEesKaoyjBNs9G9EDDyyOe1o1zuLnN507mK0
Bjkbyr82Mw83mIVl21n8mxherLqhyuDH3QoHscgIL7rQKBhejP
qGqQLj0WvVYdvYNc6I+FuW4v960+1xiF5XZuEDhKJpFODIZIE7
tIDHJh8NEBWAY5AnenqtydH7382TaVbn+1e0oLFrrSWrNWVRbJ
QcRUR5xpa+yWOZ7U52iudA27ZZr5Z8+LrU9/QVre3ehO+WSW7D
Z/.../g8VejWlSUiforHpVjPJojsfYfmeLOjRoSPBTQZ
Q0LL4ie/QFmKXY/TI7GjJCs5UuPM=-->
```

https://asp.net-hacker.rocks/2021/06/15/aspnetcore6-10-blazor-preserve-prerendered-state.html

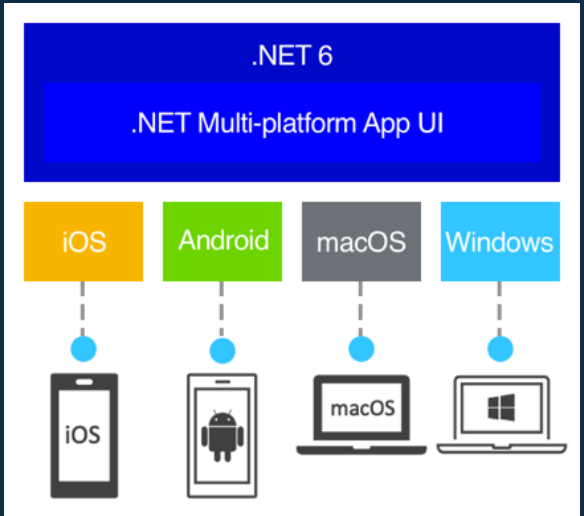
```
MAUI (Multi-Platform
      App UI) {

  // - Ein Ausblick
}
```

MAUI (Multi-Platform App UI)

Leider nicht in .NET 6

MAUI (Multi-Platform App UI)

- Weiterentwicklung von Xamarin Forms
- Ein Projekttyp für Android, iOS und MacCatalyst



https://docs.microsoft.com/en-us/dotnet/maui/what-is-maui

Danke!