C# 10

A Sneaky Peek 👀

HELLO!

I am Stuart Lang

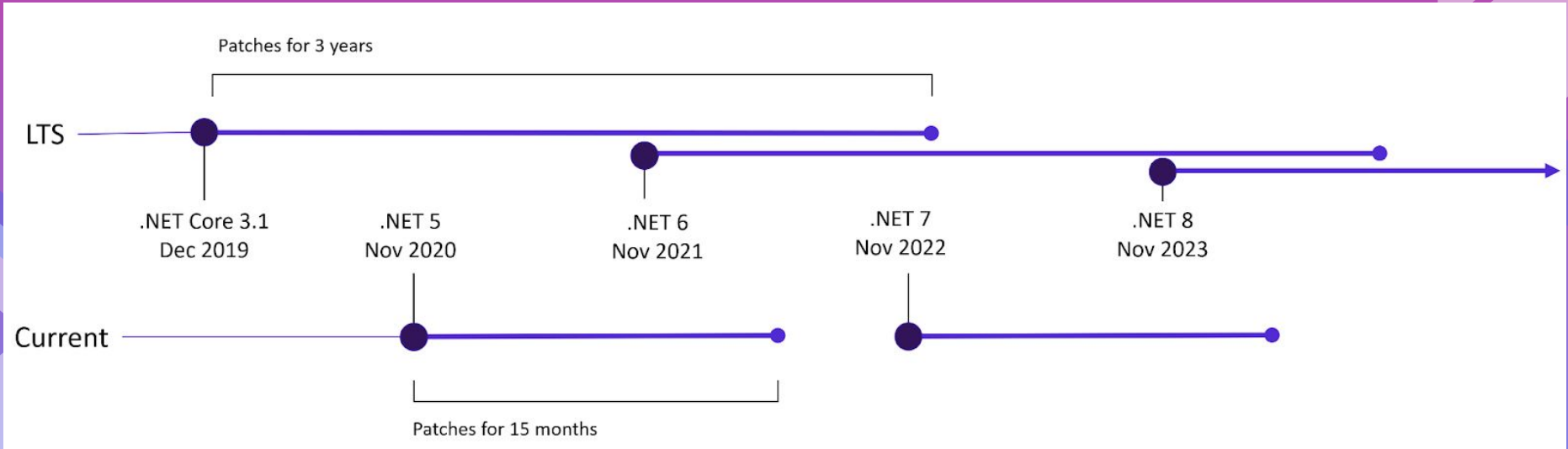
You can find me at

stu.dev

DMs open!

Microsoft® Most Valuable Professional

Quick Recap


Patches for 3 years
LTS
.NET Core 3.1
Dec 2019
.NET 5
Nov 2020
.NET 6
Nov 2021
.NET 7
Nov 2022
.NET 8
Nov 2023
Current
Patches for 15 months

C# 9 Recap

C# 9 Recap

- Top-level Statements
- Records
- Pattern Matching Enhancements
- Source Generators

Code along!

- .NET 6 Preview 7
  - https://aka.ms/dotnet-download
- LINQPad 6 beta
  - Go to Edit | Preferences > Query
  - Check "Use Roslyn Daily build"
- https://sharplab.io
  - Select the Roslyn branch dropdown
  - Pick “main”

File Scoped Namespaces

```
namespace Company.Project
{
    public class Person
    {
        public string Name { get; set; }
        public int Age { get; set; }
    }
}
```

```
namespace Company.Project
{
    public class Person
    {
        public string Name { get; set; }
        public int Age { get; set; }
    }
}
```

```
namespace Company.Project;

public class Person
{
    public string Name { get; set; }
    public int Age { get; set; }
}
```

Global Usings

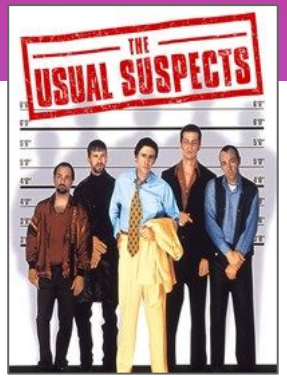
```
namespace Company.Project;

public class Person
{
    public string Name { get; set; }
    public int Age { get; set; }
}
```

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace Company.Project;

public class Person
{
    public string Name { get; set; }
    public DateTime DateOfBirth { get; set; }
    public IReadOnlyCollection<string> FavouriteColors { get; set; }
}
```



C# GlobalUsings.cs

```
1  global using System;
2  global using System.Collections.Generic;
3  global using System.Threading.Tasks;
```

C# Person.cs

```
1  namespace Company.Project;
2
3  public class Person
4  {
5      public string Name { get; set; }
6      public DateTime DateOfBirth { get; set; }
7      public IReadOnlyCollection<string> FavouriteColors { get; set; }
8  }
```

C# GlobalUsings.cs

```
1  global using System;
2  global using System.Collections.Generic;
3  global using System.Threading.Tasks;
4  global using static System.Console;
```

```
<Project Sdk="Microsoft.NET.Sdk">

    <PropertyGroup>
        <OutputType>Exe</OutputType>
        <TargetFramework>net6.0</TargetFramework>
        <ImplicitUsings>enable</ImplicitUsings>
    </PropertyGroup>

</Project>
```

```
<ItemGroup Condition="'$(ImplicitUsings)' == 'true' Or '$(ImplicitUsings)' == 'enable'">
    <Using Include="System" />
    <Using Include="System.Collections.Generic" />
    <Using Include="System.IO" />
    <Using Include="System.Linq" />
    <Using Include="System.Net.Http" />
    <Using Include="System.Threading" />
    <Using Include="System.Threading.Tasks" />
</ItemGroup>
```

```
// <auto-generated/>
global using global::System;
global using global::System.Collections.Generic;
global using global::System.IO;
global using global::System.Linq;
global using global::System.Net.Http;
global using global::System.Threading;
global using global::System.Threading.Tasks;
```

Console app

```
// <auto-generated/>
global using global::Microsoft.Extensions.Configuration;
global using global::Microsoft.Extensions.DependencyInjection;
global using global::Microsoft.Extensions.Hosting;
global using global::Microsoft.Extensions.Logging;
global using global::System;
global using global::System.Collections.Generic;
global using global::System.IO;
global using global::System.Linq;
global using global::System.Net.Http;
global using global::System.Threading;
global using global::System.Threading.Tasks;
```

```
// <auto-generated/>
global using global::Microsoft.AspNetCore.Builder;
global using global::Microsoft.AspNetCore.Hosting;
global using global::Microsoft.AspNetCore.Http;
global using global::Microsoft.AspNetCore.Routing;
global using global::Microsoft.Extensions.Configuration;
global using global::Microsoft.Extensions.DependencyInjection;
global using global::Microsoft.Extensions.Hosting;
global using global::Microsoft.Extensions.Logging;
global using global::System;
global using global::System.Collections.Generic;
global using global::System.IO;
global using global::System.Linq;
global using global::System.Net.Http;
global using global::System.Net.Http.Json;
global using global::System.Threading;
global using global::System.Threading.Tasks;
```

Web app

```
<ItemGroup>
    <Using Include="MyOwnNamespace.Models" />
    <Using Include="MyOwnNamespace.Thing" Alias="SomeAlias" />
    <Using Include="System.Console" Static="true" />
</ItemGroup>
```

Record Structs

```
record Coordinates(double Lat, double Long);
```

```
record Coordinates
{
    public double Lat { get; init; }
    public double Long { get; init; }
}
```

```
record Coordinates(double Lat, double Long);
```

```
record class Coordinates(double Lat, double Long);
```

```
record struct Coordinates(double Lat, double Long);
```

```
readonly record struct Coordinates(double Lat, double Long);
```

SharpLab Example

```
var startPosition = new Coordinates(2, 3);
var newPosition = startPosition with { Lat = 5 };
```

More on Records

Records Deep Dive: What, Why and How - David Wengier - NDC Melbourne 2021:
https://www.youtube.com/watch?v=lxwNcg2q1-Y

Better Lambdas

Before...

```
var f1 = (Func<string, int>)(input ⇒ int.Parse(input));
var f2 = new Func<string, int>(input ⇒ int.Parse(input));
```

Natural Inferred Types for Lambdas

```
// Won't work 💥
var f1 = () => null;        // error: no natural type
var f2 = x => { };          // error: no natural type
var f3 = x => x;            // error: no natural type

// Works 😎
var f4 = () => 1;           // Func<int>
var f5 = (int a) => { };    // Action<int>
var f6 = async () => 1;     // Func<Task<int>>
var f7 = 1.GetHashCode;     // Func<int>
var f8 = 1.ToString;        // error: multiple ToString methods
```

Explicit Lambda Return Types

```
var f1 = () ⇒ null;          // error: no natural type
var f1 = string () ⇒ null;   // error: Func<string>
```

Lambda Attributes

```
var f = [Something] OrderResult ([SomethingElse] Order order) ⇒ null;
```

Minimal APIs

With our powers combined...

The Simplest API

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

app.MapGet("/", () ⇒ "Hello World!");
app.Run();
```

```
app.MapGet("/todos/{id}", async (int id, TodoDb db) ⇒
    {
        return await db.Todos.FindAsync(id)
            is Todo todo
            ? Results.Ok(todo)
            : Results.NotFound();
    })
    .WithName("GetTodoById")
    .WithTags("TodoApi")
    .Produces<TodoService>()
    .Produces(StatusCodes.Status404NotFound);
```

Source: Damian Edwards

```
app.MapPost("/todos", async (TodoService todo, TodoDb db) ⇒
    {
        if (!MinimalValidation.TryValidate(todo, out var errors))
            return Results.ValidationProblem(errors);

        db.Todos.Add(todo);
        await db.SaveChangesAsync();

        return Results.Created($"/todo/{todo.Id}", todo); ;
    })
    .WithName("AddTodo")
    .WithTags("TodoApi")
    .ProducesValidationProblem()
    .Produces<TodoService>(StatusCodes.Status201Created);
```

Source: Damian Edwards

```
// Existing
public static IEndpointConventionBuilder! MapGet(
    this IEndpointRouteBuilder! endpoints,
    string! pattern,
    RequestDelegate! requestDelegate)
{
    // ...

// New
public static MinimalActionEndpointConventionBuilder! MapGet(
    this IEndpointRouteBuilder! endpoints,
    string! pattern,
    Delegate! action)
{
    // ...
```

Caller Expression Attribute

Caller Attributes Recap

```
WriteCallerMethodName(); // <Main>$

void WriteCallerMethodName([CallerMemberName] string memberName = null)
{
    Console.WriteLine(memberName);
}
```

- [CallerMemberName]
- [CallerFilePath]
- [CallerLineNumber]

CallerArgementExpression

```
object something = null;
AssertIsNotNull(something as string); // `something as string` was null

void AssertIsNotNull(string input, [CallerArgumentExpression("input")] string inputExpression = null)
{
    if (input is null) throw new Exception($"`{inputExpression}` was null");
}
```

```
/// <summary>Throws an <see cref="ArgumentNullException"/> if <paramref name="argument"/> is null.</summary>
/// <param name="argument">The reference type argument to validate as non-null.</param>
/// <param name="paramName">The name of the parameter with which <paramref name="argument"/> corresponds.</param>
public static void ThrowIfNull([NotNull] object? argument, [CallerArgumentExpression("argument")] string? paramName = null)
{
    if (argument is null)
    {
        Throw(paramName);
    }
}

[DoesNotReturn]
private static void Throw(string? paramName) =>
    throw new ArgumentNullException(paramName);
```

```
public int Parse(string input)
{
    ArgumentNullException.ThrowIfNull(input);
```

String Interpolation Improvements

Const Interpolation

```
const string BasePath = "/api";

[Route($"{BasePath}/path")]
public IActionResult Index()
{
```

String Interpolation Handler

```
[InterpolatedStringHandler]
public ref struct DefaultInterpolatedStringHandler
{
    public DefaultInterpolatedStringHandler(int literalLength, int formattedCount);
    public string ToStringAndClear();

    public void AppendLiteral(string value);

    public void AppendFormatted<T>(T value);
    public void AppendFormatted<T>(T value, string? format);
    public void AppendFormatted<T>(T value, int alignment);
    public void AppendFormatted<T>(T value, int alignment, string? format);

    public void AppendFormatted(ReadOnlySpan<char> value);
    public void AppendFormatted(ReadOnlySpan<char> value, int alignment = 0, string? format = null);

    public void AppendFormatted(string? value);
    public void AppendFormatted(string? value, int alignment = 0, string? format = null);

    public void AppendFormatted(object? value, int alignment = 0, string? format = null);
}
```

String Interpolation Handler

```
var first = "C#";
var second = 10;
Log($"{first} and {second}");
```

```
var first = "C#";
var second = 10;
Log(string.Format("{0} and {1}", first, second));
```

String Interpolation Handler

```
var first = "C#";
var second = 10;
Log($"{first} and {second}");
```

```
var first = "C#";
var second = 10;
var defaultInterpolatedStringHandler = new DefaultInterpolatedStringHandler(5, 2);
defaultInterpolatedStringHandler.AppendFormatted(first);
defaultInterpolatedStringHandler.AppendLiteral(" and ");
defaultInterpolatedStringHandler.AppendFormatted(second);
Log(defaultInterpolatedStringHandler.ToStringAndClear());
```

Extended Property Pattern

Extended Property Pattern

```
// Old: Nested Propertey Pattern
if (e is MethodCallExpression {Method: {Name: "MethodName"}})
{

}

// New: Extended Propertey Pattern
if (e is MethodCallExpression {Method.Name: "MethodName"})
{

}
```

Required Properties

C# 11

```
class Person
{
    public string Name { get; set; } // Must be set!
    public string? PetsName { get; set; }
}
```

```
class Person
{
    public required string Name { get; set; }
    public string? PetsName { get; set; }
}
```

!! (Bang Bang)

C# 11

```
// s is non-nullable
public void MyLovelyMethod(string s)
{

}
```

```
// s is non-nullable
public void MyLovelyMethod(string s)
{
    if (s is null) throw new ArgumentNullException(nameof(s));
}
```

```
// s is non-nullable
public void MyLovelyMethod(string s!!)
{

}
```

Raw String Literals

Raw String Literals

```
var xml = """
          <element attr="content">
            <body>
            </body>
          </element>
          """;
```

Generic Attributes

Generic Attributes

```
[ProducesResponseType<TEntity>(200)]
public virtual Task<IActionResult> Get()
{

}
```

Static Abstracts in Interfaces

```
<Project Sdk="Microsoft.NET.Sdk">

    <PropertyGroup>
        <EnablePreviewFeatures>true</EnablePreviewFeatures>
        <LangVersion>preview</LangVersion>
        <OutputType>Exe</OutputType>
        <TargetFramework>net6.0</TargetFramework>
    </PropertyGroup>

    <ItemGroup>
        <PackageReference Include="System.Runtime.Experimental" Version="6.0.0-preview.7.21377.19" />
    </ItemGroup>

</Project>
```

```
// Interface specifies static properties and operators
interface IAddable<T> where T : IAddable<T>
{
    static abstract T Zero { get; }
    static abstract T operator +(T t1, T t2);
}
```

```
// Classes and structs (including built-ins) can implement interface
struct Int32 : …, IAddable<Int32>
{
static Int32 I.operator +(Int32 x, Int32 y) ⇒ x + y; // Explicit
public static int Zero ⇒ 0;                          // Implicit
}

// Generic algorithms can use static members on T
public static T AddAll<T>(T[] ts) where T : IAddable<T>
{
    T result = T.Zero;                    // Call static operator
    foreach (T t in ts) { result += t; } // Use `+`
    return result;
}

// Generic method can be applied to built-in and user-defined types
int sixtyThree = AddAll(new [] { 1, 2, 4, 8, 16, 32 });
```

More Resources

- .NET 5 & C# 9 Talk
- C# Language Design
- Language Feature Status
- Mads Torgersen Talk - C# 10

THANKS!

Any questions?

You can find me at:

@stuartblang · stu.dev