C# 11

Rainer Stropek | @rstropek

C# 11 Status

- C# 11 is still under development
- This presentation is based on the Language Feature Status on GitHub
-
    - Date: Februar 20th, 2022
- Interesting reads
    - C# Language Proposals
    - Language Design Meetings protocols
- Code snippets have been verified using sharplab.io
    - See links on right bottom corner of slides
    - Some code samples already work with VS 2022 Preview

Newlines in Interpolations

Newline in Interpolations

```csharp
using System;
using System.Linq;

var numbers = new[] { 1, 2, 3, 4, 5 };

Console.WriteLine($"{
    numbers
        .Where(n => n % 2 == 0)
        .Select(n => n * n)
        .Sum()
    }");
```

https://sharplab.io/#gist:ead8b763599fa023953adecc5c0e770a (main)

Pattern Matching Enhancements

List patterns! Finally... 😉

List Patterns

```csharp
using System;
using System.Collections.Generic;

var numbers = new List<int>() { 1, 2, 3, 5, 8 };

// List pattern
if (numbers is [ 1, 2, 3, 5, 8 ])
{
    Console.WriteLine("Fibonacci");
}

// Property pattern
if (numbers is [ var first, 2, 3, 5, var last ] && first == 1 && last == 8)
{
    Console.WriteLine("Very special Fibonacci");
}

// Slice pattern
Console.WriteLine(numbers switch {
    [ 1, .., var sl, 8 ] => $"Starts with 1, ends with 8, and 2nd last number is {sl}",
    [ 1, .., var sl, > 8 or < 0 ] =>
        $"Starts with 1, ends with something > 8 or < 0, and 2nd last number is {sl}",
    [ 1, _, _, .. ] => "Starts with 1 and is at least 3 long",
    [ 1, .. ] => "Starts with 1 and is at least 1 long",
    _ => "WAT?"
});
```

https://sharplab.io/#gist:dc754b51dcec565451fbdcc39d84b82b main

Simplified Null Checking

We have to talk... 😁🤪🤮🤬

Null Checking

```
#nullable enable

using System;

string BuildFullName(Person p)
{
    // We compile with #nullable enable, so we rely on
    // p and p.FirstName to not be null.
    if (p.FirstName.Length == 0) {
        return p.FirstName;
    }

    return $"{p.LastName}, {p.FirstName}";
}

var p = new Person("Foo", "Bar");
Console.WriteLine(BuildFullName(p));

// Pass null and supress nullable warning (shouldn't do that,
// but sh... sometimes happen).
Console.WriteLine(BuildFullName(null!));

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

https://sharplab.io/#gist:7d9a29458a9a9be8a569f4b1753969a1 main

Null Checking

```
#nullable enable

using System;

string BuildFullName(Person p)
{
    if (p is null)
    {
        throw new ArgumentNullException(nameof(p));
    }

    if (p.FirstName.Length == 0) {
        return p.FirstName;
    }

    return $"{p.LastName}, {p.FirstName}";
}

var p = new Person("Foo", "Bar");
Console.WriteLine(BuildFullName(p));

// Pass null and supress nullable warning (shouldn't do that,
// but sh... sometimes happen).
Console.WriteLine(BuildFullName(null!));

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

Null Checking

```csharp
#nullable enable

using System;

string BuildFullNameWithCheck(Person p!!)
{
    // We compile with #nullable enable, so we rely on
    // p and p.FirstName to not be null.
    if (p.FirstName.Length == 0) {
        return p.FirstName;
    }

    return $"{p.LastName}, {p.FirstName}";
}

var p = new Person("Foo", "Bar");
Console.WriteLine(BuildFullName(p));

// Pass null and supress nullable warning (shouldn't do that,
// but sh... sometimes happen).
Console.WriteLine(BuildFullName(null!));

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

Inlining

Note: C# compiler will inline short methods and skip null check entirely if it can be proven that parameter is never null (e.g. non-null literal has been passed)

Raw String Literals

""""""""""""""""""""😵💫""""""""""""""""""""

Traditional String

```csharp
using static System.Console;

// This is how Hello World can look like in C#

{
    var s = "System.Console.WriteLine(\n\t\"Hello World!\"\n);";
    WriteLine(s);
}
```

https://sharplab.io/#gist:dad5b01aeaade008d84e3f27495aedff

String Interpolation

```
using static System.Console;

{
    var greet = "Hello World!";
    var s = $"System.Console.WriteLine(\n\t\"{greet}\"\n);";
    WriteLine(s);
}
```

Verbatim String Literals

```
using static System.Console;

{
    var s = @"System.Console.WriteLine(
        ""Hello World!""
);";
    WriteLine(s);
}
```

verbatim = "wortwörtlich" in German

Verbatim String Interpolation

```csharp
using static System.Console;

{
    var greet = "Hello World!";
    var s = @$"System.Console.WriteLine(
        ""{greet}""
);";
    WriteLine(s);
}
```

Escaping

```csharp
using static System.Console;

{
    var s = "using static System.Console;\nnamespace Demo\n{\n\tpublic class Program\n\t{\n\t\tpublic void Main()\n\t\t{\n\t\t\tWriteLine(\"Hello World!\");\n\t\t}\n\t}\n}";
    WriteLine(s);
}
```

Multi-line Verbatim String

```
using static System.Console;

{
    var s = @"using static System.Console;
namespace Demo
{
    public class Program
    {
        public void Main()
        {
            WriteLine(""Hello World!"");
        }
    }
}";
    WriteLine(s);
}
```

Raw String Literal

```
using static System.Console;

{
    // Note: First and last newline are ignored
    // Be careful when mixing tabs and spaces. Shouldn't to that.
    var s = """
            using static System.Console;
            namespace Demo
            {
                public class Program
                {
                    public void Main()
                    {
                        WriteLine("Hello World!");
                    }
                }
            }
            """;
    WriteLine(s);
}
```

Want more " 🤪

```
using static System.Console;

{
    WriteLine(""""
              The new Raw String Literal feature is great:
              System.Console.WriteLine("""
                                       Hello!
                                       """);
              """");
}
```

Raw String Literal Interpolation

```csharp
using static System.Console;

{
    var greet = "Hello!";
    WriteLine($""""
                The new Raw String Literal feature is great:
                System.Console.WriteLine("""
                                         {greet}
                                         """);
                """");
}
```

Generic Attributes

Generic Attributes

```
#nullable enable

using System;
using System.Linq;

[AttributeUsage(System.AttributeTargets.Class)]
class MyVersionAttribute<T> : Attribute
{
    public T Version { get; }

    public MyVersionAttribute(T version)
    {
        Version = version;
    }
}

[MyVersion<int>(42)]
class A { }

[MyVersion<string>("4.2")]
class B { }
```

https://sharplab.io/#gist:e92524c61495a3fa5aabc86d89cad018

Generic Attributes

```
#nullable enable

using System;
using System.Linq;

MyVersionAttribute<T>? GetVersion<T>(Type t)
{
    return t.GetCustomAttributes(false)
        .OfType<MyVersionAttribute<T>>()
        .FirstOrDefault();
}

Console.WriteLine($"The version is {GetVersion<int>(typeof(A)).Version}");
Console.WriteLine($"The version is {GetVersion<string>(typeof(B)).Version}");
Console.WriteLine($"The version is {GetVersion<int>(typeof(B))?.Version ?? -1}");
```

Deconstructing default

Sounds philosophical, doesn't it? 😉

Deconstructing default

```
using System;

ErrorKind err;

// Note mixint variable declaration and assignment in a deconstruction (C# 10)
(int result, err) = DoSomethingThatMightFail(42);
(result, err) = DoSomethingThatMightFail(41);
if (err != ErrorKind.NoError)
{
    Console.WriteLine(err);
}

(int result, ErrorKind err) DoSomethingThatMightFail(int parameter)
{
    // Note use of default in the implementation
    if (parameter >= 42) { return (42, default); };
    return (default, ErrorKind.NotImplemented);
}

enum ErrorKind
{
    NoError,
    GeneralError,
    NotImplemented,
    InvalidState,
}
```

https://sharplab.io/#gist:75c4ff0b34b9550d6f99ff694addda50

Deconstructing default

```csharp
using System;

ErrorKind err;

// Note mixint variable declaration and assignment in a deconstruction (C# 10)
(int result, err) = DoSomethingThatMightFail(42);
(result, err) = DoSomethingThatMightFail(41);
if (err != ErrorKind.NoError)
{
    Console.WriteLine(err);
}

// Note deconstruction of default (C# vNext)
(int result2, ErrorKind err2) = default;
(result2, err2) = DoSomethingThatMightFail(41);
if (err != ErrorKind.NoError)
{
    Console.WriteLine(err);
}
```

Semi-Auto Properties

Semi-auto Properties

```csharp
using System;
using System.ComponentModel;

class Person : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;
    public string FullName => $"{LastName}, {FirstName}";
    public string FirstName
    {
        get => field;
        set
        {
            if (field != value)
            {
                field = value;
                PropertyChanged?.Invoke(this, new(nameof(FirstName)));
                PropertyChanged?.Invoke(this, new(nameof(FullName)));
            }
        }
    }

    public string LastName
    {
        // ... (like FirstName above)
    }
}
```

https://sharplab.io/#gist:1f62e3187fa76fc38a722a731de1e43b

Semi-auto Properties

```csharp
using System;
using System.ComponentModel;

var p = new Person { FirstName = "Foo", LastName = "Bar" };
p.PropertyChanged += (s, ea) => Console.WriteLine($"{ea.PropertyName} changed");
p.LastName = "Baz";

class Person : INotifyPropertyChanged
{
    // ...
    public string FirstName
    {
        get => field;
        set
        {
            if (field != value)
            {
                field = value;
                PropertyChanged?.Invoke(this, new(nameof(FirstName)));
                PropertyChanged?.Invoke(this, new(nameof(FullName)));
            }
        }
    }
    // ...
}
```

Some proposals in early stages

Required Members

```csharp
using System;

// The following statement will not be ok as required members are missing.
var p = new Person();

// The following statement is ok as all required members are set.
var p2 = new Person()
{
    FirstName = "Foo",
    LastName = "Bar",
    Age = 42,
};

namespace System.Runtime.CompilerServices
{
    public class RequiredMemberAttribute :Attribute { }
}

class Person {
    public required string FirstName { get; init; }
    public string MiddleName { get; init; }
    public required string LastName { get; init; }
    public required int Age { get; init; }
}
```

https://sharplab.io/#gist:81ba66820395d433818ee41c1904088b

UTF8 String Literals

```
// C# will allow conversions between string constants and byte sequences
// where the text is converted into the equivalent UTF8 byte representation.

byte[] array = "hello";                // new byte[] { 0x68, 0x65, 0x6c, 0x6c, 0x6f }
Span<byte> span = "dog";               // new byte[] { 0x64, 0x6f, 0x67 }
ReadOnlySpan<byte> span = "cat";       // new byte[] { 0x63, 0x61, 0x74 }

// Will work with string constants, too.
const string data = "dog"
ReadOnlySpan<byte> span = data;        // new byte[] { 0x64, 0x6f, 0x67 }

// New u8 suffix.
var s2 = "hello"u8;                    // Okay and type is byte[]
```

https://github.com/dotnet/csharplang/blob/main/proposals/utf8-string-literals.md

Interested in more?

https://github.com/dotnet/roslyn/blob/main/docs/Language%20Feature%20Status.md

C# 11 🤘

Rainer Stropek | @rstropek