The State of C#

What Have I Missed?

Filip Ekberg

Filip Ekberg

Microsoft MVP

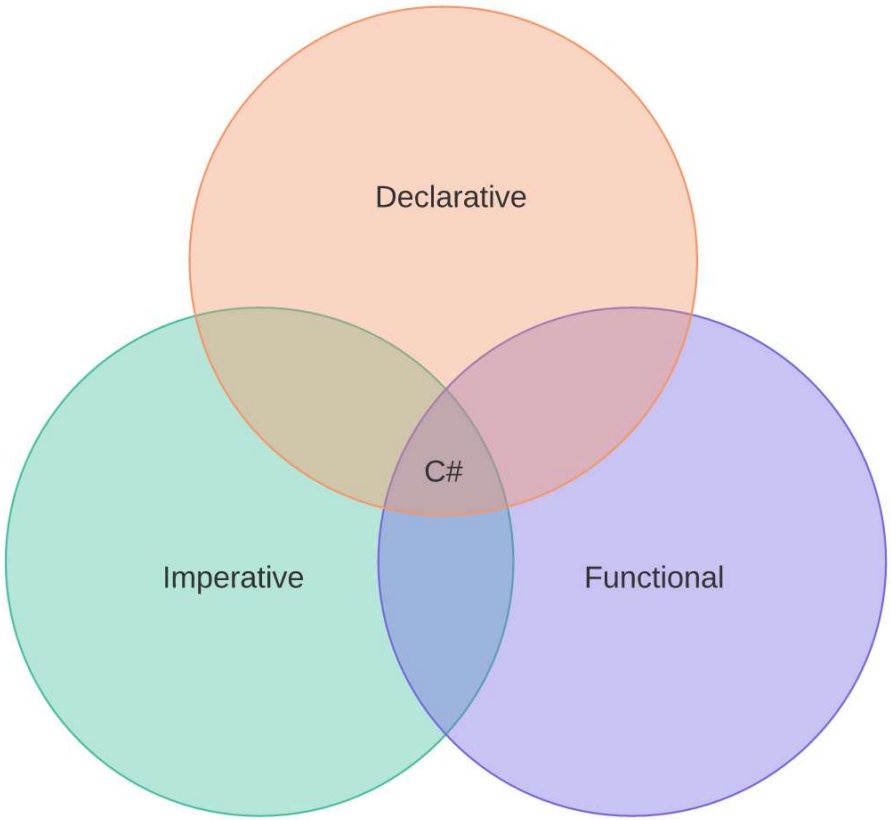
@fekberg

C# smorgasbord free @ filipekberg.se

pluralsight

fekberg

The Best of All Worlds – C#



C# 2.0

Generics

Partial types

Anonymous methods

Iterators

Nullable types

Getter/Setter separate accessibility

Static classes

And more..

C# 3.0

Implicitly typed local variables

Object and collection initializers

Auto-properties

Anonymous types

Extension methods

Query expressions

Lambda expressions

Expression trees

And more..

C# 4.0

Dynamic binding

Named and optional parameters

Generic co- and contravariance

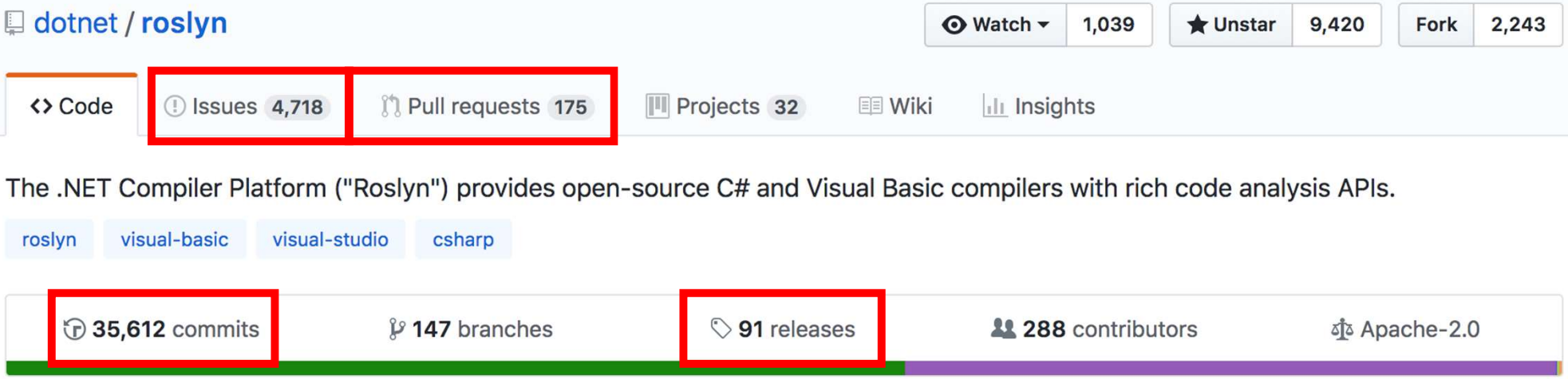
And more..

C# 5.0

Asynchronous methods

Caller info attributes


roslyn


dotnet / roslyn
Watch 1,039
Unstar 9,420
Fork 2,243
Code
Issues 4,718
Pull requests 175
Projects 32
Wiki
Insights
The .NET Compiler Platform ("Roslyn") provides open-source C# and Visual Basic compilers with rich code analysis APIs.
roslyn
visual-basic
visual-studio
csharp
35,612 commits
147 branches
91 releases
288 contributors
Apache-2.0

C# Design Notes for Jul12, 2016 Area-Language Design Design Notes Language-C# Language-VB 29

New Language Feature - Tuples

#13022 opened on Aug 9 by MadsTorgersen

C# Design Notes for Jul 13, 2016 Area-Language Design Design Notes Language-C# Language-VB 57

New Language Feature - Pattern Matching New Language Feature - Tuples

#13015 opened on Aug 9 by MadsTorgersen

Proposal: Language support for Tuples #347

Open **MadsTorgersen** opened this issue on Feb 10 2015 · 512 comments

MadsTorgersen commented on Feb 10 2015

There are many scenarios where you'd like to group a set of typed values temporarily, without the grouping itself warranting a "concept" or type name of its own.

Other languages use variations over the notion of **tuples** for this. Maybe C# should too.

This proposal follows up on #98 and addresses #102 and #307.

Background

The most common situation where values need to be temporarily grouped, a list of arguments to (e.g.) a method, has syntactic support in C#. However, the probably *second*-most common, a list of *results*, does not.

While there are many situations where tuple support could be useful, the most prevalent by far is the ability to return multiple values from an operation.

Your options today include:

Out parameters:

```
public void Tally(IEnumerable<int> values, out int sum, out int count) { ... }

int s, c;
Tally(myValues, out s, out c);
Console.WriteLine($"Sum: {s}, count: {c}");
```


No more
Language Parity

C# 6.0

Using Statements for Static Members

Auto-Property Initializers

Await inside Catch & Finally blocks

Null-conditional operators

String interpolation

Dictionary Initializers

Expression-bodied members

Exception Filters

nameof operator

C# 7.0

Tuples and Deconstruction

Pattern Matching

Digit Separator & Binary Literals

Local Functions

ref returns

out improvements

C# 7.1

Async Main

Default Expressions

Infer Tuple Names

Pattern-matching with Generics

Ref Assemblies

C# 7.2

ref readonly & in parameter

non-trailing named arguments

stackalloc with Span<T>

private protected

```
CLR protectedAndInternal = private protected
```

conditional ref operator

```
<condition> ? ref <consequence> : ref <alternative>
```

digit separator after base specifier

```
0x_1_2
```

C# 7.3

Tuple equality

stackalloc initializers

Improved generic constraints

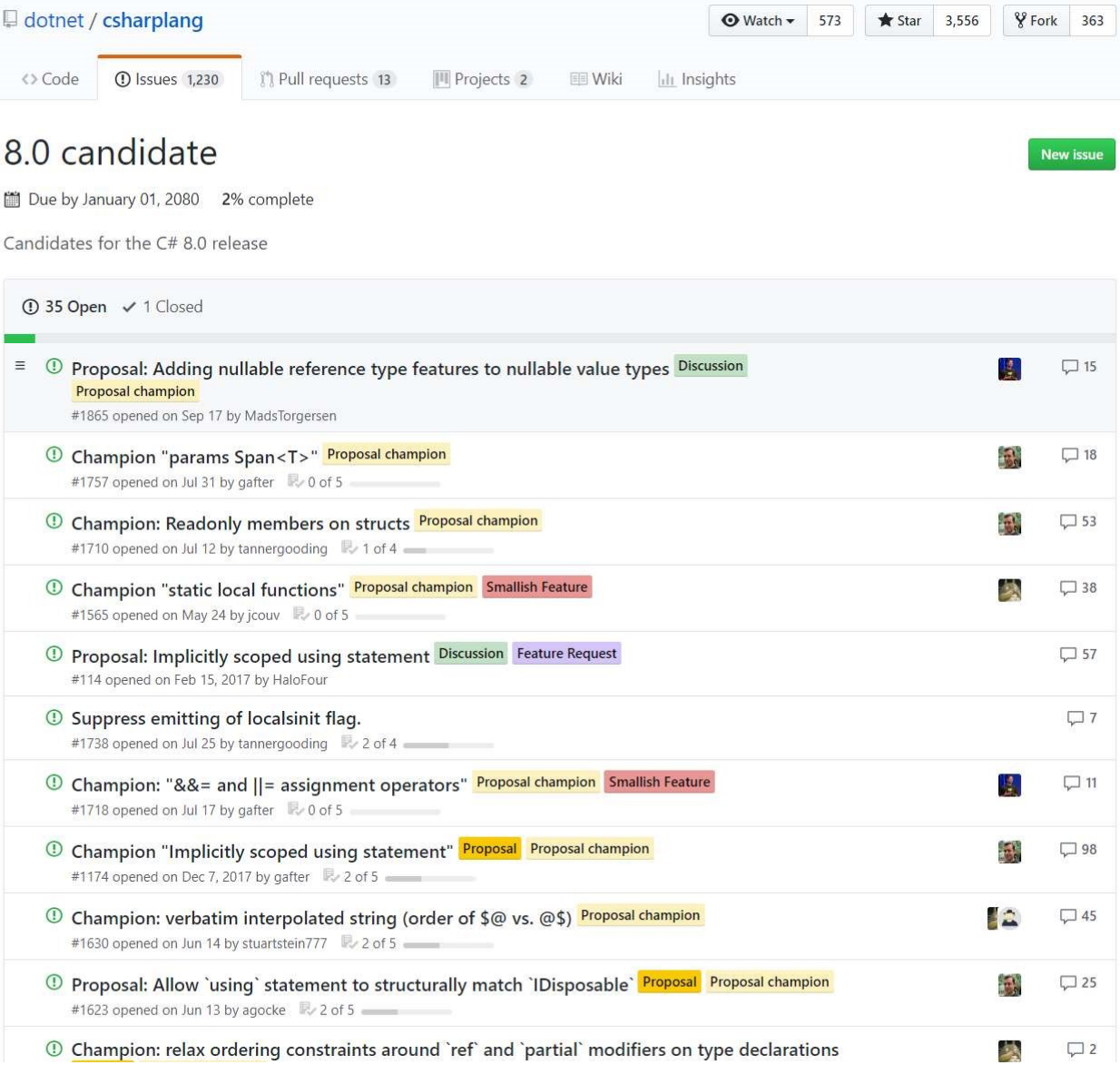

What's
NEXT

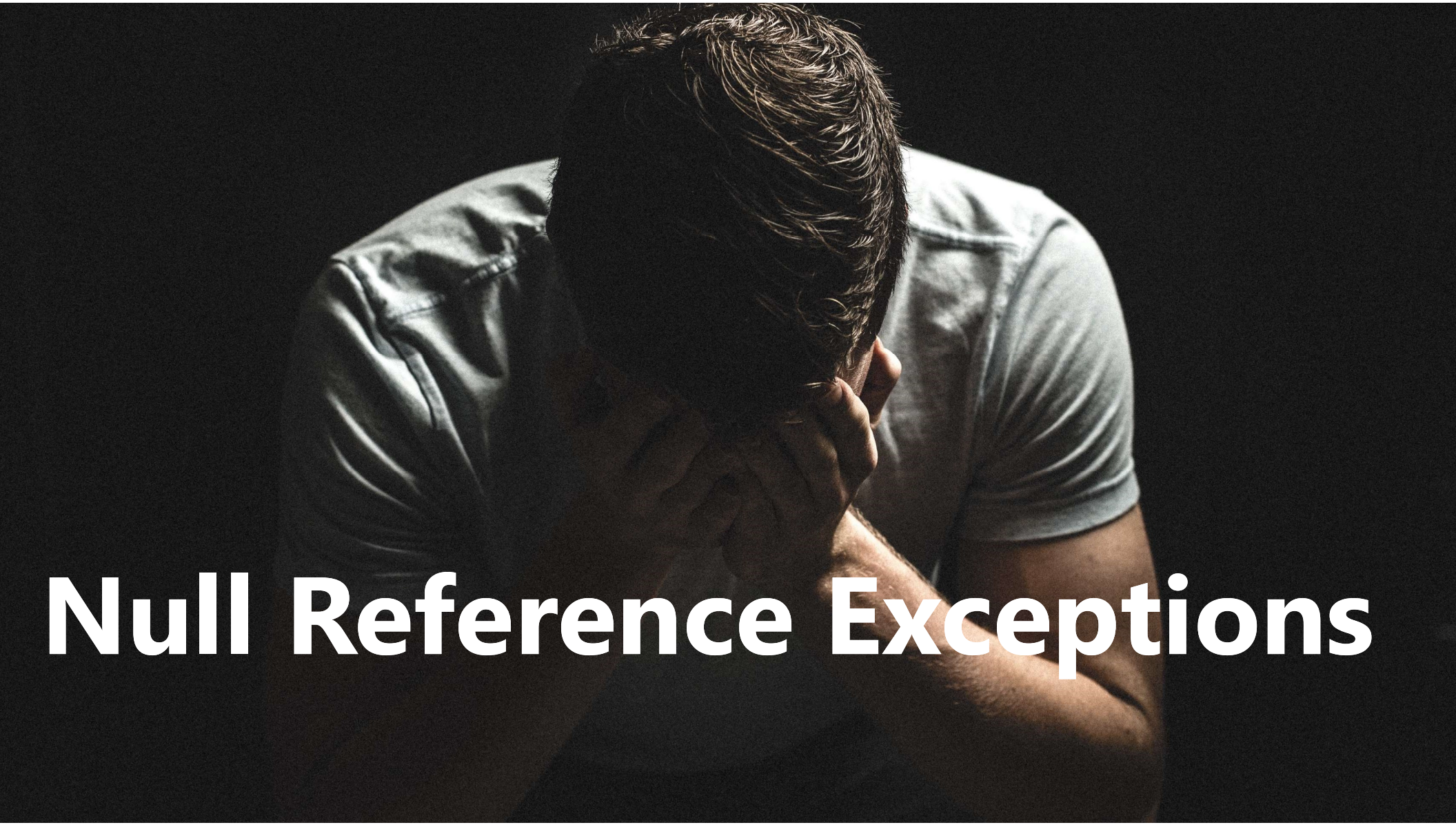
Language Feature Status

This document reflects the status, and planned work in progress, for the compiler team. It is a live document and will be updated as work progresses, features are added / removed, and as work on feature progresses. This is not an exhaustive list of our features but rather the ones which have active development efforts behind them.

C# 8.0

Feature	Branch	State	Developers	Reviewer	LDM Champ
Default Interface Methods	defaultInterfaceImplementation	Prototype	AlekseyTs	gafter	gafter


dotnet / csharplang
Watch 573
Star 3,556
Fork 363
Code
Issues 1,230
Pull requests 13
Projects 2
Wiki
Insights
8.0 candidate
New issue
Due by January 01, 2080
2% complete
Candidates for the C# 8.0 release
35 Open
1 Closed
Proposal: Adding nullable reference type features to nullable value types
Discussion
Proposal champion
#1865 opened on Sep 17 by MadsTorgersen
15
Champion "params Span<T>"
Proposal champion
#1757 opened on Jul 31 by gafter
0 of 5
18
Champion: Readonly members on structs
Proposal champion
#1710 opened on Jul 12 by tannergooding
1 of 4
53
Champion "static local functions"
Proposal champion
Smallish Feature
#1565 opened on May 24 by jcouv
0 of 5
38
Proposal: Implicitly scoped using statement
Discussion
Feature Request
#114 opened on Feb 15, 2017 by HaloFour
57
Suppress emitting of localsinit flag.
#1738 opened on Jul 25 by tannergooding
2 of 4
7
Champion: "&&= and ||= assignment operators"
Proposal champion
Smallish Feature
#1718 opened on Jul 17 by gafter
0 of 5
11
Champion "Implicitly scoped using statement"
Proposal
Proposal champion
#1174 opened on Dec 7, 2017 by gafter
2 of 5
98
Champion: verbatim interpolated string (order of $@ vs. @$)
Proposal champion
#1630 opened on Jun 14 by stuartstein777
2 of 5
45
Proposal: Allow `using` statement to structurally match `IDisposable`
Proposal
Proposal champion
#1623 opened on Jun 13 by agocke
2 of 5
25
Champion: relax ordering constraints around `ref` and `partial` modifiers on type declarations
2


Null Reference Exceptions


C# X

Ranges

Async Streams & Enumerables

Record Types

Target-type new expression

default in deconstruction

Generic attributes

Caller expression attribute

Enhanced using (pattern-based using)

Default interface methods

Nullable reference type

Null coalescing assignment

Switch expressions

Declaration expressions

Dictionary literals

Type classes

&&= and ||= assignment operators

static local functions

params Span<T>

readonly struct members

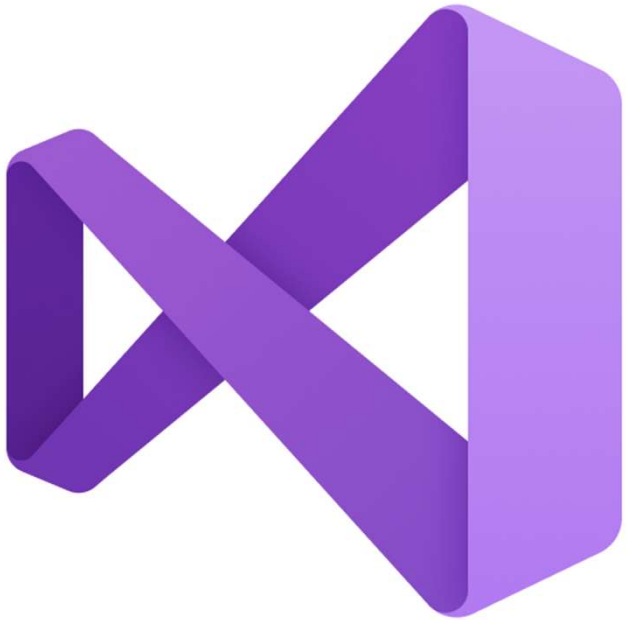
Negated-condition if statement

Null-conditional await

And More!

8.0 candidate

Due by January 01, 2080 **2%** complete




C# 8.0

Async streams

```
var enumerable = GetAsynchronousEnumerable();
await foreach (T item in enumerable)
{

}
```

https://github.com/dotnet/csharplang/blob/master/proposals/async-streams.md

Async streams

```
var enumerator = enumerable.GetAsyncEnumerator();
try
{
    while (await enumerator.MoveNextAsync())
    {
        T item = enumerator.Current;
        ...
    }
}
finally
{
    await enumerator.DisposeAsync();
}
```

https://github.com/dotnet/csharplang/blob/master/proposals/async-streams.md

Default interface methods

```
interface IAuthentication
{
    // Version 1
    User Authenticate(Credentials credentials);

    // Version 2
    Task<User> AuthenticateAsync(Credentials credentials) =>
                                                Task.FromResult<User>(default)
}

class Authenticate : IAuthentication
{
    public User Authenticate(Credentials credentials) => ...
}

IAuthentication auth = new Authenticate();

var user = await auth.AuthenticateAsync();
```

https://github.com/dotnet/csharplang/issues/52

Record Types

```
public class Triangle : Shape
{
    public int A { get; }
    public int B { get; }
    public int C { get; }

    public Triangle(int a, int b, int c)
    {
        A = a;
        B = b;
        C = c;
    }
}
```

```
public class Triangle(int A, int B, int C ) : Shape {}
```

https://github.com/dotnet/csharplang/issues/39

Record Types

```
public class Triangle : Shape, IEquatable<Triangle>
{
    public int A { get; }
    public int B { get; }
    public int C { get; }

    public Triangle(int a, int b, int c)
    {
        A = a; B = b; C = c;
    }

    public bool Equals(Triangle other)
        => other != null && Equals(A, other.A) && Equals(B, other.B) && Equals(C, other.C);

    public override bool Equals(object other) => this.Equals(other as Triangle);

    public override int GetHashCode() =>  A.GetHashCode() * 17 + B.GetHashCode() + C.GetHashCode();

    public Triangle With(int A = this.A, int B = this.B, int C = this.C) => new Triangle(A, B, C);

    public void Deconstruct(out int A, out int B, out int C)  { A = this.A; B = this.B; C = this.C; }
}
```

Negated-condition if statement

```
if( !(shape is Triangle) ) {};
```

```
if(shape is not Triangle) {};

if!(shape is Triangle) {};

unless(shape is Triangle) {};
```

https://github.com/dotnet/csharplang/issues/882

Declaration expressions

```
char ch;
while ((ch = GetNextChar()) == 'a'
        || ch == 'b'
        || ch == 'c')
{ }
```

```
while ((char ch = GetNextChar()) == 'a'
            || ch == 'b'
            || ch == 'c')
{ }
```

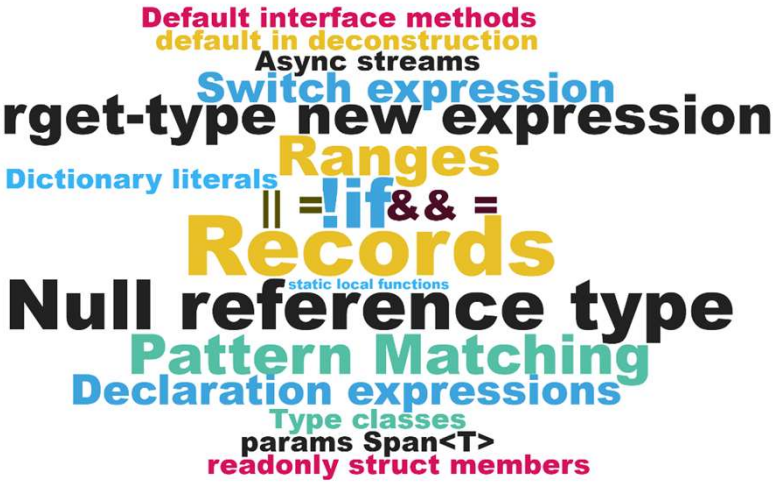
https://github.com/dotnet/csharplang/issues/973

C# 1 => 7.3


Using Statements for Static Members
Implicitly typed local variables
Extension methods
Getter & Setter separate accessibility
Null-conditional operators
Exception Filters
Anonymous methods
Named and optional parameters
Lambdas
Static classes
Auto-properties
String interpolation
Nullable types
Digit Separators
Stackalloc with Span
Dynamic binding
non-trailing named arguments
Default Expressions
digit separators
Generics
Expression trees
Dictionary Initializers
Local Functions
Tuples and Deconstruction
Ref Assemblies
Async & Await
Infer Tuple Names
ref returns
Query expressions
Iterators
nameof operator
Generic co- and contravariance
Partial types
Pattern Matching
Object and collection initializers
conditional ref operator
Async Main
Tuple equality
Binary Literals
Expression-bodied members
Auto-Property Initializers
Anonymous types
ref readonly & in parameter

+

C# 8 => n


Default interface methods
default in deconstruction
Async streams
Switch expression
Target-type new expression
Ranges
Dictionary literals
|| =!if&& =
Records
static local functions
Null reference type
Pattern Matching
Declaration expressions
Type classes
params Span<T>
readonly struct members

Wrap Up

Thanks!

I'm Filip Ekberg

@fekberg

C# smorgasbord free @ filipekberg.se

pluralsight

fekberg