Breaking Changes .NET 6 / 7

[Breaking changes in the C# compiler | Microsoft Docs](https://docs.microsoft.com/nl-nl/dotnet/csharp/whats-new/breaking-changes)

## This document lists known breaking changes in Roslyn after .NET 6 all the way to .NET 7.

1. In Visual Studio 17.1, the contextual keyword var cannot be used as an explicit lambda return type.
2. using System;
3. F(var () => default); // error: 'var' cannot be used as an explicit lambda return type
4. F(@var () => default); // ok
5. static void F(Func<var> f) { }

class var { }

1. In Visual Studio 17.1, indexers that take an interpolated string handler and require the receiver as an input for the constructor cannot be used in an object initializer.
2. using System.Runtime.CompilerServices;
3. \_ = new C { [$""] = 1 }; // error: Interpolated string handler conversions that reference the instance being indexed cannot be used in indexer member initializers.
4. class C
5. {
6. public int this[[InterpolatedStringHandlerArgument("")] CustomHandler c]
7. {
8. get => throw null;
9. set => throw null;
10. }
11. }
12. [InterpolatedStringHandler]
13. class CustomHandler
14. {
15. public CustomHandler(int literalLength, int formattedCount, C c) {}

}

1. In Visual Studio 17.1, ref/ref readonly/in/out are not allowed to be used on return/parameters of a method attributed with UnmanagedCallersOnly.  
   <https://github.com/dotnet/roslyn/issues/57025>
2. using System.Runtime.InteropServices;
3. [UnmanagedCallersOnly]
4. static ref int M1() => throw null; // error CS8977: Cannot use 'ref', 'in', or 'out' in a method attributed with 'UnmanagedCallersOnly'.
5. [UnmanagedCallersOnly]
6. static ref readonly int M2() => throw null; // error CS8977: Cannot use 'ref', 'in', or 'out' in a method attributed with 'UnmanagedCallersOnly'.
7. [UnmanagedCallersOnly]
8. static void M3(ref int o) => throw null; // error CS8977: Cannot use 'ref', 'in', or 'out' in a method attributed with 'UnmanagedCallersOnly'.
9. [UnmanagedCallersOnly]
10. static void M4(in int o) => throw null; // error CS8977: Cannot use 'ref', 'in', or 'out' in a method attributed with 'UnmanagedCallersOnly'.
11. [UnmanagedCallersOnly]

static void M5(out int o) => throw null; // error CS8977: Cannot use 'ref', 'in', or 'out' in a method attributed with 'UnmanagedCallersOnly'.

1. Beginning with C# 11.0, Length and Count properties on countable and indexable types are assumed to be non-negative for purpose of subsumption and exhaustiveness analysis of patterns and switches. Those types can be used with implicit Index indexer and list patterns.
2. void M(int[] i)
3. {
4. if (i is { Length: -1 }) {} // error: impossible under assumption of non-negative length

}

1. Starting with Visual Studio 17.1, format specifiers in interpolated strings can not contain curly braces (either { or }). In previous versions {{ was interpreted as an escaped { and }} was interpreted as an escaped } char in the format specifier. Now the first } char in a format specifier ends the interpolation, and any { char is an error. <https://github.com/dotnet/roslyn/issues/5775>
2. using System;
3. Console.WriteLine($"{{{12:X}}}");

//prints now: "{C}" - not "{X}}"

## This document lists known breaking changes in Roslyn in C# 10.0 which will be introduced with .NET 6.

1. Beginning with C# 10.0, null suppression operator is no longer allowed in patterns.
2. void M(object o)
3. {
4. if (o is null!) {} // error

}

1. In C# 10, lambda expressions and method groups with inferred type are implicitly convertible to System.MulticastDelegate, and bases classes and interfaces of System.MulticastDelegate including object, and lambda expressions and method groups are implicitly convertible to System.Linq.Expressions.Expression and System.Linq.Expressions.LambdaExpression. These are function\_type\_conversions.

The new implicit conversions may change overload resolution in cases where the compiler searches iteratively for overloads and stops at the first type or namespace scope containing any applicable overloads.

a. Instance and extension methods

class C

{

static void Main()

{

var c = new C();

c.M(Main); // C#9: E.M(); C#10: C.M()

c.M(() => { }); // C#9: E.M(); C#10: C.M()

}

void M(System.Delegate d) { }

}

static class E

{

public static void M(this object x, System.Action y) { }

}

b. Base and derived methods

using System;

using System.Linq.Expressions;

class A

{

public void M(Func<int> f) { }

public object this[Func<int> f] => null;

public static A operator+(A a, Func<int> f) => a;

}

class B : A

{

public void M(Expression e) { }

public object this[Delegate d] => null;

public static B operator+(B b, Delegate d) => b;

}

class Program

{

static int F() => 1;

static void Main()

{

var b = new B();

b.M(() => 1); // C#9: A.M(); C#10: B.M()

\_ = b[() => 2]; // C#9: A.this[]; C#10: B.this[]

\_ = b + F; // C#9: A.operator+(); C#10: B.operator+()

}

}

c. Method group or anonymous method conversion to Expression or LambdaExpression

using System;

using System.Linq.Expressions;

var c = new C();

c.M(F); // error CS0428: Cannot convert method group 'F' to non-delegate type 'Expression'

c.M(delegate () { return 1; }); // error CS1946: An anonymous method expression cannot be converted to an expression tree

static int F() => 0;

class C

{

public void M(Expression e) { }

}

static class E

{

public static void M(this object o, Func<int> a) { }

}

1. In C#10, a lambda expression with inferred type may contribute an argument type that affects overload resolution.
2. using System;
3. class Program
4. {
5. static void F(Func<Func<object>> f, int i) { }
6. static void F(Func<Func<int>> f, object o) { }
7. static void Main()
8. {
9. F(() => () => 1, 2); // C#9: F(Func<Func<object>>, int); C#10: ambiguous
10. }

}

1. In Visual Studio 17.1, struct type declarations with field initializers must include an explicitly declared constructor. Additionally, all fields must be definitely assigned in struct instance constructors that do not have a : this() initializer so any previously unassigned fields must be assigned from the added constructor or from field initializers.

For instance, the following results in an error in 17.1:

struct S

{

int X = 1; // error: struct with field initializers must include an explicitly declared constructor

int Y;

}

The error could be resolved by adding a constructor and assigning the other field.

struct S

{

int X = 1;

int Y;

public S() { Y = 0; } // ok

}

Verwijzingen

(2021). *Learn about any breaking changes in the C# compiler.*  microsoft.com. https://docs.microsoft.com/nl-nl/dotnet/csharp/whats-new/breaking-changes

roslyn/Compiler Breaking Changes - DotNet 6.md at main · dotnet/roslyn

roslyn/Compiler Breaking Changes - DotNet 7.md at main · dotnet/roslyn