This document lists known breaking changes in Roslyn after .NET 7 all the way to .NET 8.

Article • 04/02/2024

Ref modifiers of dynamic arguments should be compatible with ref modifiers of corresponding parameters

Introduced in Visual Studio 2022 version 17.10

Ref modifiers of dynamic arguments should be compatible with ref modifiers of corresponding parameters at compile time. This can cause an overload resolution involving dynamic arguments to fail at compile time instead of runtime.

Previously, a mismatch was allowed at compile time, delaying the overload resolution failure to runtime.

For example, the following code used to compile without an error, but was failing with exception: "Microsoft.CSharp.RuntimeBinder.RuntimeBinderException: The best overloaded method match for 'C.f(ref object)' has some invalid arguments" It is going to produce a compilation error now.

C#

```
public class C
{
    public void f(ref dynamic a)
    {
    }

    public void M(dynamic d)
    {
        f(d); // error CS1620: Argument 1 must be passed with the
'ref' keyword
    }
}
```

Collection expression target type must have constructor and `Add` method

Introduced in Visual Studio 2022 version 17.10

Conversion of a collection expression to a `struct` or `class` that implements `System.Collections.IEnumerable` and *does not* have a `CollectionBuilderAttribute` requires the target type to have an accessible constructor that can be called with no arguments and, if the collection expression is not empty, the target type must have an accessible `Add` method that can be called with a single argument.

Previously, the constructor and `Add` methods were required for *construction* of the collection instance but not for *conversion*. That meant the following call was ambiguous since both `char[]` and `string` were valid target types for the collection expression. The call is no longer ambiguous because `string` does not have a parameterless constructor or `Add` method.

C#

```
Print(['a', 'b', 'c']); // calls Print(char[])

static void Print(char[] arg) { }
static void Print(string arg) { }
```

`ref` arguments can be passed to `in` parameters

Introduced in Visual Studio 2022 version 17.8p2

Feature ref readonly parameters relaxed overload resolution allowing `ref` arguments to be passed to `in` parameters when `LangVersion` is set to 12 or later. This can lead to behavior or source breaking changes:

cs

```
var i = 5;
System.Console.Write(new C().M(ref i)); // prints "E" in C# 11, but
"C" in C# 12
System.Console.Write(E.M(new C(), ref i)); // workaround: prints "E"
always

class C
{
    public string M(in int i) => "C";
}
static class E
{
    public static string M(this C c, ref int i) => "E";
}
```

cs

```cs
var i = 5;
System.Console.Write(C.M(null, ref i)); // prints "1" in C# 11, but
fails with an ambiguity error in C# 12
System.Console.Write(C.M((I1)null, ref i)); // workaround: prints "1"
always

interface I1 { }
interface I2 { }
static class C
{
    public static string M(I1 o, ref int x) => "1";
    public static string M(I2 o, in int x) => "2";
}
```

Prefer pattern-based over interface-based disposal in async `using`

Introduced in Visual Studio 2022 version 17.10p3

An async `using` prefers to bind using a pattern-based `DisposeAsync()` method rather than the interface-based `IAsyncDisposable.DisposeAsync()`.

For instance, the public `DisposeAsync()` method will be picked, rather than the private interface implementation:

C#

```csharp
await using (var x = new C()) { }

public class C : System.IAsyncDisposable
{
    ValueTask IAsyncDisposable.DisposeAsync() => throw null; // no
longer picked

    public async ValueTask DisposeAsync()
    {
        Console.WriteLine("PICKED");
        await Task.Yield();
        }
}
```