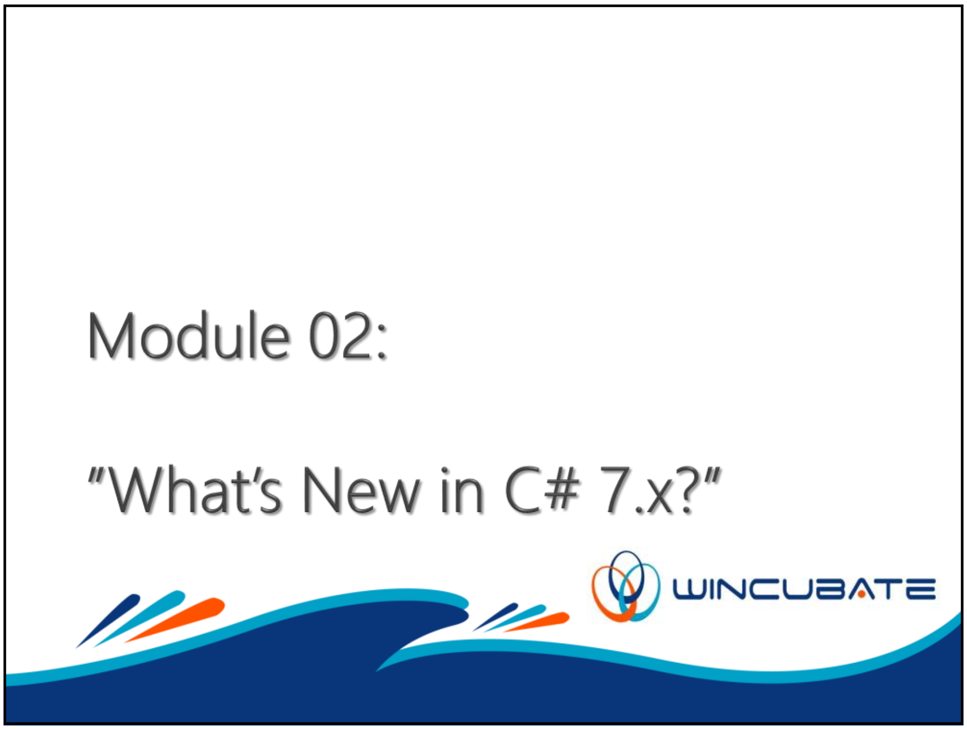
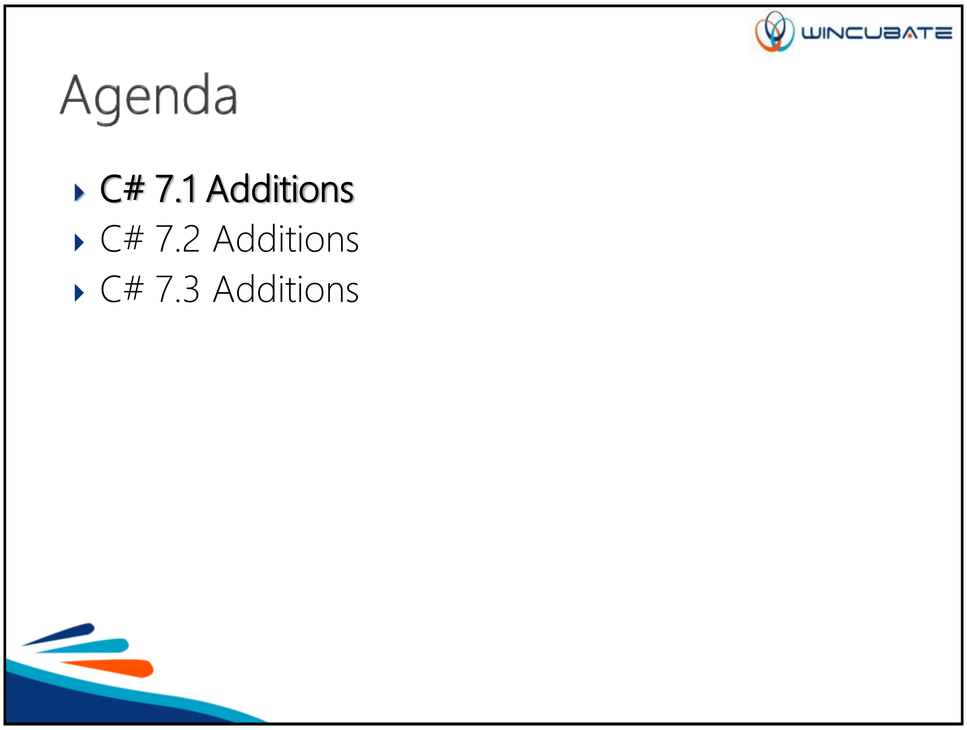


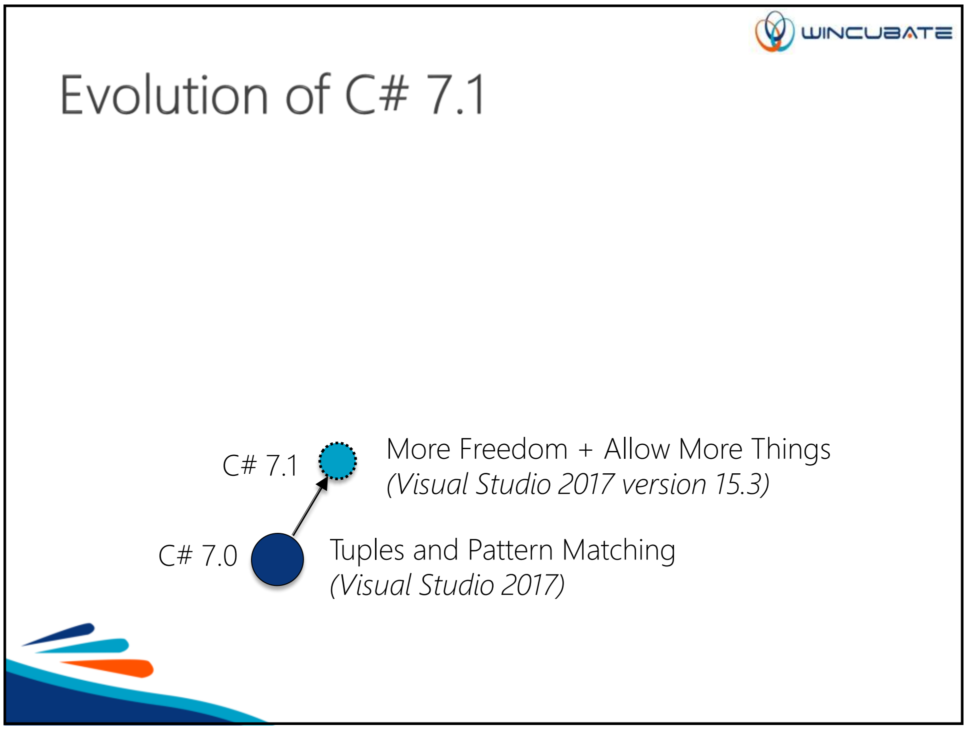
Module 02:

"What's New in C# 7.x?"



Agenda

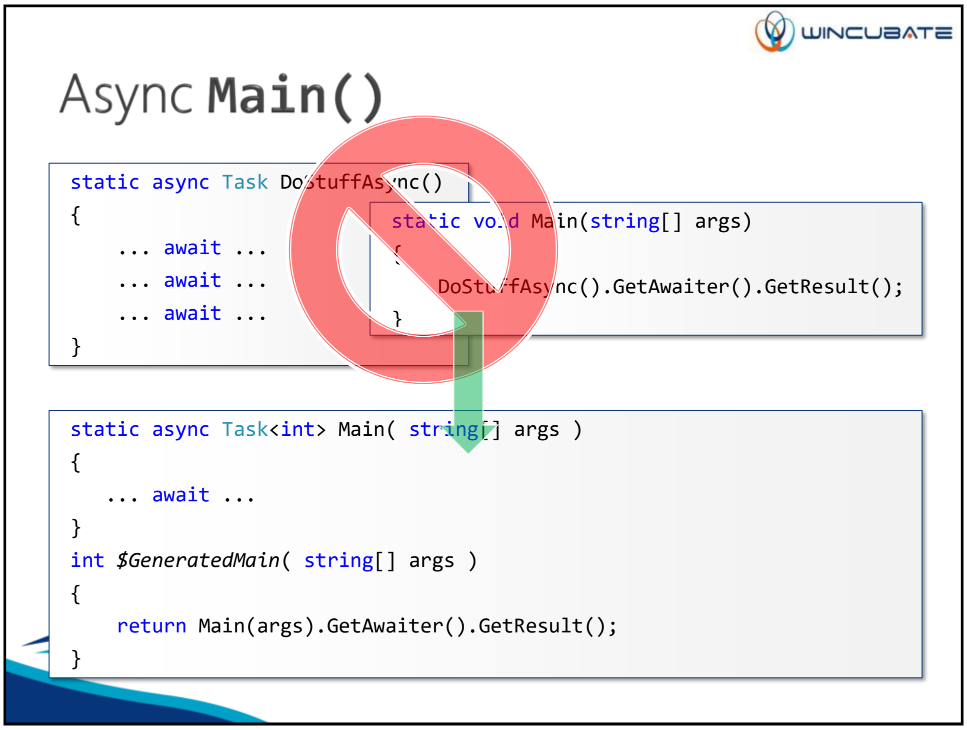
- **C# 7.1 Additions**
- C# 7.2 Additions
- C# 7.3 Additions



Evolution of C# 7.1

More Freedom + Allow More Things
(Visual Studio 2017 version 15.3)

Tuples and Pattern Matching
(Visual Studio 2017)



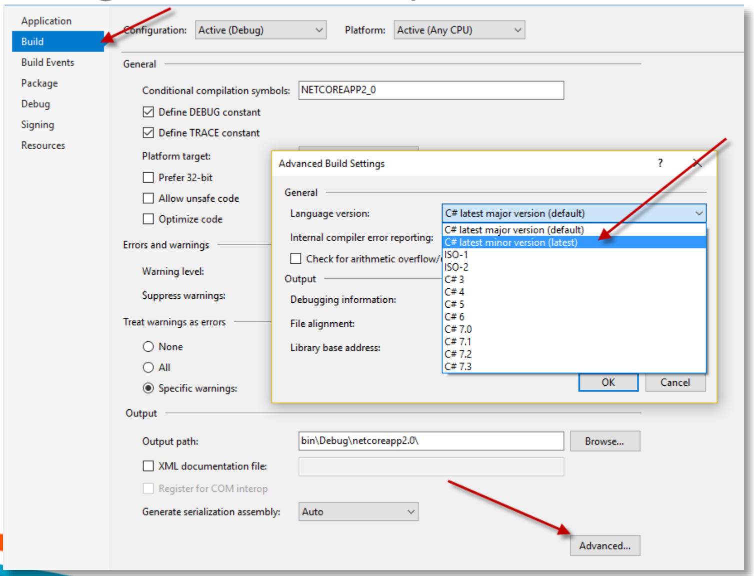
Async Main()

```
static async Task DoStuffAsync()
{
    ... await ...
    ... await ...
    ... await ...
}
```

```
static void Main(string[] args)
{
    DoStuffAsync().GetAwaiter().GetResult();
}
```

```
static async Task<int> Main( string[] args )
{
   ... await ...
}
int $GeneratedMain( string[] args )
{
    return Main(args).GetAwaiter().GetResult();
}
```

Enabling C# 7.x Compilation


Application
Build
Build Events
Package
Debug
Signing
Resources

Configuration: Active (Debug) Platform: Active (Any CPU)

General
Conditional compilation symbols: NETCOREAPP2_0
☑ Define DEBUG constant
☑ Define TRACE constant
Platform target:
☐ Prefer 32-bit
☐ Allow unsafe code
☐ Optimize code
Errors and warnings
Warning level:
Suppress warnings:
Treat warnings as errors
○ None
○ All
● Specific warnings:
Output
Output path: bin\Debug\netcoreapp2.0\
☐ XML documentation file:
☐ Register for COM interop
Generate serialization assembly: Auto

Advanced Build Settings
General
Language version: C# latest major version (default)
- C# latest major version (default)
- C# latest minor version (latest)
- ISO-1
- ISO-2
- C# 3
- C# 4
- C# 5
- C# 6
- C# 7.0
- C# 7.1
- C# 7.2
- C# 7.3

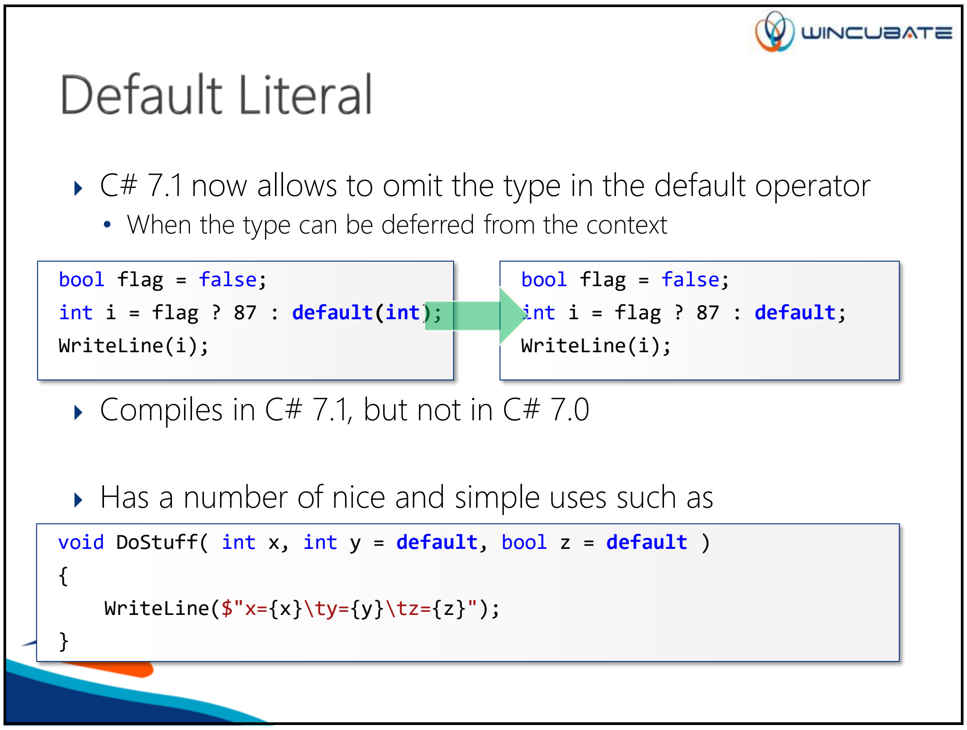
Internal compiler error reporting:
☐ Check for arithmetic overflow/
Output
Debugging information:
File alignment:
Library base address:

Pattern Matching Open Types

- Patterns now play well with (sub-)type constraints for generic types

```
static void Promote<T>( T employee )
{
    switch (employee)
    {
        case SoftwareArchitect sa:
            sa.Level = SoftwareArchitectLevel.Lead;
            break;
        case SoftwareEngineer se:
            se.Level = SoftwareEngineerLevel.Chief;
            break;
    }
}
```

- Compiles in C# 7.1, but not in C# 7.0



Default Literal

- C# 7.1 now allows to omit the type in the default operator
  - When the type can be deferred from the context

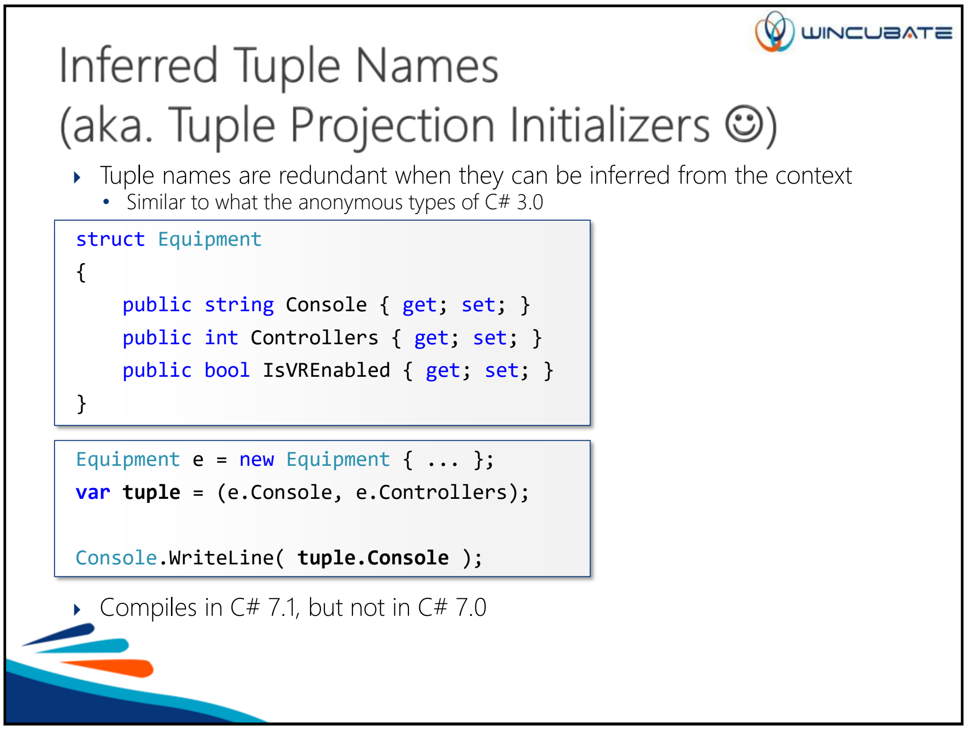
```
bool flag = false;
int i = flag ? 87 : default(int);
WriteLine(i);
```

```
bool flag = false;
int i = flag ? 87 : default;
WriteLine(i);
```

- Compiles in C# 7.1, but not in C# 7.0

- Has a number of nice and simple uses such as

```
void DoStuff( int x, int y = default, bool z = default )
{
    WriteLine($"x={x}\ty={y}\tz={z}");
}
```



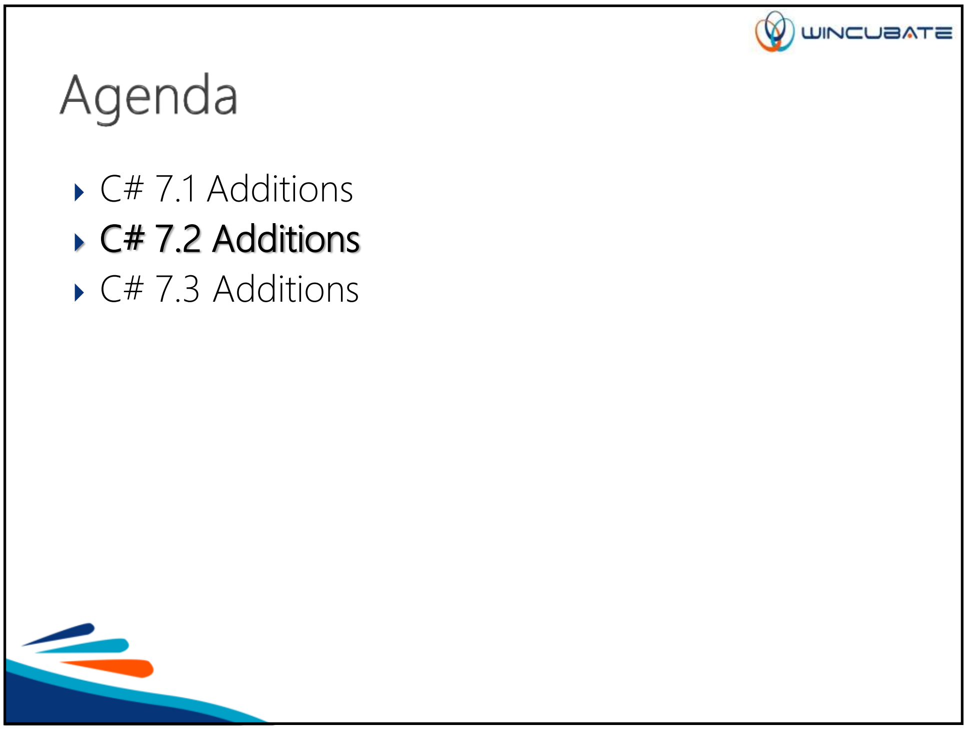
Inferred Tuple Names (aka. Tuple Projection Initializers ☺)

- Tuple names are redundant when they can be inferred from the context
  - Similar to what the anonymous types of C# 3.0

```
struct Equipment
{
    public string Console { get; set; }
    public int Controllers { get; set; }
    public bool IsVREnabled { get; set; }
}
```

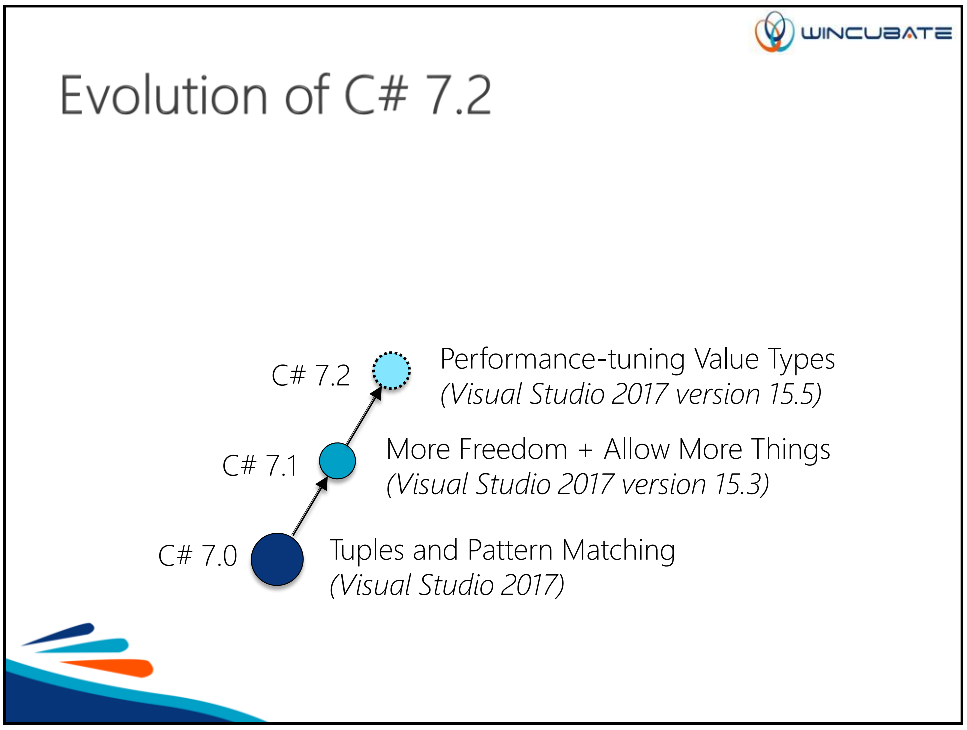
```
Equipment e = new Equipment { ... };
var tuple = (e.Console, e.Controllers);

Console.WriteLine( tuple.Console );
```

- Compiles in C# 7.1, but not in C# 7.0



Agenda

- C# 7.1 Additions
- **C# 7.2 Additions**
- C# 7.3 Additions

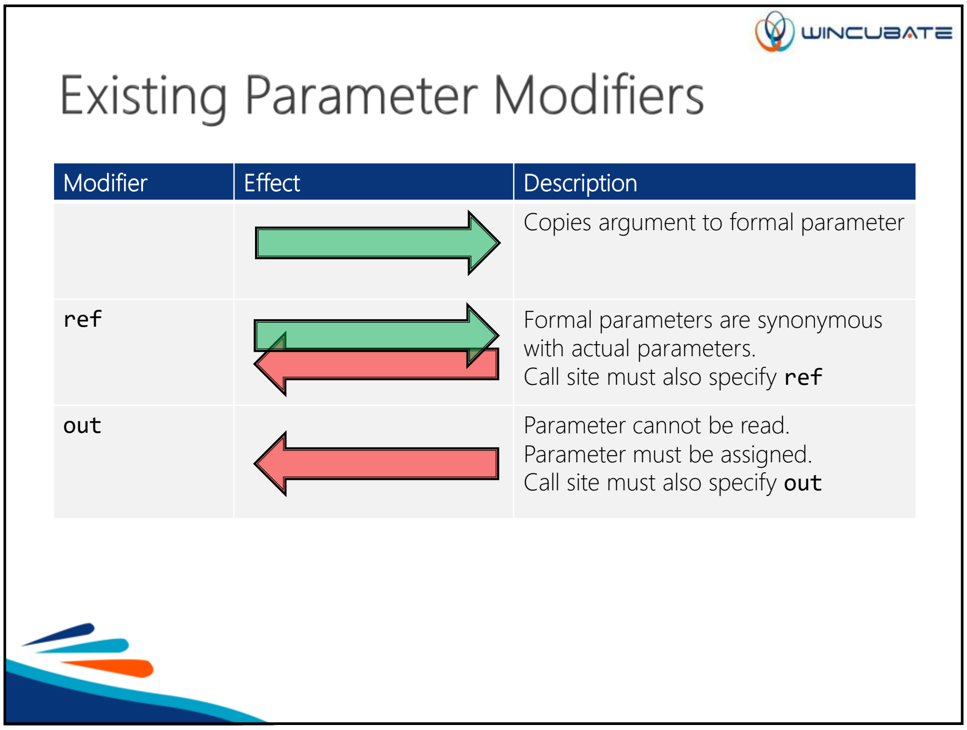


Evolution of C# 7.2

C# 7.2 — Performance-tuning Value Types *(Visual Studio 2017 version 15.5)*

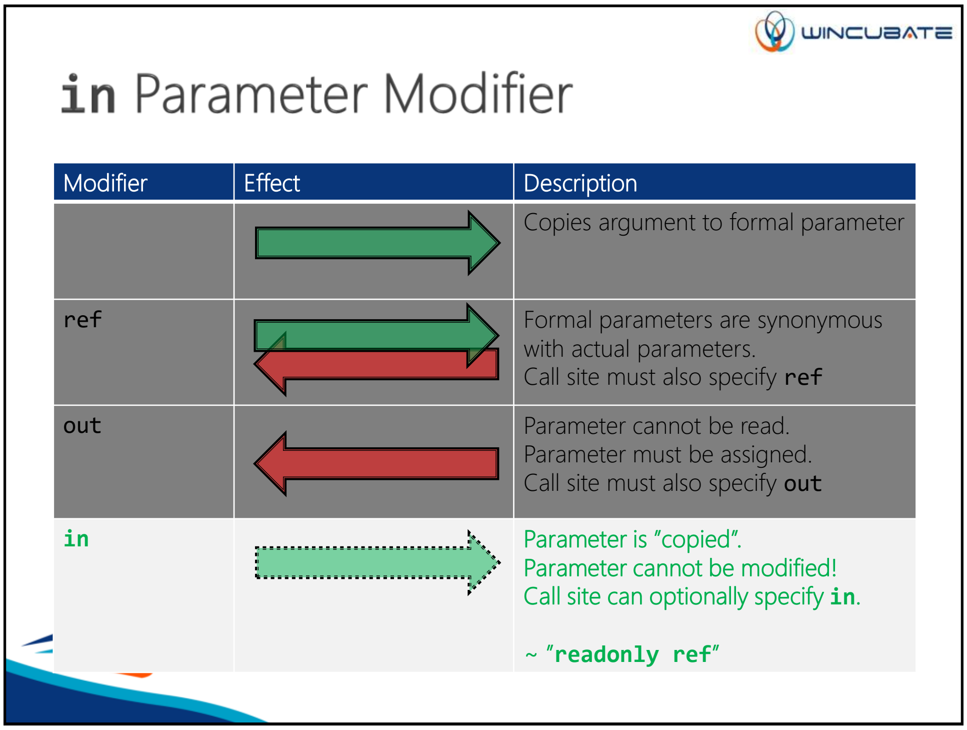
C# 7.1 — More Freedom + Allow More Things *(Visual Studio 2017 version 15.3)*

C# 7.0 — Tuples and Pattern Matching *(Visual Studio 2017)*



Existing Parameter Modifiers

Modifier	Effect	Description
	→	Copies argument to formal parameter
`ref`	⇄	Formal parameters are synonymous with actual parameters. Call site must also specify **`ref`**
`out`	←	Parameter cannot be read. Parameter must be assigned. Call site must also specify **`out`**



`in` Parameter Modifier

Modifier	Effect	Description
	→	Copies argument to formal parameter
`ref`	⇄	Formal parameters are synonymous with actual parameters. Call site must also specify **`ref`**
`out`	←	Parameter cannot be read. Parameter must be assigned. Call site must also specify **`out`**
`in`	⇢	Parameter is "copied". Parameter cannot be modified! Call site can optionally specify **`in`**. ~ "**`readonly ref`**"

in Parameter Modifier

- It can be passed as a reference by the runtime system for performance reasons

```
double CalculateDistance( in Point3D first, in Point3D second = default )
{
    double xDiff = first.X - second.X;
    double yDiff = first.Y - second.Y;
    double zDiff = first.Z - second.Z;

    return Sqrt(xDiff * xDiff + yDiff * yDiff + zDiff * zDiff);
}
```

- The call site does not need to specify **in**
- Can call with constant literal -> Compiler will create variable

```
Point3D p1 = new Point3D { X = -1, Y = 0, Z = -1 };
Point3D p2 = new Point3D { X = 1, Y = 2, Z = 3 };
double d = CalculateDistance(p1, p2));
```

Ref Readonly Returns

- Ref Returns can be enforced read-only by the compiler

```
ref readonly int FindMax( int[] numbers )
{
    int indexOfMax = 0;
    ...
    return ref numbers[indexOfMax];
}
```

```
ref readonly int max = ref FindMax(numbers);
WriteLine($"{nameof(max)} is now {max}");

max = 1000; // Not allowed!
```

- Must manually create a copy to make it modifiable later

```
int maxCopy = FindMax(numbers); // Copy
maxCopy = 999999;
```

Readonly Structs

- Define immutable structs for performance reasons

```
readonly struct Point3D
{
    public double X { get; }
    public double Y { get; }
    public double Z { get; }

    public Point3D( double x, double y, double z ) { ... }

    public override string ToString() => $"({X},{Y},{Z})";
}
```

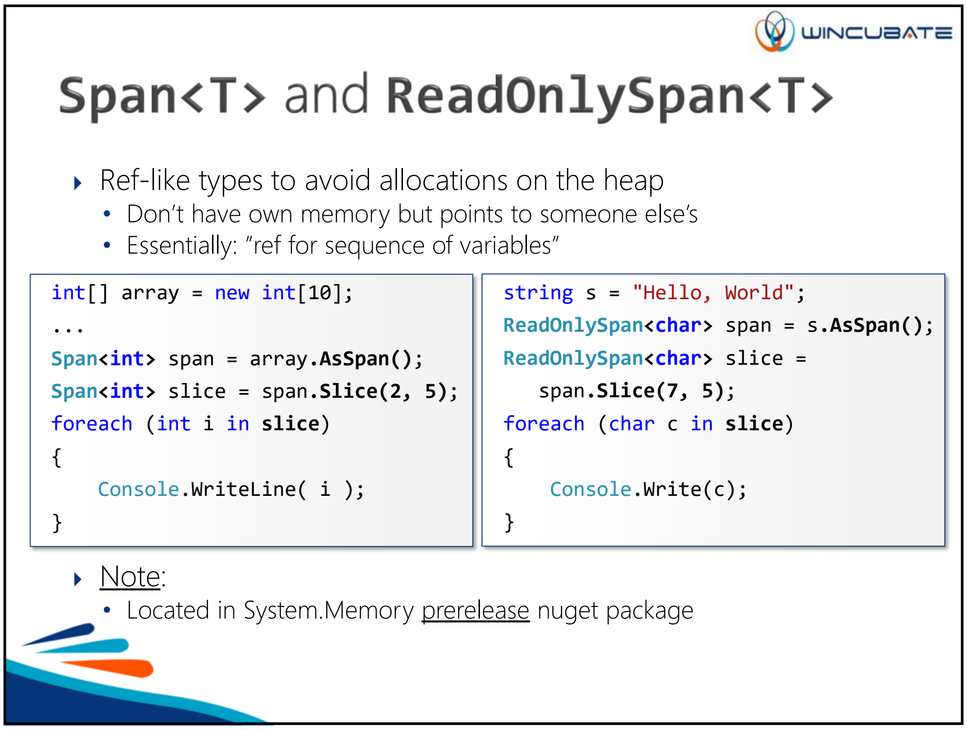
- Can always be passed as `in`
- Can always be `readonly ref` returned
- Compiler generates more optimized code for these values

Ref Structs

- Structs can be enforced as "always stack allocated" using `ref struct`

```
ref struct Point3D
{
    public double X { get; }
    public double Y { get; }
    public double Z { get; }
    ...
}
```

- These values can <u>never</u> be allocated on the heap
  - Cannot be boxed
  - Cannot be declared members of a class or (non-ref) struct
  - Cannot be local variables in async methods
  - Cannot be declared local variables in iterators
  - Cannot be captured in lambda expressions or local functions



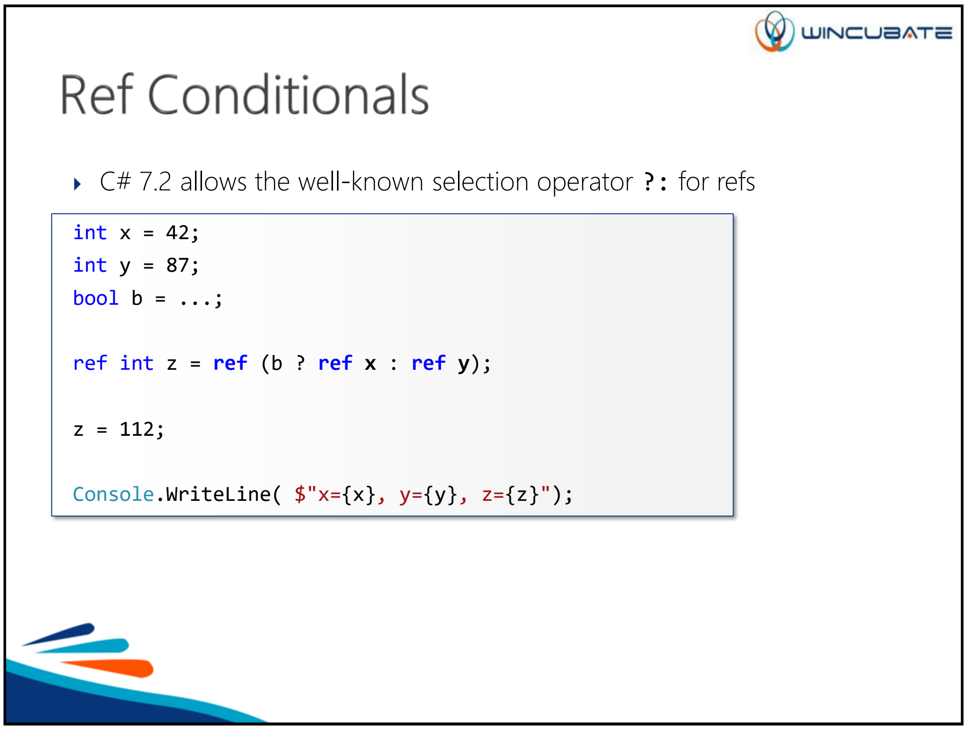
Span<T> and ReadOnlySpan<T>

- Ref-like types to avoid allocations on the heap
  - Don't have own memory but points to someone else's
  - Essentially: "ref for sequence of variables"

```
int[] array = new int[10];
...
Span<int> span = array.AsSpan();
Span<int> slice = span.Slice(2, 5);
foreach (int i in slice)
{
    Console.WriteLine( i );
}
```

```
string s = "Hello, World";
ReadOnlySpan<char> span = s.AsSpan();
ReadOnlySpan<char> slice =
   span.Slice(7, 5);
foreach (char c in slice)
{
    Console.Write(c);
}
```

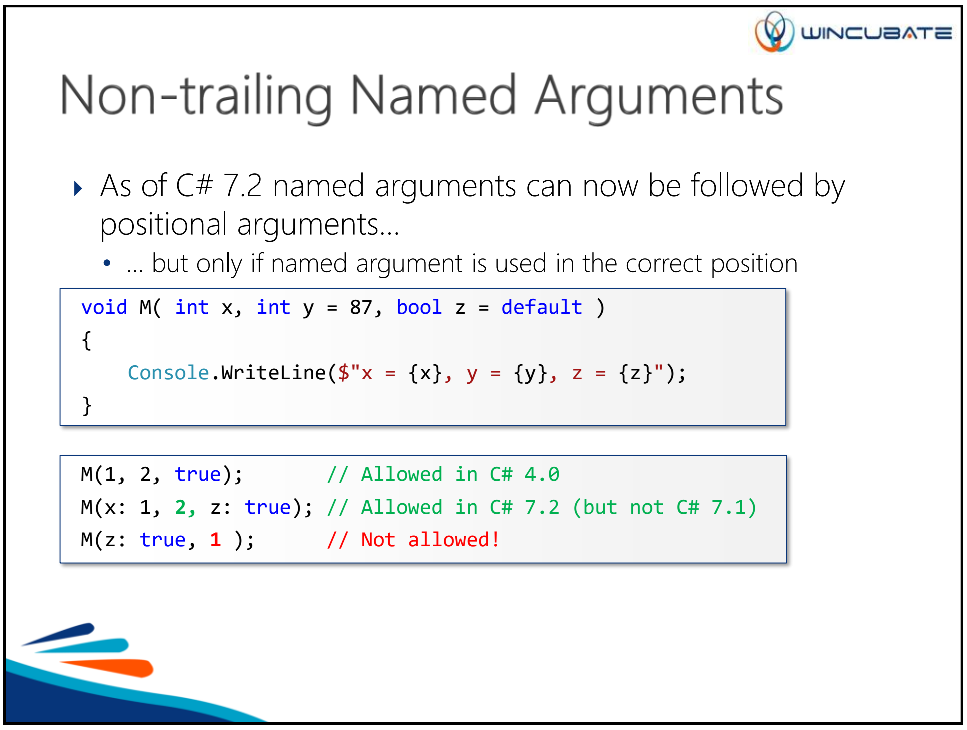
- Note:
  - Located in System.Memory prerelease nuget package



Ref Conditionals

- C# 7.2 allows the well-known selection operator `?:` for refs

```
int x = 42;
int y = 87;
bool b = ...;

ref int z = ref (b ? ref x : ref y);

z = 112;

Console.WriteLine( $"x={x}, y={y}, z={z}");
```

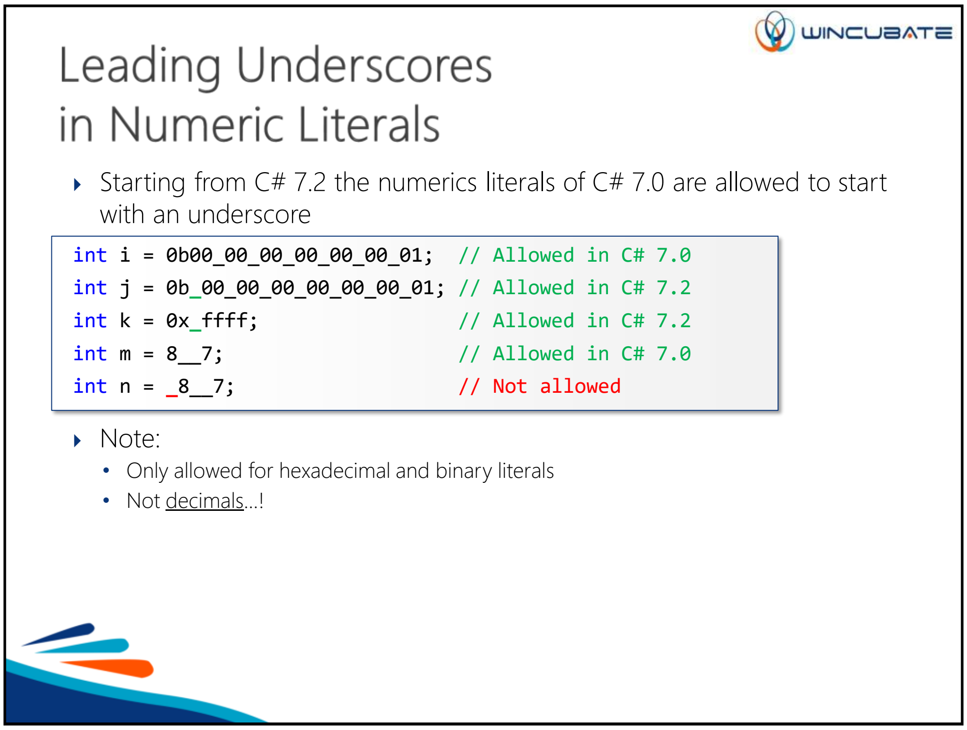


Non-trailing Named Arguments

- As of C# 7.2 named arguments can now be followed by positional arguments…
  - … but only if named argument is used in the correct position

```
void M( int x, int y = 87, bool z = default )
{
    Console.WriteLine($"x = {x}, y = {y}, z = {z}");
}
```

```
M(1, 2, true);        // Allowed in C# 4.0
M(x: 1, 2, z: true); // Allowed in C# 7.2 (but not C# 7.1)
M(z: true, 1 );       // Not allowed!
```



Leading Underscores in Numeric Literals

- Starting from C# 7.2 the numerics literals of C# 7.0 are allowed to start with an underscore

```
int i = 0b00_00_00_00_00_00_01;  // Allowed in C# 7.0
int j = 0b_00_00_00_00_00_00_01; // Allowed in C# 7.2
int k = 0x_ffff;                 // Allowed in C# 7.2
int m = 8__7;                    // Allowed in C# 7.0
int n = _8__7;                   // Not allowed
```

- Note:
  - Only allowed for hexadecimal and binary literals
  - Not decimals…!

`private protected` Access Modifier

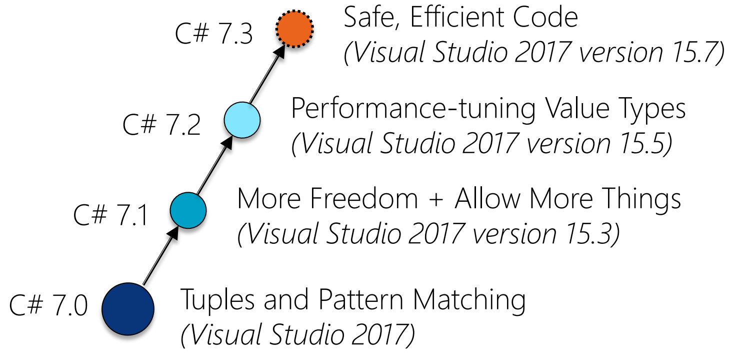
- Since C# 1: `protected internal` — `"protected || internal"`
  - Is visible to types in same assembly
  - Is visible to derived classes (in **same** or **other** assemblies)
- New in C# 7.2: `private protected` — `"protected && internal"`
  - Is visible to containing types
  - Is visible to derived classes in the **same** assembly

```
public class ClassInOtherAssembly
{
    private protected int X { get; set; }

    public void Print() => Console.WriteLine(X);
}
```

Summary

- C# 7.1 Additions
- C# 7.2 Additions
- **C# 7.3 Additions**

Evolution of C# 7.3


C# 7.3
Safe, Efficient Code
(Visual Studio 2017 version 15.7)
C# 7.2
Performance-tuning Value Types
(Visual Studio 2017 version 15.5)
C# 7.1
More Freedom + Allow More Things
(Visual Studio 2017 version 15.3)
C# 7.0
Tuples and Pattern Matching
(Visual Studio 2017)

Tuple Comparison Now Works…!

- C# 7.0 built-in implicit tuple conversions
  - `ToString()` + `Equals()` + `GetHashCode()`
- C# 7.3 completes comparison by adding `==` and `!=`

```
var t0 = (4, 8);


var t1 = (a: 8, b: 4);
var t2 = (8, 4);
(int x, int y) t3 = (8, 4);
(double p, double q) t4 = (8, 4);
```

```
WriteLine(t0 != t1);


WriteLine(t1 == t2);
WriteLine(t1 == t3);
WriteLine(t2 == t3);
WriteLine(t3 == t4);
```

- Performs component-wise `==` and `!=` with implicit conversions

Ref Locals Reassignment

- C# 7.0 added references in the style of C++
- C# 7.3 completes ref locals by allowing them to be reassigned

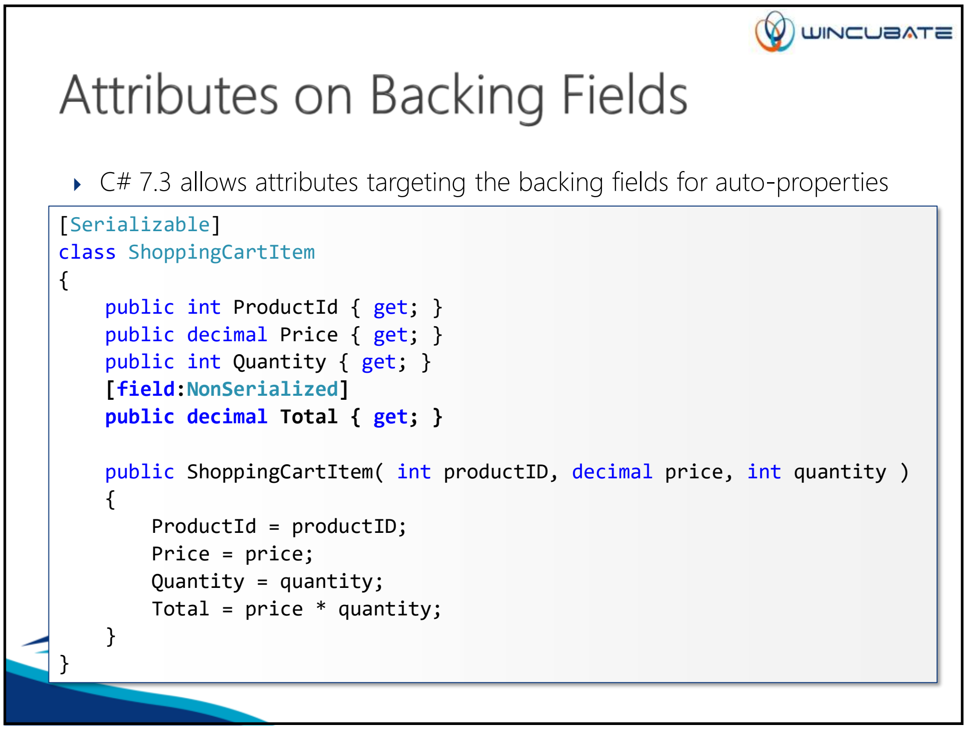
```
int x = 42;
int y = 87;
ref int z = ref x; // Declaration and Initialization of z;

x = 112;
WriteLine($"z = {z}");

z = ref y; // Reassignment of z;
WriteLine($"z = {z}");
```

Expression Variables in Initializers

- More flexible initialization was introduced in C# 7.0
- C# 7.3 extends out var and pattern variables to initializers

```
class Base
{
    public int Coordinate { get; } =
        int.TryParse("hello", out int x) ? x : default;

    public Base( int coordinate = default ) => Coordinate = coordinate;
}
```

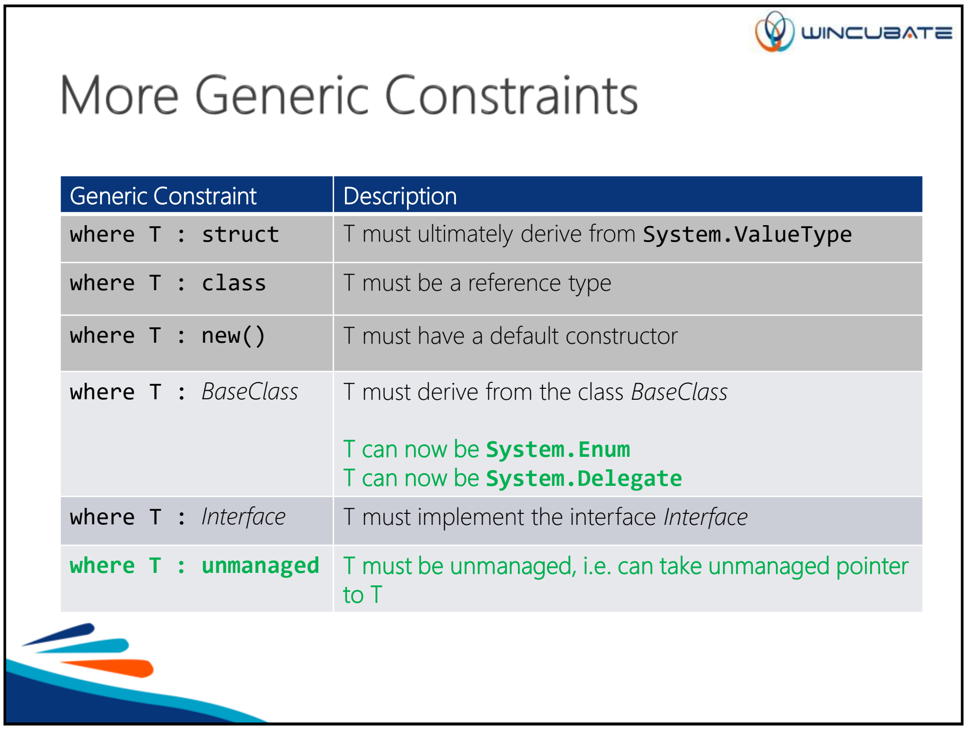
```
class Derived : Base
{
    public Derived( object o ) : base(o is Point p ? p.X : default)
    {
    }
}
```



Attributes on Backing Fields

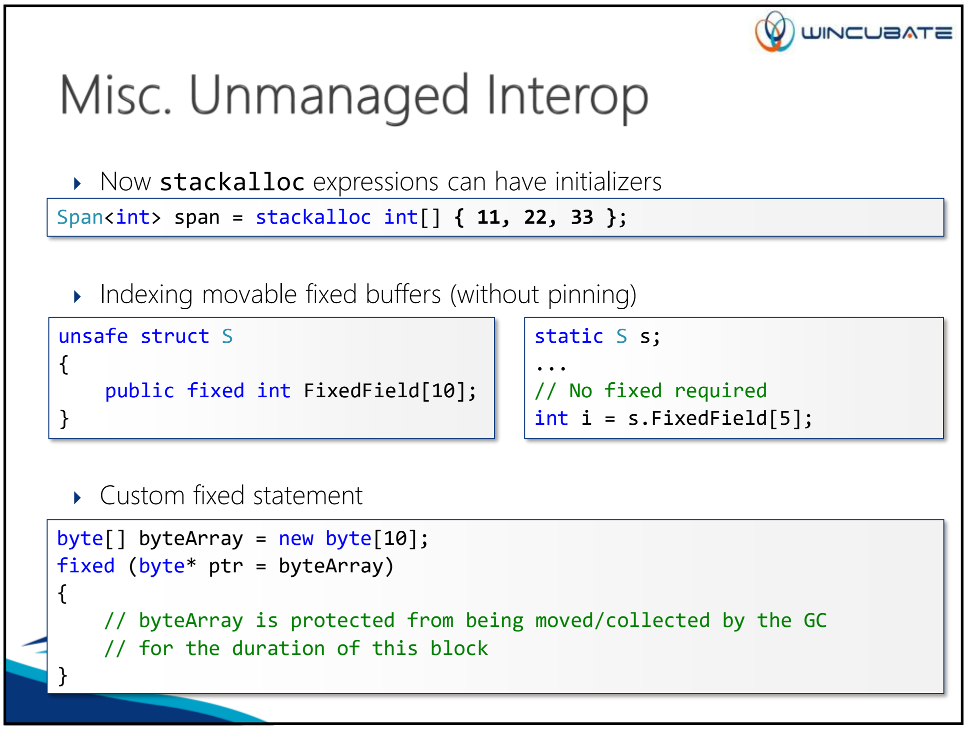
- C# 7.3 allows attributes targeting the backing fields for auto-properties

```
[Serializable]
class ShoppingCartItem
{
    public int ProductId { get; }
    public decimal Price { get; }
    public int Quantity { get; }
    [field:NonSerialized]
    public decimal Total { get; }

    public ShoppingCartItem( int productID, decimal price, int quantity )
    {
        ProductId = productID;
        Price = price;
        Quantity = quantity;
        Total = price * quantity;
    }
}
```



More Generic Constraints

Generic Constraint	Description
`where T : struct`	T must ultimately derive from **`System.ValueType`**
`where T : class`	T must be a reference type
`where T : new()`	T must have a default constructor
`where T :` *BaseClass*	T must derive from the class *BaseClass* T can now be **`System.Enum`** T can now be **`System.Delegate`**
`where T :` *Interface*	T must implement the interface *Interface*
`where T : unmanaged`	T must be unmanaged, i.e. can take unmanaged pointer to T



Misc. Unmanaged Interop

- Now `stackalloc` expressions can have initializers

```
Span<int> span = stackalloc int[] { 11, 22, 33 };
```

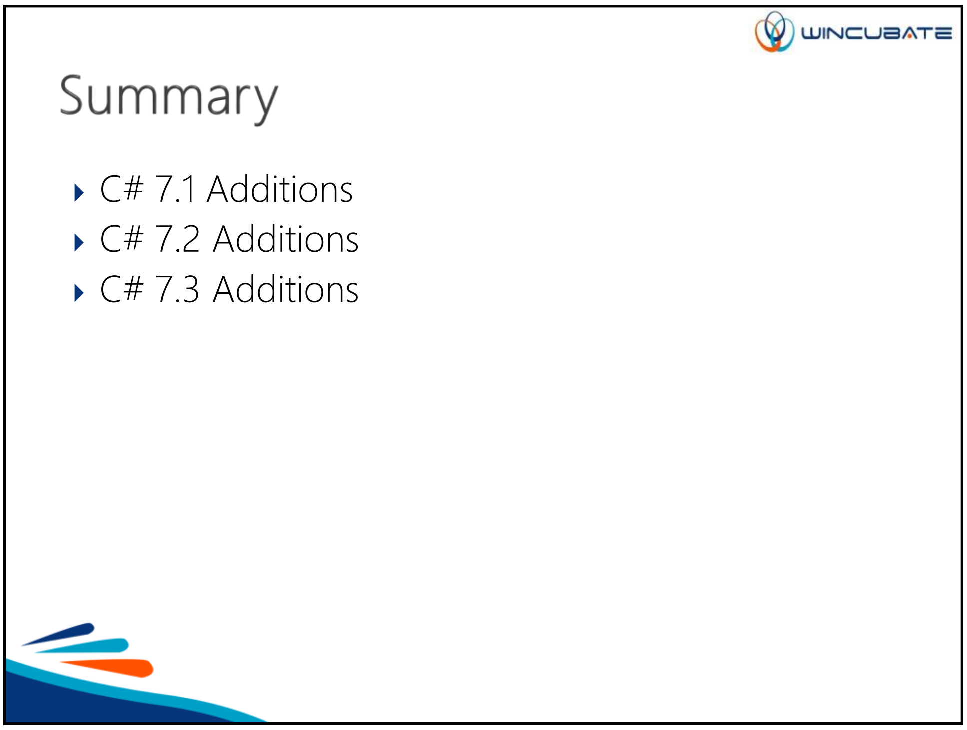
- Indexing movable fixed buffers (without pinning)

```
unsafe struct S
{
    public fixed int FixedField[10];
}
```

```
static S s;
...
// No fixed required
int i = s.FixedField[5];
```

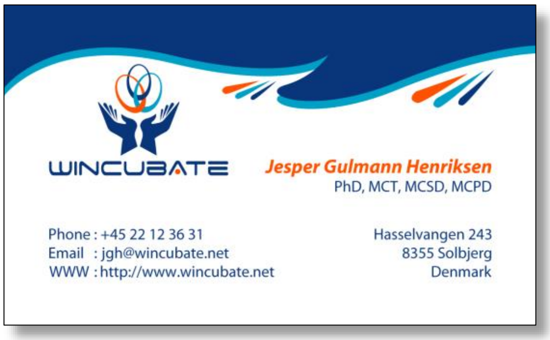
- Custom fixed statement

```
byte[] byteArray = new byte[10];
fixed (byte* ptr = byteArray)
{
    // byteArray is protected from being moved/collected by the GC
    // for the duration of this block
}
```



Summary

- C# 7.1 Additions
- C# 7.2 Additions
- C# 7.3 Additions



Jesper Gulmann Henriksen
PhD, MCT, MCSD, MCPD

Phone : +45 22 12 36 31
Email : jgh@wincubate.net
WWW : http://www.wincubate.net

Hasselvangen 243
8355 Solbjerg
Denmark