# Welcome to C# 7.1

Mads Torgersen

October 31st, 2017

With C# we have always tended towards major releases: bundle a lot of features up, and release less frequently. We even went so far as routinely omitting the traditional ".0" when we talked about C# 6.0!

In the C# 7.0 "wave" we are trying something new. Tools such as Visual Studio upgrade on a frequent cadence, and there's no longer a technical reason why C# couldn't also be updated more frequently. So this time around we are embracing a notion of "point releases"; minor versions of C# that trickle out useful but smaller language features with shorter intervals. This means that you don't have to wait so long for additional value to ship, but also makes it easier to align a C# release with the shipping of related features, e.g. in .NET.

Of course, upgrading to a new version of the language "all the time" can be a hassle in an organization, and isn't for everyone. Visual Studio 2017 lets you decide whether to snap to the latest, or only to major (".0") versions of C#. You can choose your cadence.

## The first point release

In August of 2017 we shipped the first point release of C#. It's called C# 7.1, and its main purpose is really for us to get practice with point releases, without having too many dependencies on accompanying technology to complicate matters.

C# 7.1 is therefore a small release with just a few (but well-chosen) new language features; useful we think, but definitely in the minor range. It's supported by Visual Studio 2017, starting with Update 15.3.

Let's have a look! Here's a program that uses the three C# 7.1 features we're going to cover, plus a number of the recent C# 7.0 ones, just to make it interesting.

```
1  using System.Linq;
2  using System.Threading;
3  using System.Threading.Tasks;
4
5  using static System.Console;
6
7  class Program
8  {
9      static async Task Main(string[] args)
10     {
11         var results = Enumerable.Range(1, 40)
12             .Select(input => (input, task: FibonacciAsync(input)))
13             .ToArray();
14
15         foreach (var tuple in results)
16         {
17             WriteLine($"Fib {tuple.input} = {await tuple.task}");
18         }
19     }
20
21     private static Task<int> FibonacciAsync(int n, CancellationToken token = default)
22     {
23         return Task.Run(() => Fib(n).curr, token);
24
25         (int curr, int prev) Fib(int i)
```

```
        {
            if (i is 0) return (1, 0);
            var (c, p) = Fib(i - 1);
            return (c + p, c);
        }
    }
}
```

cs7.1-Example.cs hosted with ❤ by GitHub view raw

It computes the first 40 Fibonacci numbers in parallel on the thread pool, and prints them out in order.

Let's look at each of the new C# 7.1 features used in here. For a full overview of the features in C# 7.1, check out the docs.

## Async Main

The `Main` entry point method can now return a `Task` or a `Task<int>`. When it does, execution will wait for the returned task to complete, before shutting down the program.

Of course, the common use of this will be to make the `Main` method `async`, which you can also do:

```
static async Task Main(string[] args)
```

This lets you `await` directly in the `Main` method, something you couldn't do before.

```
WriteLine($"Fib {tuple.input} = {await tuple.task}");
```

What you had to do previously was quite unappetizing: first you'd create an async helper method, `MainAsync`, say, with all the logic in. Then you'd write this cryptic `Main` method:

```
static void Main(string[] args) => MainAsync().GetAwaiter().GetResult();
```

Now you can just make your `Main` method async, and the compiler will rewrite it for you.

## Inferred tuple element names

In this lambda expression inside the query:

```
input => (input, task: FibonacciAsync(input))
```

You notice that we create a tuple, but only give a name, `task`, for the second element. Yet a few lines later we are able to say

```
WriteLine($"Fib {tuple.input} = {await tuple.task}");
```

Accessing the first element by the name `tuple.input`. That's because when you create a tuple with an expression that "has" a name, like `input` in the lambda expression above, we'll now automatically give the corresponding tuple element that name.

## Default literals

If there's an expected type for a `default` expression, you can now omit the mention of the type, as we do for the `CancellationToken` in in the signature of the `FibonacciAsync` method:

```
private static Task<int> FibonacciAsync(int n, CancellationToken token = default)
```

This avoids tedious repetition of type names, or typing out long ones when they are already given by context.

## What's next?

We are already working on C# 7.2, as well as features that are intended for the next major release. If you are curious, you can follow along and participate at the C# language design GitHub repo: github.com/dotnet/csharplang.

Happy hacking!

**Mads Torgersen**, Lead Designer of C#

---

Mads Torgersen C# Lead Designer, .NET Team
**Follow**