Hainton

Digital Solutions

C#/.NET Source Generators

Writing code, to write code.

Peter “Shawty” Shaw/Digital Solutions UK

Introductions

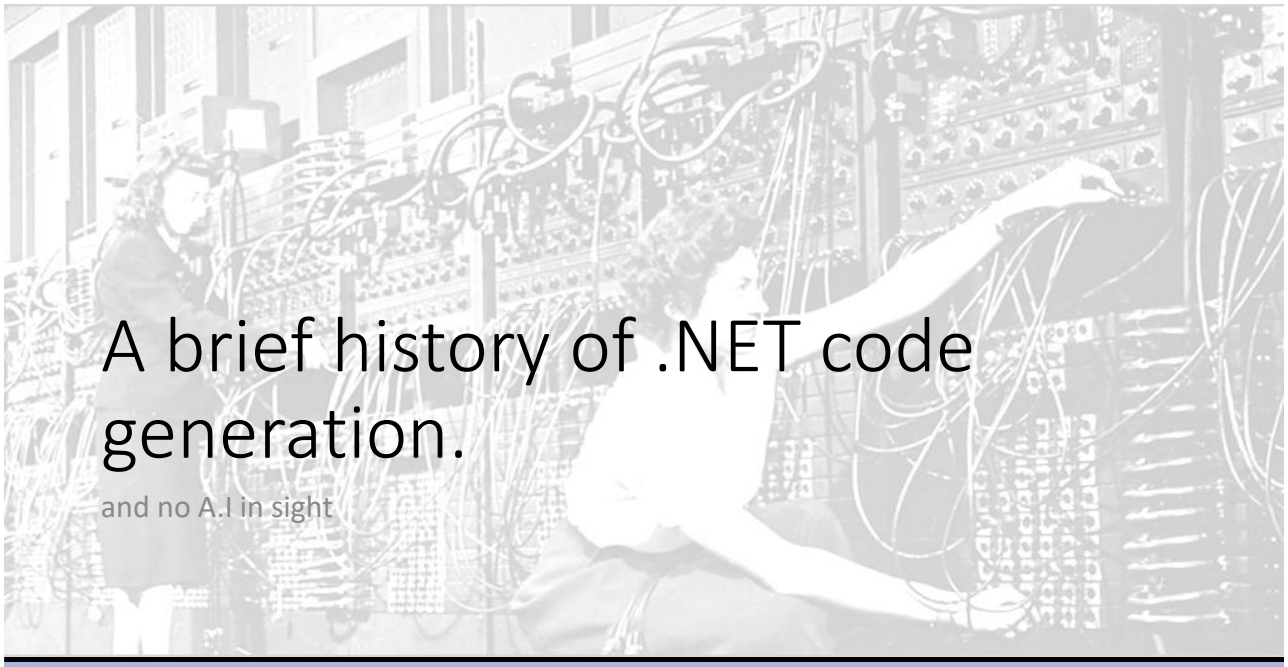
- Many of you will already know me, I’ve been a .NET developer here in the north east for a long time now.
- I generally go by the name of “Shawty”, but I answer to just about anything especially in a beer induced environment 😁
- Please note:
  - **I am Clinically Deaf**, and wear hearing aid’s in order to hear, these don’t always have the effect they should, especially when there’s a lot of background noise, if you want to ask me questions during this session, you might have to raise your hand, or jump about like a lunatic to attract my attention.

Introductions

- If you've not met me before, then you can stalk me via the following means:
  - Email – shawty.d.ds@googlemail.com
  - Twitter - @shawty_ds
  - Linked-in – "Peter Shaw"/"LiDNUG" (The online .NET user group I help run)
- The code for this session is available from my Github repo at:
  - https://github.com/shawty/codegeneratorshainton2022

Tonight's Agenda

- As the title suggests we'll be taking a look at the new source generation features in the latest .NET builds, covering the following topics:
  - Brief history of .NET code generation
  - What source generation options are currently in use in .NET
  - Why do we need another source generation option?
  - Demos of the new features
  - Where are we currently with the new offering
  - What has the .NET community already done with the new features
  - Summary



A brief history of .NET code generation.

and no A.I in sight

A brief history

- The first properly known code generation tool for any platform called “Matlab” was released in 1984, and is still in use to day in a surprising number of arenas.
- Matlab generates code for everything from “software designed radio” to huge scale “Industrial robotics”
- It’s a proprietary tool, that costs a lot to license, but has been customised and released as a “tool to control” a lot of different systems.

A brief history

- The FIRST code generation platform specifically for .NET was called “outsystems” and was released in 2001
- Unfortunately it’s a very closed platform, and not much information is available about it
- These days OUTSYSTEMS are known for “low code” and “no code” tools and are backed heavily by investors in the finance industry.

A brief history

- The most well known historical 3rd party code generation tool that worked for .NET is “Codesmith Generator”
- Codesmith is still available today, but it is now showing it’s age, it still works with the old .NET framework, but can generate any code you want.
- It uses a template system that looks very much like classic ASP web pages (Pre ASP.NET)
- Having used Codesmith myself, it’s a stable application, but hard to control in modern day CI/CD pipeline.

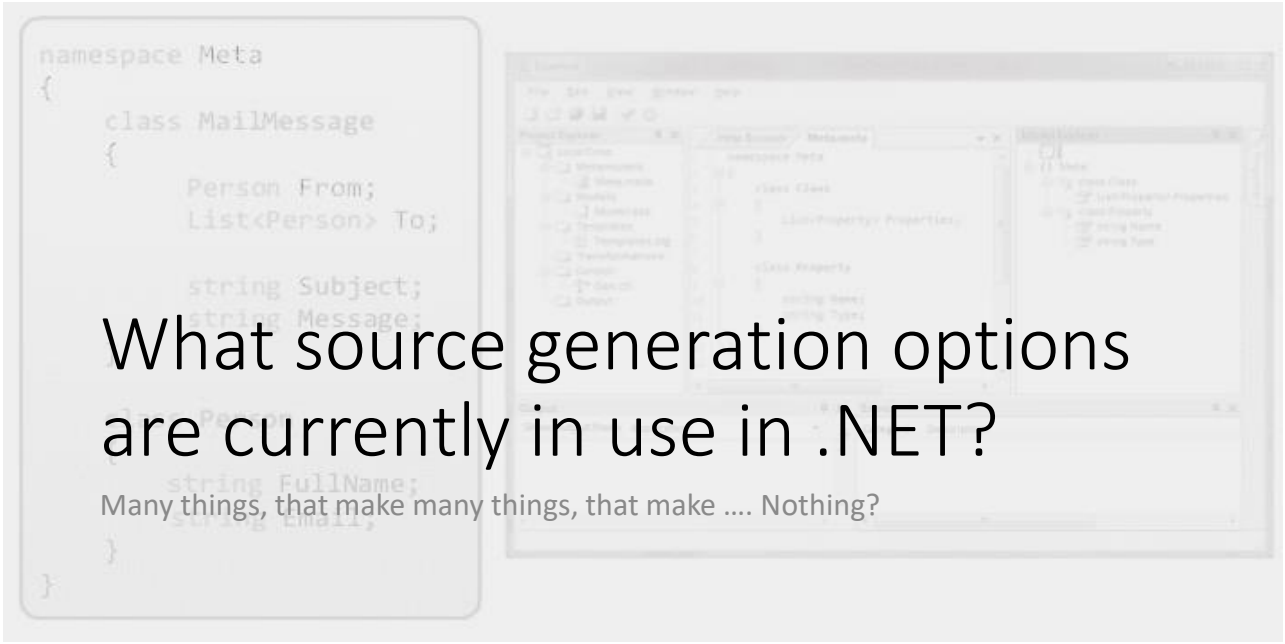
A brief history

- The first actual built in code generation system, provided by Microsoft in their dev tools, was “T4 Templates” released in 2005.
- The T4 system is now open source and available under an MIT license, but no one has taken up the challenge to improve it for the modern day tooling.
- T4 is still available even in Visual Studio 2022, but there is no syntax highlighting, intelli-sense or code completion and is quite cumbersome to use.
- T4 up until now however, remained the best built in tool in the VS ecosystem for code generation.

A brief history

- Notable mentions
  - Accelerator by Surround Technologies, released 2007
  - Code4Green by Code 4 Green Ltd, released 2009
  - Reegenerator by Kodeo Ltd, released 2010
  - CodeBaght by CodeBaght LLC, released 2014

From 2014 onwards, the landscape for code generators has been very quiet, Microsoft released things like Linq to SQL, but they where largely very specialised and only generated one thing.



What source generation options are currently in use in .NET?

Many things, that make many things, that make …. Nothing?

What's currently in use?

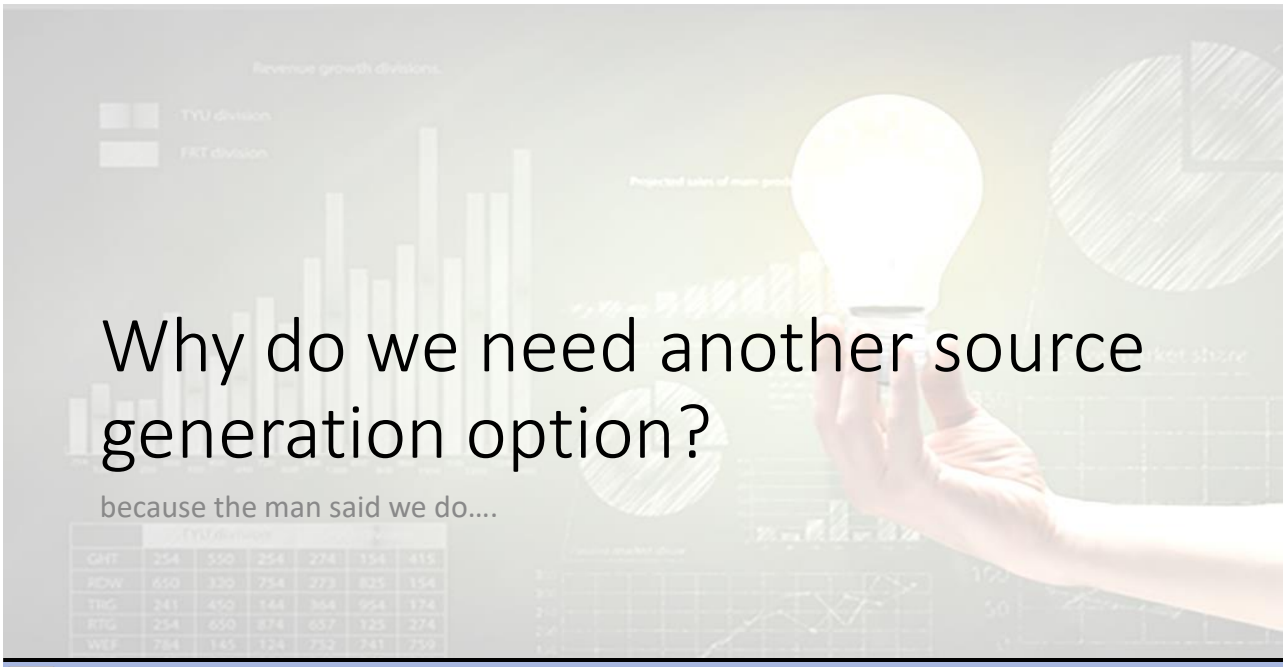
- The two main contenders for 3rd party tools are
  - Codesmith generator
  - T4 Templates
- Many developers have resorted to homebrew tools, code project and git-hub are littered with tools to generate various data abstraction layers from various database technologies.
- A number of ORM systems have CLI based tooling that can be called inside a CI/CD pipeline to perform various tasks.

What's currently in use?

- JavaScript environments have a stupid amount of tools for generating code, just to mention a few
  - Parcel
  - Webpack
  - Yeoman
  - Most of the CLI build tools for things like Angular, Aurelia and others
  - Even NPM can generate code

What's currently in use?

- In C#/.NET itself, the choices are very limited unless you want to use T4 or pay a lot of money.
- There are many open source tools, but you take what you get using this route.
- Reflection inside the runtime of your app is possible, as is the Roslyn compiler.
- Most of the well known NuGet packages for code generation, are VERY specific (EG: Swagger for API gen)
- Systems like Linq2Sql and EF Migrations are still a thing, but can be cumbersome to get right.



Why do we need another source generation option?

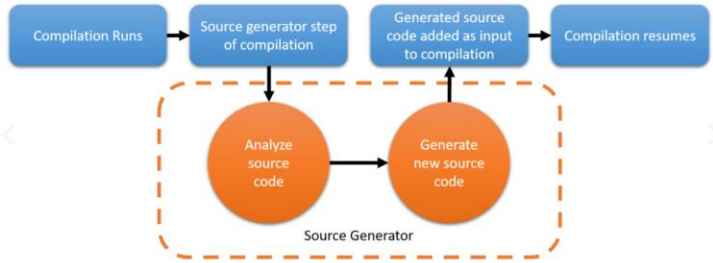
because the man said we do….

Why do we need another option?

- In all honesty, we don't actually, what we need is an option that runs where we need it too.
- The existing systems all run:
  - Pre build
  - At specific points in the build beyond our control
  - For specific scenarios only
  - Post Build

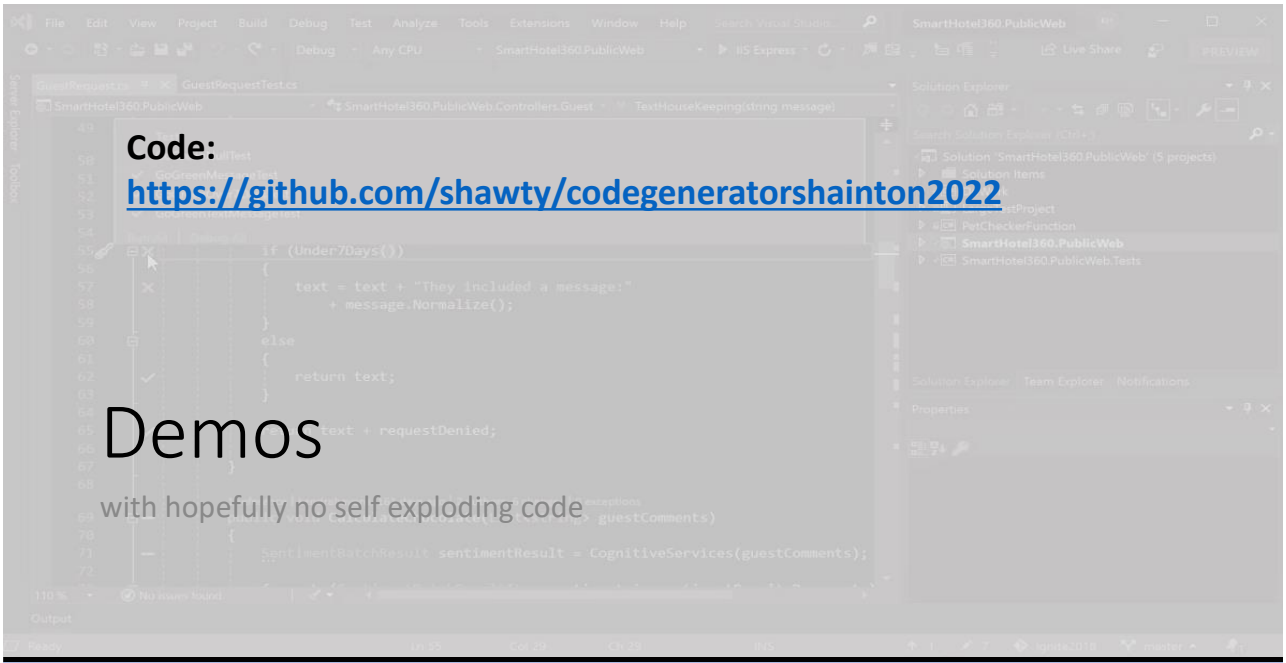
Why do we need another option?

- The new source generation features run during compilation of your regular source files.
- They generate NEW source code while the compile is currently in progress.



Why do we need another option?

- Source generators are built on the back of the Roslyn Compiler technologies and libraries.
- They use the same frameworks as Roslyn Code Analysers.
- Source generators CAN ONLY GENERATE C# Source code, they CANNOT create HTML, CSS, XML, Json or anything that is non compliable.



Demos…

- IMPORTANT NOTES FOR THE GENERATOR:
  - Generator classes **MUST** be .NET Standard 2 targeted
  - Generator classes **MUST** use the [Generator] attribute
  - Generator classes **MUST** inherit from ISourceGenerator
  - The following NuGet packages **ARE REQUIRED** in the generator project
    - Microsoft.CodeAnalysis.Csharp (4.1.0 or higher)
    - Microsoft.CodeAnalysis.Analyzers (3.3.3 or Higher)
- If you want to debug, you either have to develop your generator as a standard class lib, or trap the debugger in init.

Demos…

- IMPORTANT NOTES FOR THE TARGET PROJECT:
  - You have to add your generator class manually by editing the project config file
  - You **MUST** set the output type of the reference to “Analyzer”
  - You **MUST** set “ReferenceOutput” of the reference to “false”

Where are we currently with the new offering?

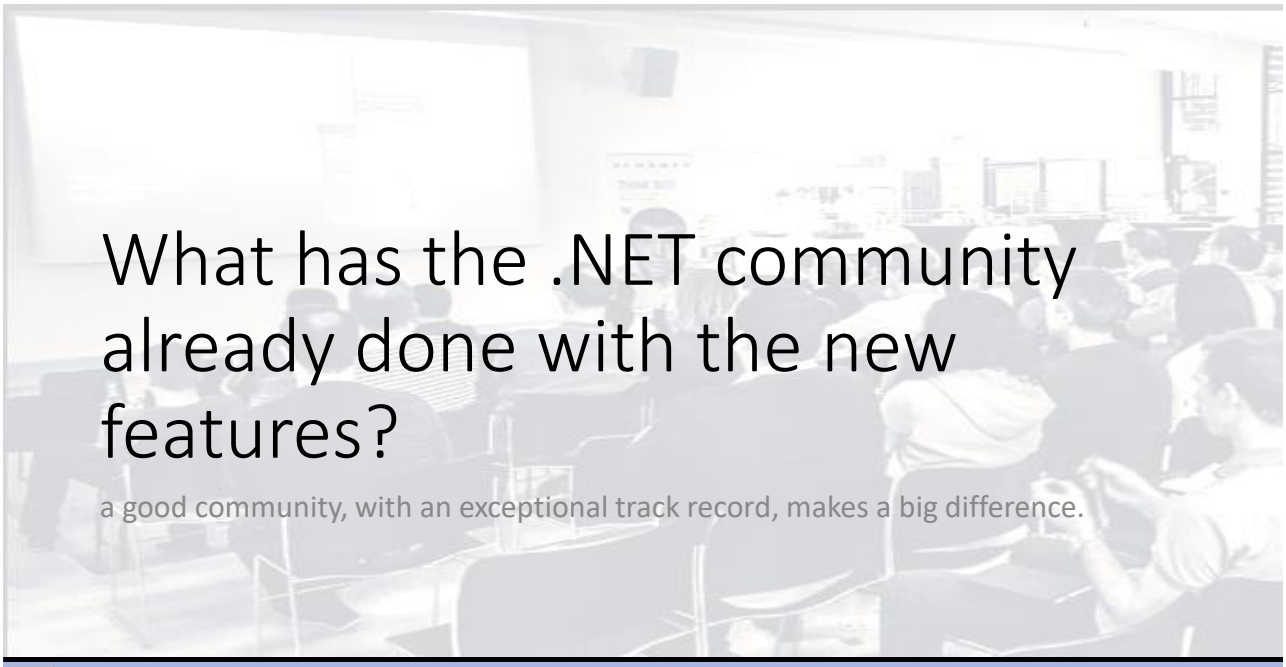
It’s always harder at first

Where are we currently?

- The technology is very simple to use
- It works very well
- Adding it to a project is still somewhat manual
- You can use NuGet to deploy generators but you have to tell NuGet it's a Roslyn analyser.
- Debugging could be better

Where are we currently?

- Microsoft ARE putting their weight behind this, they've already announced a slew of new features around the technology.
- One really interesting idea is "Regex Source Generators"
- Another idea that has been touted as "Very Useful" is auto generation of "Notify Properties" from "Normal Properties" in WPF/MVVM models



What has the .NET community already done with the new features?

a good community, with an exceptional track record, makes a big difference.

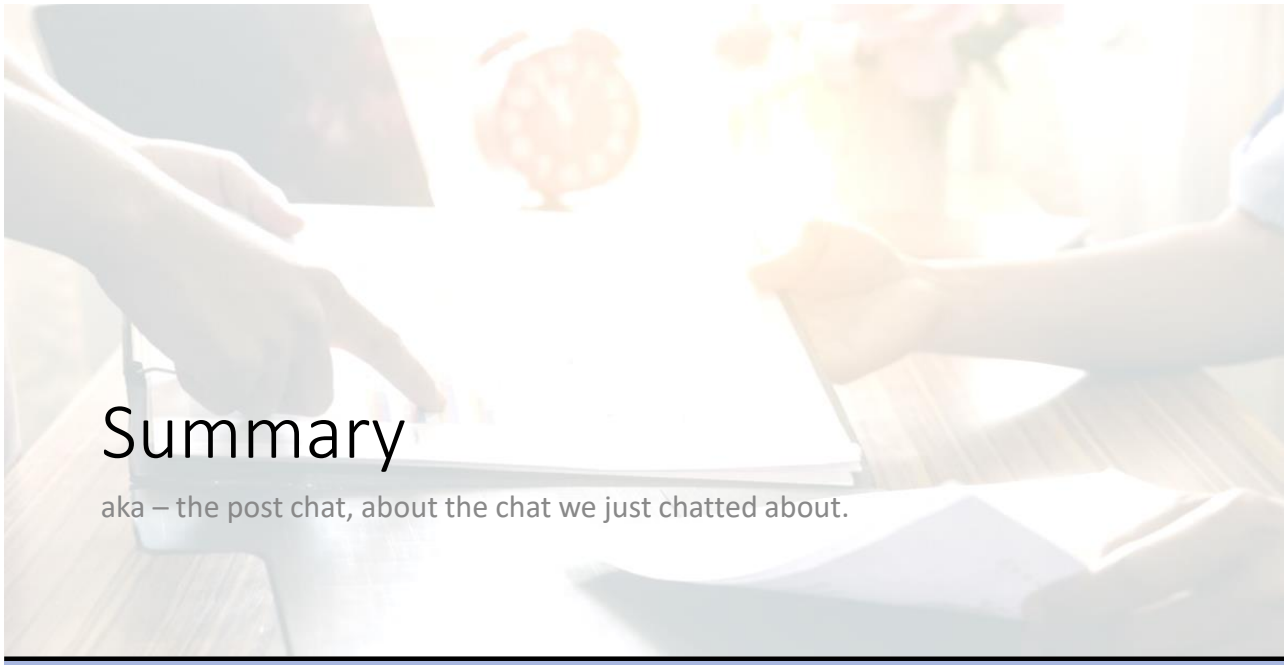
What has the community done?

- As is usually the case, the .NET community have been feverously busy putting the new technology through it's paces.
- There are plenty of blog posts
  - https://medium.com/rocket-mortgage-technology-blog/generating-code-in-c-1868ebbe52c5
  - https://nicksnettravels.builttoroam.com/debug-code-gen/
  - https://blog.jetbrains.com/dotnet/2020/11/12/source-generators-in-net-5-with-resharper/
  - https://devblogs.microsoft.com/dotnet/using-c-source-generators-to-create-an-external-dsl/

What has the community done?

- As is usually the case, the .NET community have been feverously busy putting the new technology through it's paces.

- We already have the makings of an "Awesome xxxxx" page for it:
  - https://github.com/amis92/csharp-source-generators
- And the obligatory Microsoft samples project for it too:
  - https://devblogs.microsoft.com/dotnet/new-c-source-generator-samples/
- Microsoft also publish the "Source Generators Cookbook"
  - https://github.com/dotnet/roslyn/blob/main/docs/features/source-generators.cookbook.md



Summary

aka – the post chat, about the chat we just chatted about.

Summary

- Source generators fill in a missing gap in the .NET build system that until now has been difficult to get at.
- It's still an early technology with a few rough edges, but it has great promise.
- Yes debugging is cumbersome, but there are ways to deal with that easily until things improve.
- You can generate pretty much ANY extra source you want and it can be as dynamic as you want, your imagination is the only restriction.
- For everything else T4 is still a great option.
- The .NET community as expected has stepped up and taken the bull by the horns with some fantastic results.

Code & Slides:

https://github.com/shawty/codegeneratorshainton2022

Contacting me:

Twitter: @shawty_ds
Email: shawty.d.ds@googlemail.com
Search: "shawty/ds", "shawty/Lidnug"

Thanks for listening