Using Source Generators for Fun (and Maybe Profit)

Jason Bock

Personal Info

- http://www.jasonbock.net
- https://www.twitter.com/jasonbock
- https://www.github.com/jasonbock
- https://www.youtube.com/c/JasonBock
- jason.r.bock@outlook.com

Downloads

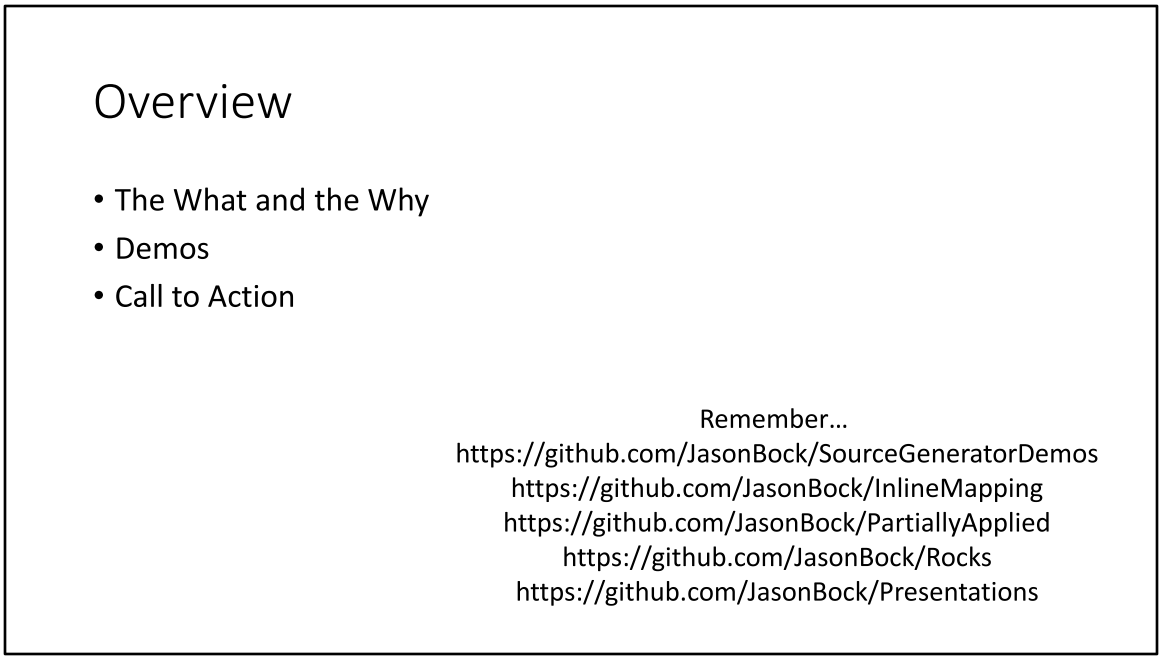
https://github.com/JasonBock/SourceGeneratorDemos

https://github.com/JasonBock/InlineMapping

https://github.com/JasonBock/PartiallyApplied

https://github.com/JasonBock/Rocks

https://github.com/JasonBock/Presentations

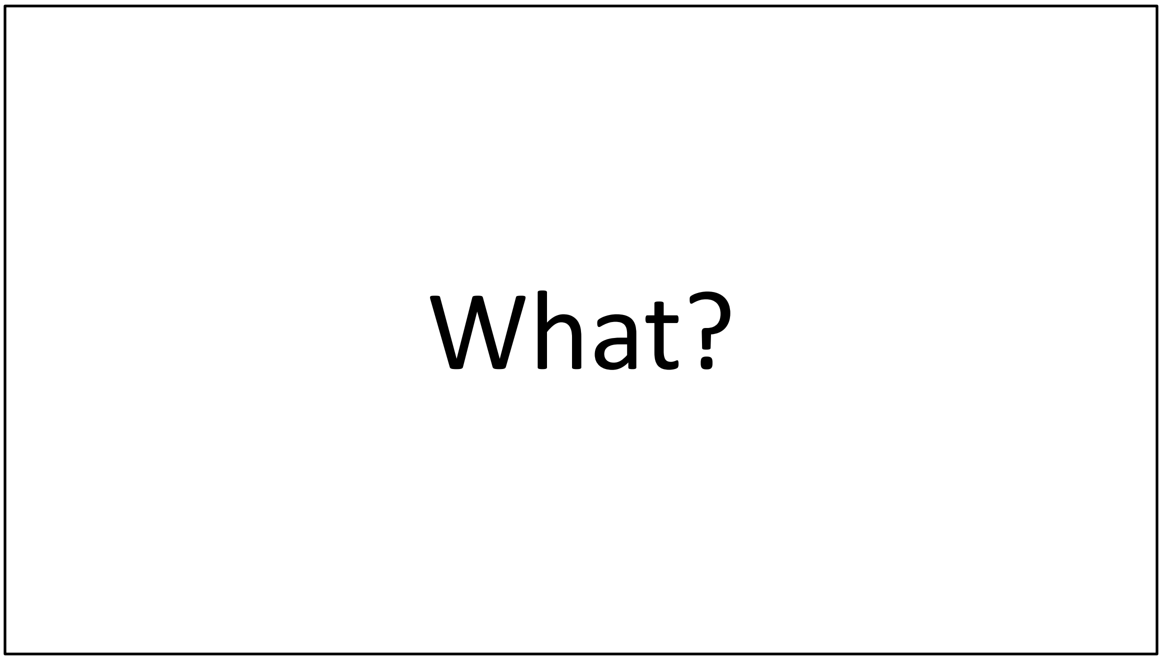


Overview

- The What and the Why
- Demos
- Call to Action

Remember…
https://github.com/JasonBock/SourceGeneratorDemos
https://github.com/JasonBock/InlineMapping
https://github.com/JasonBock/PartiallyApplied
https://github.com/JasonBock/Rocks
https://github.com/JasonBock/Presentations

The What and the Why

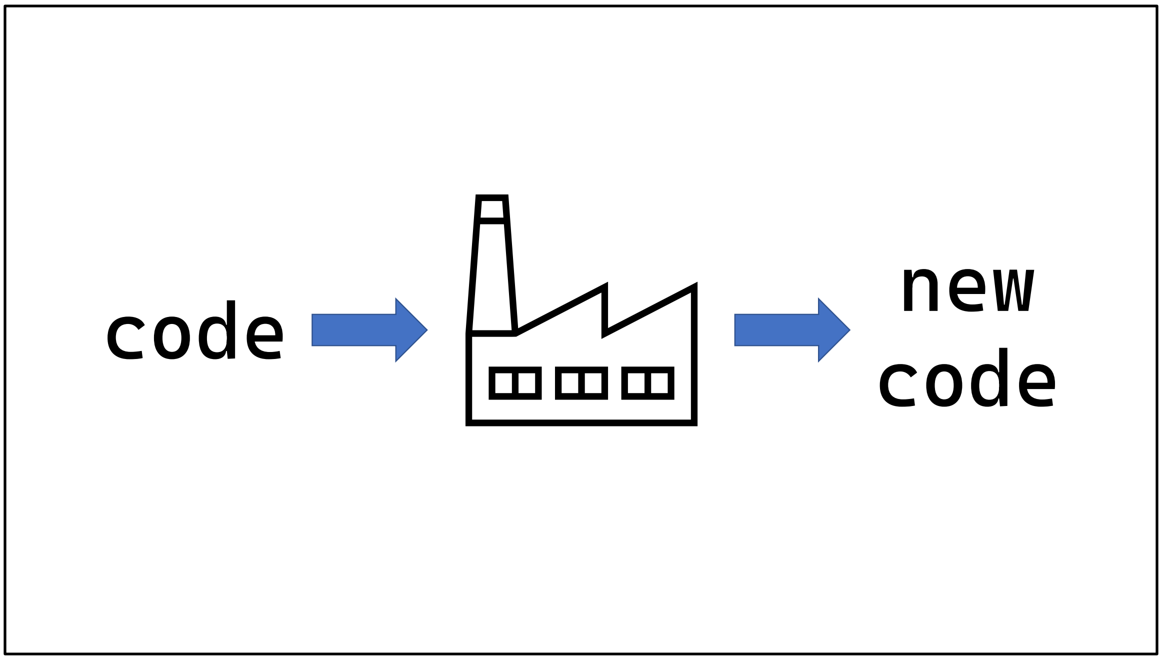


What?

So, what are source generators?

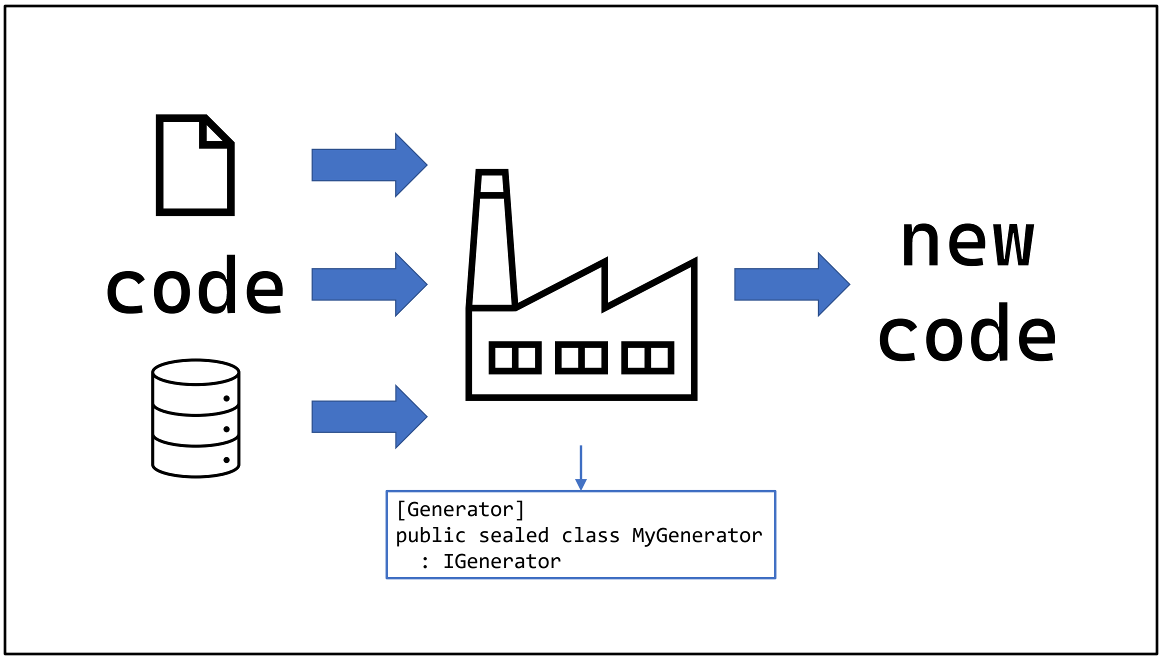
Think of a factory. You have a bunch of raw material, or pre-fabricated parts, and there’s a process in place that takes all those assets to create something at the end of the line.

https://unsplash.com/photos/QMjCzOGeglA



That’s kind of like what source generators are. They’re basically factories that create code. That’s it. Typically, you’re going to look at code that already exists, and generate new code based on what you see.

Now, I’m being explicit here in that “code” cannot be changed. You’ll create new code, but you can’t modify the existing code. So, at least for right now in 2021, we don’t get full metaprogramming with source generators. Don’t let that discourage you though, there’s a lot of powerful things you can do with source generators as you will see.



```
[Generator]
public sealed class MyGenerator
  : IGenerator
```

Note that you don’t have to just look at code. You can use CSV files, databases….anything is really permissible because the factory is C# code. Whatever you can do in a NS 2.0 assembly, you can do in a source generator. Though, keep in mind that this will be (in all likelihood) used with a tool like VS or Rider, so you want them to be as fast as possible.

Why?

OK, great, but….why would I want to use them?

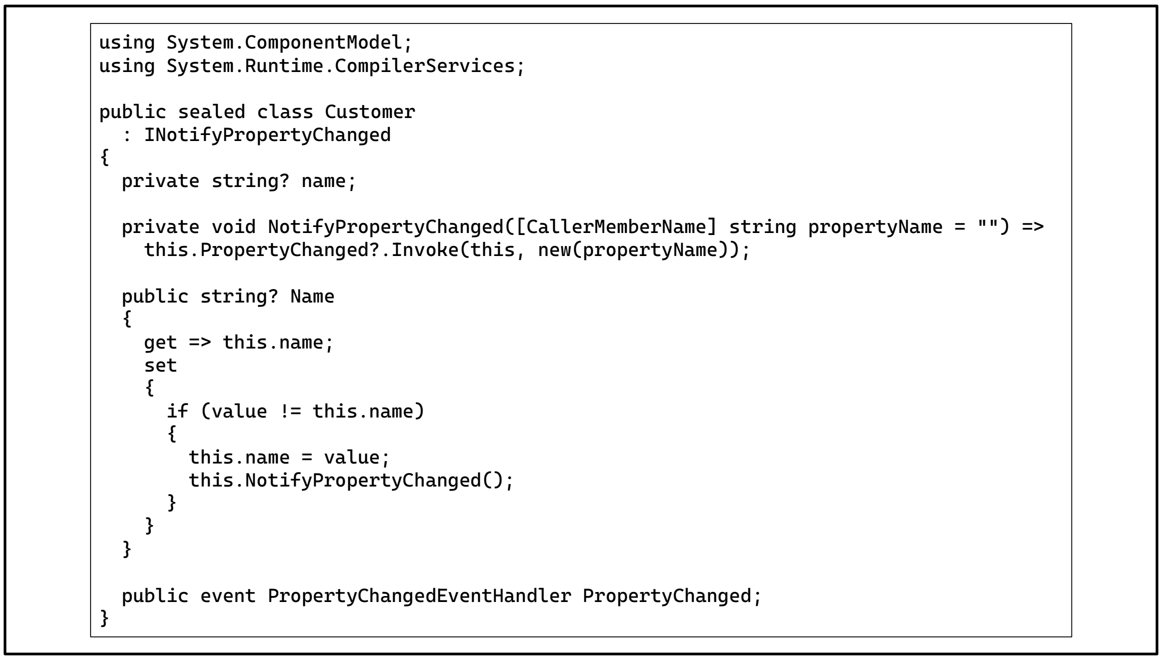
Two reasons:

Improve performance – Think of times where you wanted to make generalize code to handle any scenario. It’s not uncommon to resort to Reflection as a tool to solve the problem at hand. While Reflection is powerful, it can slow down execution time. Source generators can create the optimal path, which will be compiled into the target assembly. (Side effect is that it also makes it easy to debug the code)

https://www.pexels.com/photo/blurred-motion-of-illuminated-railroad-station-in-city-253647/

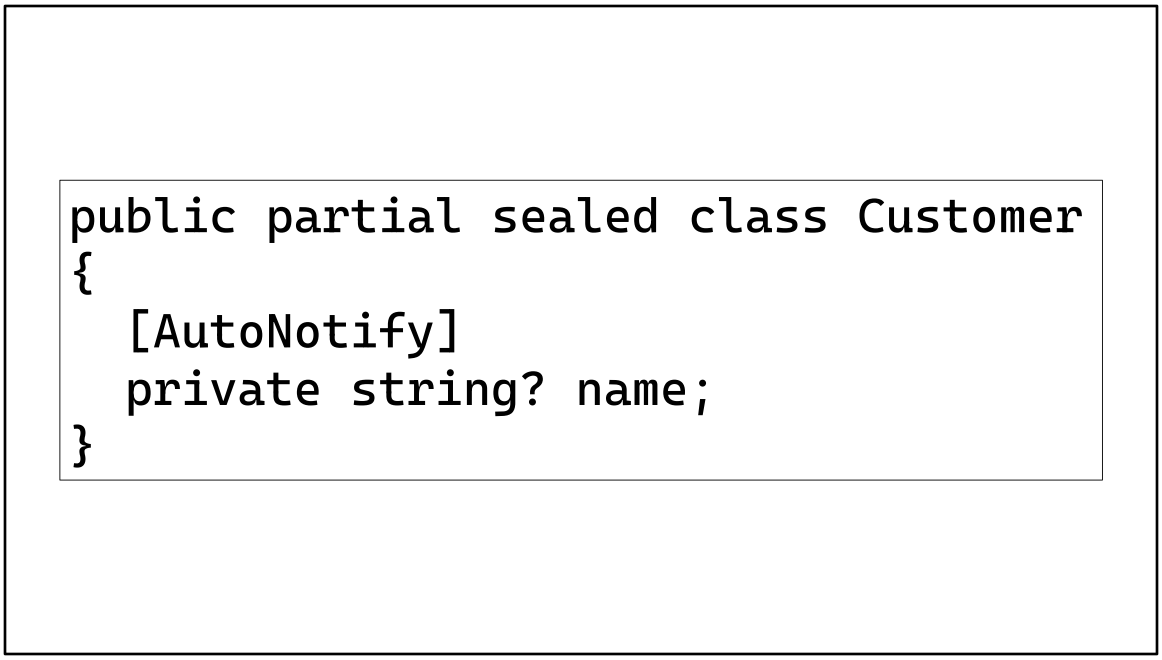
Eliminate repetitive tasks – Think of INotifyPropertyChanged. It’s an easy pattern to implement, but it’s boring, repetitive, and prone to error. Being able to generate code that implements the interface the same way every single time takes one task .

https://unsplash.com/photos/7YUvAUbfSV0



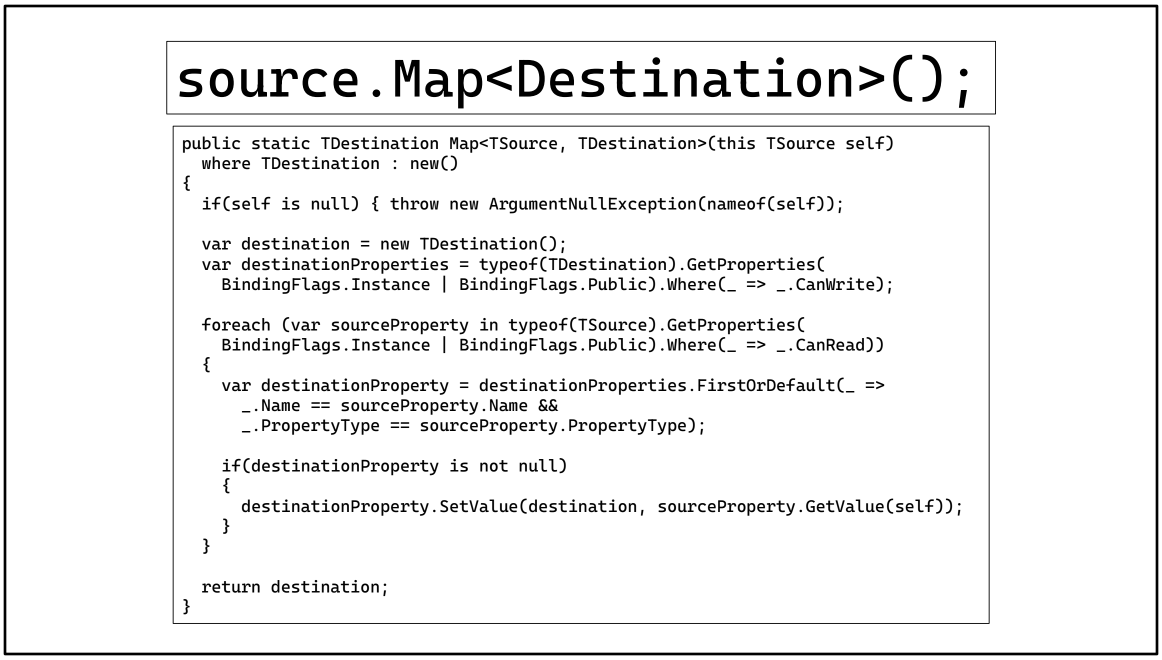
```
using System.ComponentModel;
using System.Runtime.CompilerServices;

public sealed class Customer
  : INotifyPropertyChanged
{
  private string? name;

  private void NotifyPropertyChanged([CallerMemberName] string propertyName = "") =>
    this.PropertyChanged?.Invoke(this, new(propertyName));

  public string? Name
  {
    get => this.name;
    set
    {
      if (value != this.name)
      {
        this.name = value;
        this.NotifyPropertyChanged();
      }
    }
  }

  public event PropertyChangedEventHandler PropertyChanged;
}
```

For repetitive tasks, think of INotifyPropertyChanged. Can you even read this? There's a lot of code here just to notify a listener that a property value has changed.



```
public partial sealed class Customer
{
  [AutoNotify]
  private string? name;
}
```

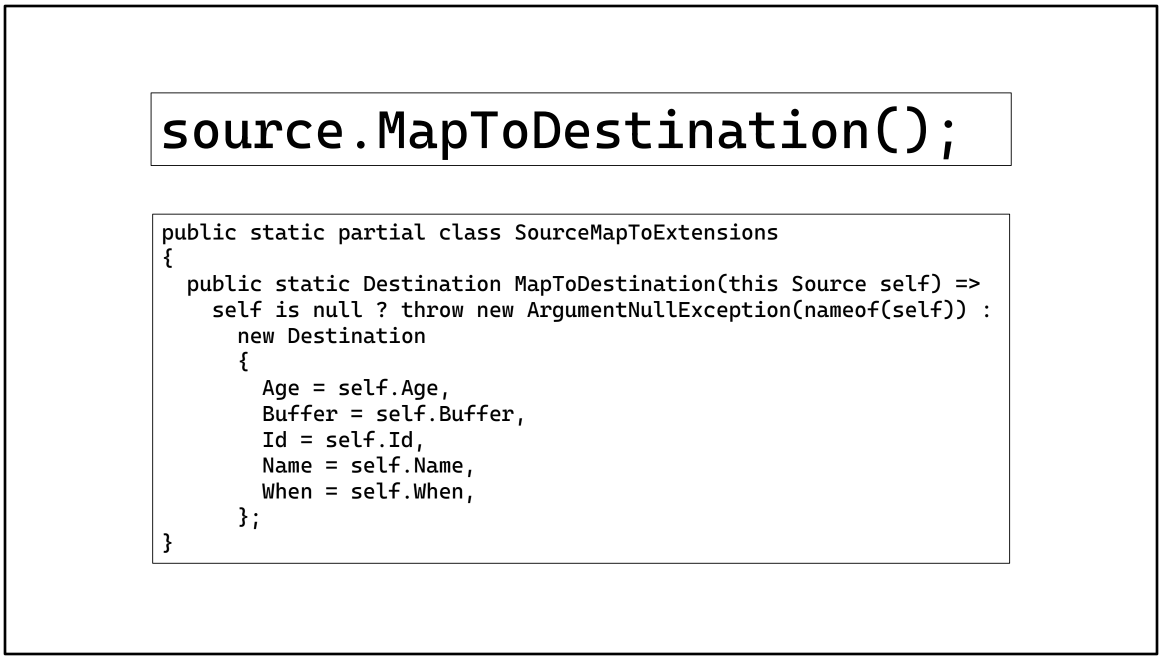
Wouldn’t you rather write this? That’s what a source generator can do. All of that INotifyPropertyChanged boilerplate code is generated into another partial class. You just need to mark a field with the attribute, and that’s it.



```
source.Map<Destination>();
```

```
public static TDestination Map<TSource, TDestination>(this TSource self)
  where TDestination : new()
{
  if(self is null) { throw new ArgumentNullException(nameof(self));

  var destination = new TDestination();
  var destinationProperties = typeof(TDestination).GetProperties(
    BindingFlags.Instance | BindingFlags.Public).Where(_ => _.CanWrite);

  foreach (var sourceProperty in typeof(TSource).GetProperties(
    BindingFlags.Instance | BindingFlags.Public).Where(_ => _.CanRead))
  {
    var destinationProperty = destinationProperties.FirstOrDefault(_ =>
      _.Name == sourceProperty.Name &&
      _.PropertyType == sourceProperty.PropertyType);

    if(destinationProperty is not null)
    {
      destinationProperty.SetValue(destination, sourceProperty.GetValue(self));
    }
  }

  return destination;
}
```

What about performance? Let’s say you wrote something that maps objects. (Forget about packages like AutoMapper for a bit). You’d probably use something like Reflection to figure out this mapping in a generic way. However, Reflection has a performance cost associated with it. You can limit it with caching, using compiled expression trees, etc., but…



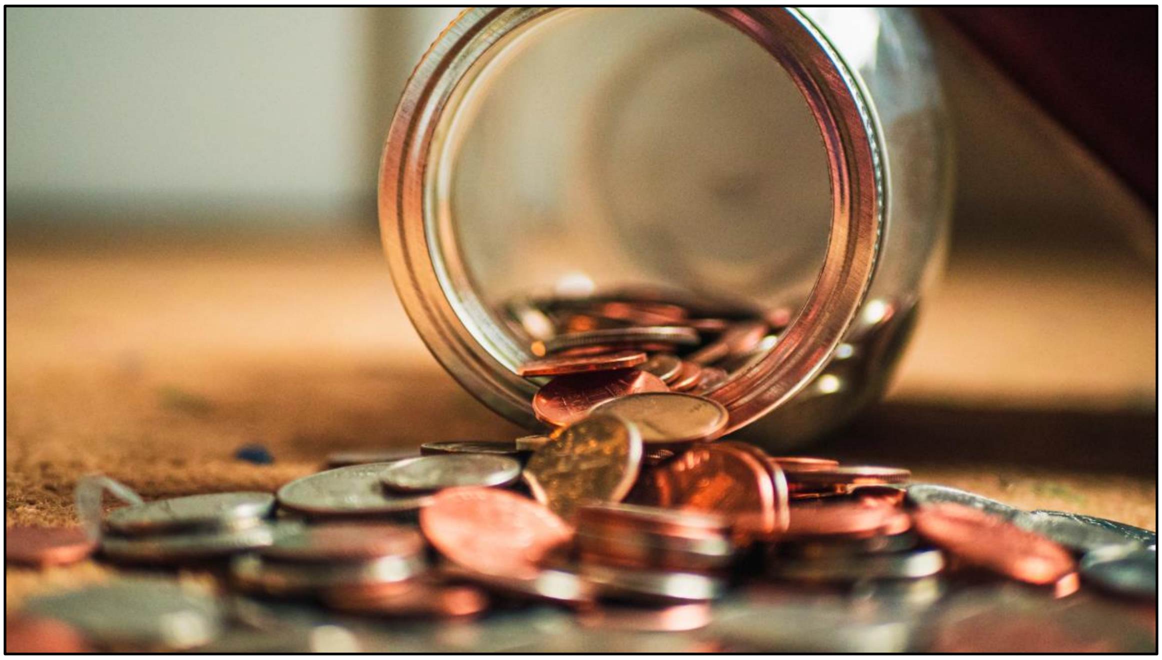
```
source.MapToDestination();
```

```
public static partial class SourceMapToExtensions
{
  public static Destination MapToDestination(this Source self) =>
    self is null ? throw new ArgumentNullException(nameof(self)) :
      new Destination
      {
        Age = self.Age,
        Buffer = self.Buffer,
        Id = self.Id,
        Name = self.Name,
        When = self.When,
      };
}
```

Wouldn’t you rather write this? All you do is use an extension method generated for you that figures out the optimal mapping path.

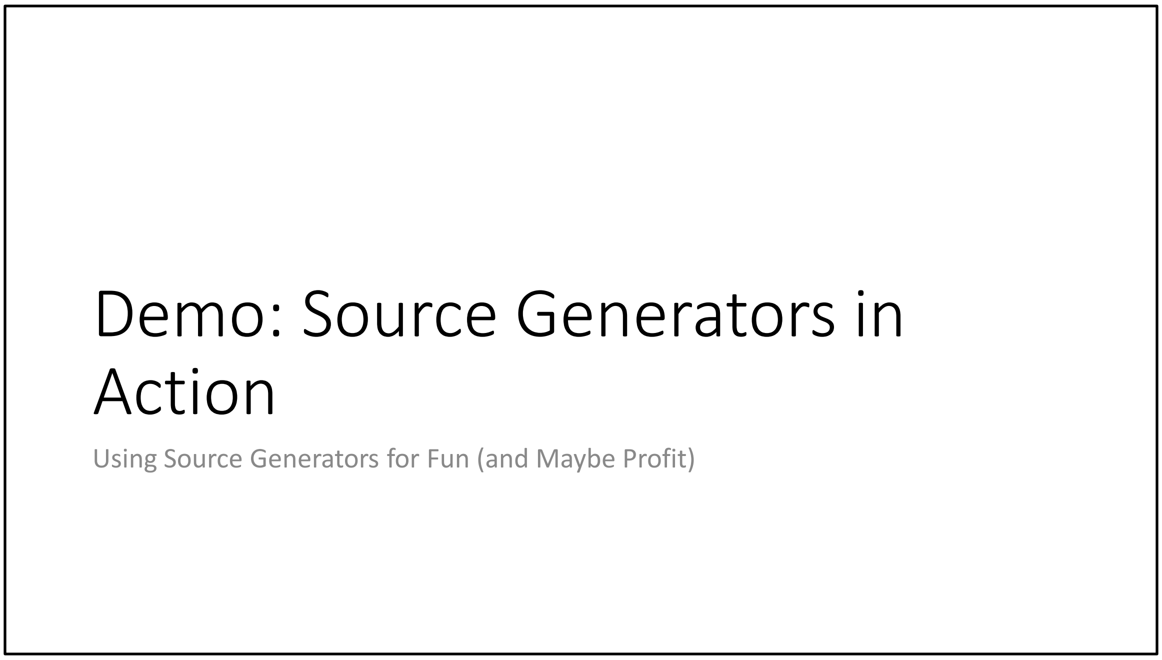
Now, you have to invest time studying the Compiler API. As you'll see in the demos, this isn't trivial.

https://www.pexels.com/photo/people-at-library-sitting-down-at-tables-757855/



But, if you’re willing to make that investment, your “profit” is less time writing and executing code.

https://unsplash.com/photos/NeTPASr-bmQ



Demo: Source Generators in Action

Using Source Generators for Fun (and Maybe Profit)

Start with InlineMapping

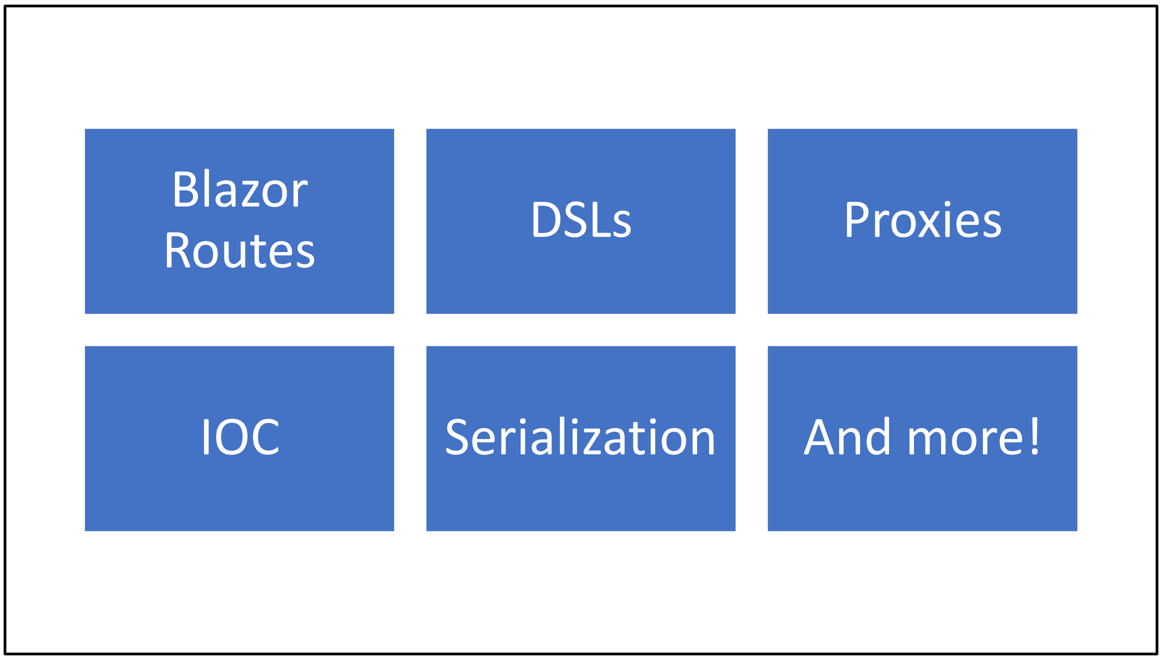
Then do PartiallyApplied

Finally, show Rocks

Call To Action

The next step for you is think of areas in your code where source generators may be useful.

https://www.pexels.com/photo/grayscale-photo-of-woman-facing-macbook-1181257/



Blazor Routes

DSLs

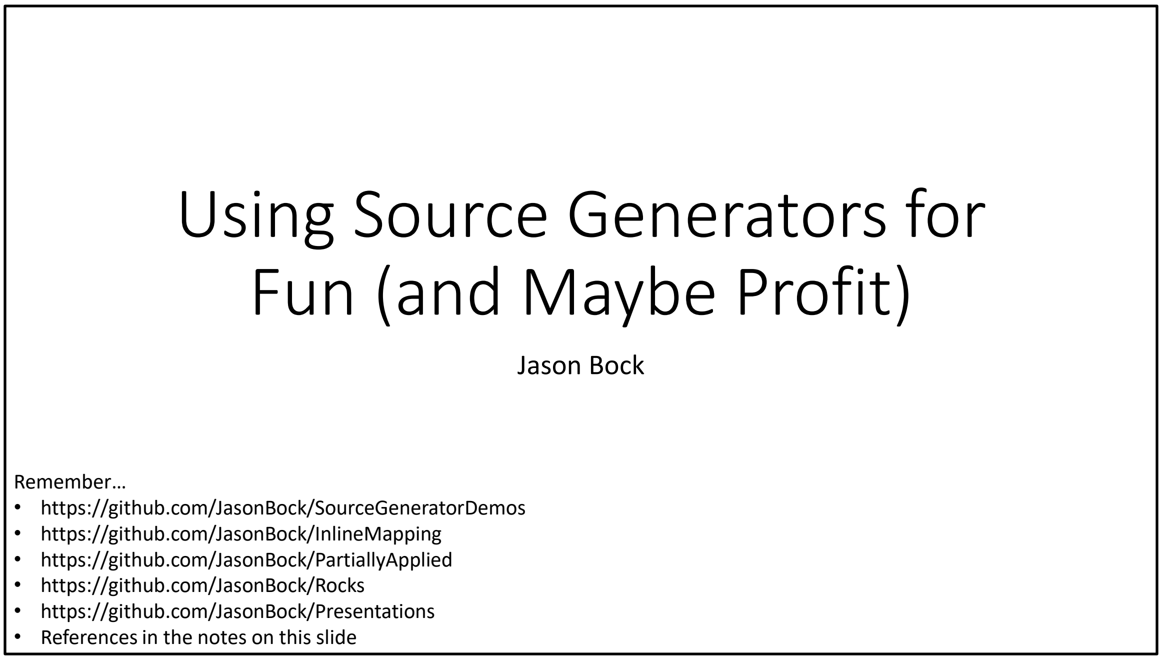
Proxies

IOC

Serialization

And more!

Here are some examples (see the notes section on the last slide for links) that I've seen that are really cool. Add to the list with your own ideas!



Using Source Generators for Fun (and Maybe Profit)

Jason Bock

Remember…
- https://github.com/JasonBock/SourceGeneratorDemos
- https://github.com/JasonBock/InlineMapping
- https://github.com/JasonBock/PartiallyApplied
- https://github.com/JasonBock/Rocks
- https://github.com/JasonBock/Presentations
- References in the notes on this slide

## References

* [Introducing C# Source Generators](https://devblogs.microsoft.com/dotnet/introducing-c-source-generators/)
* [Generating Code in C#](https://medium.com/rocket-mortgage-technology-blog/generating-code-in-c-1868ebbe52c5)
* [Debugging C# Source Generators with Visual Studio 2019 16.10](https://stevetalkscode.co.uk/debug-source-generators-with-vs2019-1610)
* [New C# Source Generator Samples](https://devblogs.microsoft.com/dotnet/new-c-source-generator-samples/)
* [Source Generators Cookbook](https://github.com/dotnet/roslyn/blob/master/docs/features/source-generators.cookbook.md)
* [C# Source Generators - Write Code that Writes Code](https://www.youtube.com/watch?v=3YwwdoRg2F4)
* [Source Generators in .NET 5 with

ReSharper](https://blog.jetbrains.com/dotnet/2020/11/12/source-generators-in-net-5-with-resharper/)
* [.NET 5 Source Generators - MediatR - CQRS - OMG!](https://www.edument.se/en/blog/post/net-5-source-generators-mediatr-cqrs)
* [A list of C# Source Generators](https://github.com/amis92/csharp-source-generators)
* [Using C# Source Generators to create an external DSL](https://devblogs.microsoft.com/dotnet/using-c-source-generators-to-create-an-external-dsl/)
* [Using source generators to find all routable components in a Blazor WebAssembly app](https://andrewlock.net/using-source-generators-to-find-all-routable-components-in-a-webassembly-app/)
* [Persisting output files from source generators](https://til.cazzulino.com/dotnet/persisting-output-files-from-source-generators)
* [GETTING STARTED WITH THE ROSLYN APIS: WRITING CODE WITH CODE](https://www.stevejgordon.co.uk/getting-started-with-the-roslyn-apis-writing-code-with-code)
* [C# 9 records as strongly-typed ids - Part 5: final bits and conclusion](https://thomaslevesque.com/2021/03/19/csharp-9-records-as-strongly-typed-ids-part-5-final-bits-and-conclusion/)
* [Consider using a source generator to generate scoped css files #30841](https://github.com/dotnet/aspnetcore/issues/30841)
* [C# Source Generators](https://www.youtube.com/watch?v=cB66gOHConw)
* [Scribian](https://github.com/scriban/scriban)
* [Channel 9: Source Generators](https://github.com/jaredpar/channel9-source-generators) - `[AutoEquality]`
* [Source Generators - real world example](https://dominikjeske.github.io/source-generators/) - `<EmitCompilerGeneratedFiles>true</EmitCompilerGeneratedFiles>`
* [Caching Enum.ToString to improve performance](https://www.meziantou.net/caching-enum-tostring-to-improve-performance.htm)

### Source Generator "Futures"
* "Artifact production" - [Initial stubs for an LSIF generator that runs as part of the normal compilation pass.](https://github.com/dotnet/roslyn/pull/49046) - "artifact production" - basically a way to produce other content types than just C# code
* Dependencies and run order - somewhat complicated
* Raw literal strings - would be helpful to have C# code in strings (esp. for tests)

* "Local" SGs - "this will only ever be used in this project"
* Tooling improvements - vague, but hopefully creating and testing them will become easier
* [Testing help](https://github.com/dotnet/roslyn-sdk/pull/694)