C# 9.0: Source Generators

Basics and features

Shcherbakov Roman

@daredever

What is a Source Generator?

Basics

```
[Generator]
public class MyGenerator : ISourceGenerator
{
    public void Initialize(GeneratorInitializationContext context)
    {
    }

    public void Execute(GeneratorExecutionContext context)
    {
    }
}
```

What is a Source Generator?

Basics

```
[Generator]
public class MyGenerator : ISourceGenerator
{
    public void Initialize(GeneratorInitializationContext context)
    {
    }

    public void Execute(GeneratorExecutionContext context)
    {
    }
}
```

What is a Source Generator?

Basics

```
[Generator]
public class MyGenerator : ISourceGenerator
{
    public void Initialize(GeneratorInitializationContext context)
    {
    }

    public void Execute(GeneratorExecutionContext context)
    {
    }
}
```

What is a Source Generator?

Project configuration

```
<Project Sdk="Microsoft.NET.Sdk">
    <PropertyGroup>
        <TargetFramework>netstandard2.0</TargetFramework>
        <LangVersion>9</LangVersion>
    </PropertyGroup>
    <ItemGroup>
        <PackageReference
          Include="Microsoft.CodeAnalysis.CSharp"
          Version="3.8.0"
          PrivateAssets="all" />
        <PackageReference
          Include="Microsoft.CodeAnalysis.Analyzers"
          Version="3.3.2"
          PrivateAssets="all" />
    </ItemGroup>
</Project>
```

What is a Source Generator?

Project configuration

```
<Project Sdk="Microsoft.NET.Sdk">
    <PropertyGroup>
        <TargetFramework>netstandard2.0</TargetFramework>
        <LangVersion>9</LangVersion>
    </PropertyGroup>
    <ItemGroup>
        <PackageReference
          Include="Microsoft.CodeAnalysis.CSharp"
          Version="3.8.0"
          PrivateAssets="all" />
        <PackageReference
          Include="Microsoft.CodeAnalysis.Analyzers"
          Version="3.3.2"
          PrivateAssets="all" />
    </ItemGroup>
</Project>
```

What is a Source Generator?

Project configuration

```
<Project Sdk="Microsoft.NET.Sdk">
    <PropertyGroup>
        <TargetFramework>netstandard2.0</TargetFramework>
        <LangVersion>9</LangVersion>
    </PropertyGroup>
    <ItemGroup>
        <PackageReference
          Include="Microsoft.CodeAnalysis.CSharp"
          Version="3.8.0"
          PrivateAssets="all" />
        <PackageReference
          Include="Microsoft.CodeAnalysis.Analyzers"
          Version="3.3.2"
          PrivateAssets="all" />
    </ItemGroup>
</Project>
```

What is a Source Generator?

How to use

```
<Project Sdk="Microsoft.NET.Sdk">

  …

  <ItemGroup>
    <ProjectReference Include=“..\Generators\Generators.csproj">
        <OutputItemType>Analyzer</OutputItemType>
        <ReferenceOutputAssembly>False</ReferenceOutputAssembly>
    </ProjectReference>
  </ItemGroup>

</Project>
```

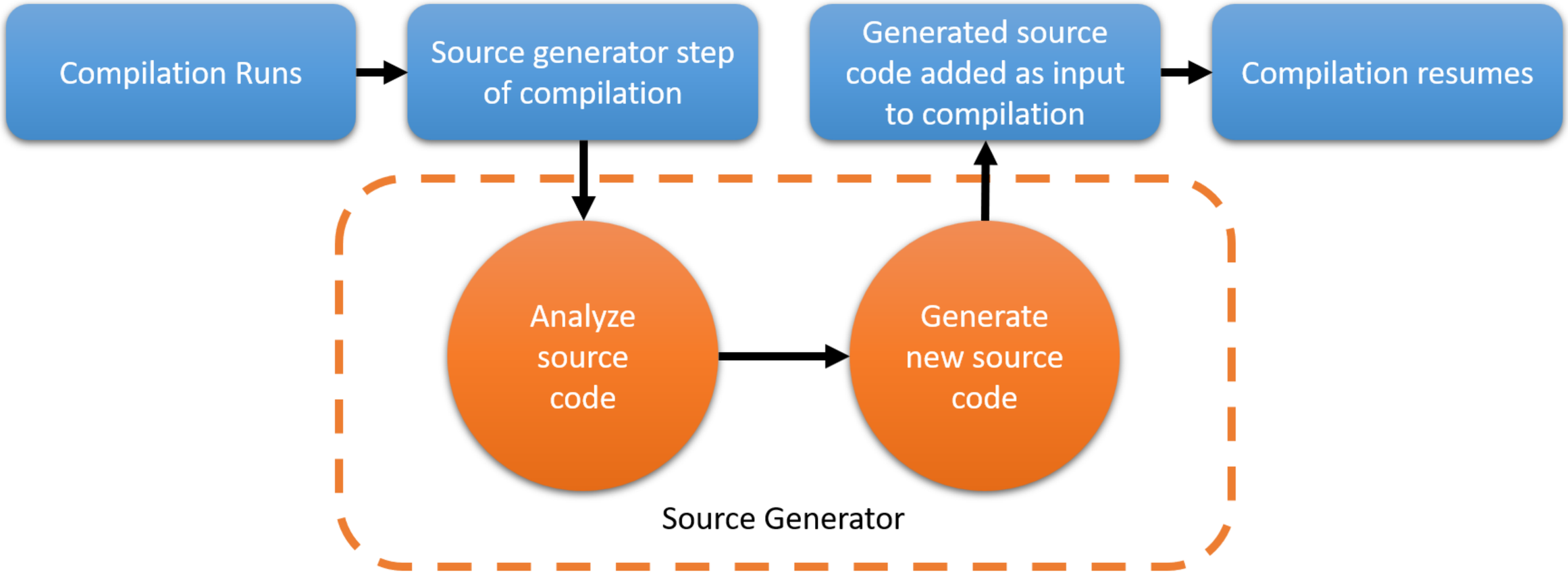
What is a Source Generator?

How to use

```
<Project Sdk="Microsoft.NET.Sdk">

  …

  <ItemGroup>
    <ProjectReference Include="..\Generators\Generators.csproj">
        <OutputItemType>Analyzer</OutputItemType>
        <ReferenceOutputAssembly>False</ReferenceOutputAssembly>
    </ProjectReference>
  </ItemGroup>

</Project>
```

What is a Source Generator?

Lifetime



Examples

Hello world

```
[Generator]
public class CustomGenerator : ISourceGenerator
{
  public void Initialize(GeneratorInitializationContext context) {}

  public void Execute(GeneratorExecutionContext context)
  {
    context.AddSource("GeneratedClass.cs", SourceText.From(@"
namespace GeneratedNamespace
{
    public class GeneratedClass
    {
        public static void GeneratedMethod()
        {
            System.Console.WriteLine(""Hello, generated, World!"");
        }
    }
}", Encoding.UTF8));
  }
}
```

Examples

Hello world

```
[Generator]
public class CustomGenerator : ISourceGenerator
{
  public void Initialize(GeneratorInitializationContext context) {}

  public void Execute(GeneratorExecutionContext context)
  {
    context.AddSource("GeneratedClass.cs", SourceText.From(@"
namespace GeneratedNamespace
{
    public class GeneratedClass
    {
        public static void GeneratedMethod()
        {
            System.Console.WriteLine(""Hello, generated, World!"");
        }
    }
}", Encoding.UTF8));
  }
}
```

Examples

Hello world

```
static void Main(string[] args)
{
    // call into a generated method
    GeneratedNamespace.GeneratedClass.GeneratedMethod();
}
```

Examples

Advanced

Countries.xml (https://www.artlebedev.ru/country-list/)

```
<?xml version="1.0" encoding="UTF-8" ?>
<country-list>
    <country>
        <name>Россия</name>
        <fullname>Российская Федерация</fullname>
        <english>Russian Federation</english>
        <alpha2>RU</alpha2>
        <alpha3>RUS</alpha3>
        <iso>643</iso>
        <location>Европа</location>
        <location-precise>Восточная Европа</location-precise>
    </country>
</country-list>
```

Examples

Advanced

Countries.xml (https://www.artlebedev.ru/country-list/)

```
<?xml version="1.0" encoding="UTF-8" ?>
<country-list>
    <country>
        <name>Россия</name>
        <fullname>Российская Федерация</fullname>
        <english>Russian Federation</english>
        <alpha2>RU</alpha2>
        <alpha3>RUS</alpha3>
        <iso>643</iso>
        <location>Европа</location>
        <location-precise>Восточная Европа</location-precise>
    </country>
</country-list>
```

Examples

Advanced

```
<Project Sdk="Microsoft.NET.Sdk">

  …

  <ItemGroup>
    <AdditionalFiles Include="Files\Countries.xml"/>
  </ItemGroup>

</Project>
```

Examples

Advanced

```
<Project Sdk="Microsoft.NET.Sdk">

  …

  <ItemGroup>
    <AdditionalFiles Include="Files\Countries.xml"/>
  </ItemGroup>

</Project>
```

Examples

Advanced

```
public partial class DirectoryOfCountries
{
    public static partial IReadOnlyList<Country> Europe();
}
```

Examples

Advanced

```
public partial class DirectoryOfCountries
{
    public static partial IReadOnlyList<Country> Europe();
}
```

Examples

Advanced

```
private class CustomSyntaxReceiver : ISyntaxReceiver
{
  public ClassDeclarationSyntax ClassToAugment { get; private set; }

  public void OnVisitSyntaxNode(SyntaxNode syntaxNode)
  {
    // Business logic to decide what we're interested in goes here
    if (syntaxNode is ClassDeclarationSyntax cds &&
        cds.Identifier.ValueText == "DirectoryOfCountries")
    {
      ClassToAugment = cds;
    }
  }
}
```

Examples

Advanced

```
private class CustomSyntaxReceiver : ISyntaxReceiver
{
  public ClassDeclarationSyntax ClassToAugment { get; private set; }

  public void OnVisitSyntaxNode(SyntaxNode syntaxNode)
  {
    // Business logic to decide what we're interested in goes here
    if (syntaxNode is ClassDeclarationSyntax cds &&
        cds.Identifier.ValueText == "DirectoryOfCountries")
    {
      ClassToAugment = cds;
    }
  }
}
```

Examples

Advanced

```
private class CustomSyntaxReceiver : ISyntaxReceiver
{
  public ClassDeclarationSyntax ClassToAugment { get; private set; }

  public void OnVisitSyntaxNode(SyntaxNode syntaxNode)
  {
    // Business logic to decide what we're interested in goes here
    if (syntaxNode is ClassDeclarationSyntax cds &&
        cds.Identifier.ValueText == "DirectoryOfCountries")
    {
      ClassToAugment = cds;
    }
  }
}
```

Examples

Advanced

```
public void Initialize(GeneratorInitializationContext context)
{
   // Register a factory that can
   // create our custom syntax receiver
   context.RegisterForSyntaxNotifications(
      () => new CustomSyntaxReceiver());
}
```

Examples

Advanced

```
public void Initialize(GeneratorInitializationContext context)
{
   // Register a factory that can
   // create our custom syntax receiver
   context.RegisterForSyntaxNotifications(
      () => new CustomSyntaxReceiver());
}
```

Examples

Advanced

```
public void Execute(GeneratorExecutionContext context)
{
  var receiver = (CustomSyntaxReceiver)context.SyntaxReceiver;
  if (receiver.ClassToAugment is null) return;

  var xmlData = context.AdditionalFiles
              .SingleOrDefault(a => Path.GetFileName(a.Path) == "Countries.xml")
              ?.GetText(context.CancellationToken)?.ToString();
  if (xmlData is null)
  {
     ReportDiagnostic(context);
    return;
  }

  AddSource(context, xmlData);
}
```

Examples

Advanced

```
public void Execute(GeneratorExecutionContext context)
{
  var receiver = (CustomSyntaxReceiver)context.SyntaxReceiver;
  if (receiver.ClassToAugment is null) return;

  var xmlData = context.AdditionalFiles
              .SingleOrDefault(a => Path.GetFileName(a.Path) == "Countries.xml")
              ?.GetText(context.CancellationToken)?.ToString();
  if (xmlData is null)
  {
     ReportDiagnostic(context);
    return;
  }

  AddSource(context, xmlData);
}
```

Examples

Advanced

```
public void Execute(GeneratorExecutionContext context)
{
  var receiver = (CustomSyntaxReceiver)context.SyntaxReceiver;
  if (receiver.ClassToAugment is null) return;

  var xmlData = context.AdditionalFiles
              .SingleOrDefault(a => Path.GetFileName(a.Path) == "Countries.xml")
              ?.GetText(context.CancellationToken)?.ToString();
  if (xmlData is null)
  {
     ReportDiagnostic(context);
    return;
  }

  AddSource(context, xmlData);
}
```

Examples

Advanced

```
public void Execute(GeneratorExecutionContext context)
{
  var receiver = (CustomSyntaxReceiver)context.SyntaxReceiver;
  if (receiver.ClassToAugment is null) return;

  var xmlData = context.AdditionalFiles
              .SingleOrDefault(a => Path.GetFileName(a.Path) == "Countries.xml")
              ?.GetText(context.CancellationToken)?.ToString();
  if (xmlData is null)
  {
     ReportDiagnostic(context);
    return;
  }

  AddSource(context, xmlData);
}
```

Examples

Advanced

```
public void Execute(GeneratorExecutionContext context)
{
  var receiver = (CustomSyntaxReceiver)context.SyntaxReceiver;
  if (receiver.ClassToAugment is null) return;

  var xmlData = context.AdditionalFiles
              .SingleOrDefault(a => Path.GetFileName(a.Path) == "Countries.xml")
              ?.GetText(context.CancellationToken)?.ToString();
  if (xmlData is null)
  {
     ReportDiagnostic(context);
    return;
  }

  AddSource(context, xmlData);
}
```

Examples

Advanced

```
private static void ReportDiagnostic(GeneratorExecutionContext context)
{
  var error = new DiagnosticDescriptor(
                  id: "AGEN001",
                  title: "Could find XML file",
                  messageFormat: "Could find XML file '{0}'",
                  category: "AugmentingGenerator",
                  defaultSeverity: DiagnosticSeverity.Error,
                  isEnabledByDefault: true);
  var diagnostic = Diagnostic.Create(
             error, Location.None, "Countries.xml");
  context.ReportDiagnostic(diagnostic);
}
```

Examples

Advanced

```
private static void ReportDiagnostic(GeneratorExecutionContext context)
{
  var error = new DiagnosticDescriptor(
                id: "AGEN001",
                title: "Could find XML file",
                messageFormat: "Could find XML file '{0}'",
                category: "AugmentingGenerator",
                defaultSeverity: DiagnosticSeverity.Error,
                isEnabledByDefault: true);
  var diagnostic = Diagnostic.Create(
           error, Location.None, "Countries.xml");
  context.ReportDiagnostic(diagnostic);
}
```

Examples

Advanced

```
private static void ReportDiagnostic(GeneratorExecutionContext context)
{
  var error = new DiagnosticDescriptor(
                  id: "AGEN001",
                  title: "Could find XML file",
                  messageFormat: "Could find XML file '{0}'",
                  category: "AugmentingGenerator",
                  defaultSeverity: DiagnosticSeverity.Error,
                  isEnabledByDefault: true);
  var diagnostic = Diagnostic.Create(
              error, Location.None, "Countries.xml");
  context.ReportDiagnostic(diagnostic);
}
```

Examples

Advanced

```
const string CountrySource = @“
namespace Countries { public record Country (string Name, string Code); }";

private static void AddSource(
  GeneratorExecutionContext context,
  string xmlData)
{
    context.AddSource(
        "Country.cs",
        SourceText.From(CountrySource, Encoding.UTF8));

    var generatedSource = GenerateSource(xmlData);
    context.AddSource(
        "DirectoryOfCountries.Generated.cs",
        SourceText.From(generatedSource, Encoding.UTF8));
}
```

Examples

Advanced

```
const string CountrySource = @“
namespace Countries { public record Country (string Name, string Code); }";

private static void AddSource(
  GeneratorExecutionContext context,
  string xmlData)
{
    context.AddSource(
        "Country.cs",
        SourceText.From(CountrySource, Encoding.UTF8));

    var generatedSource = GenerateSource(xmlData);
    context.AddSource(
        "DirectoryOfCountries.Generated.cs",
        SourceText.From(generatedSource, Encoding.UTF8));
}
```

Examples

Advanced

```
const string CountrySource = @“
namespace Countries { public record Country (string Name, string Code); }";

private static void AddSource(
  GeneratorExecutionContext context,
  string xmlData)
{
    context.AddSource(
        "Country.cs",
        SourceText.From(CountrySource, Encoding.UTF8));

    var generatedSource = GenerateSource(xmlData);
    context.AddSource(
        "DirectoryOfCountries.Generated.cs",
        SourceText.From(generatedSource, Encoding.UTF8));
}
```

Examples

Advanced

```
var sb = new StringBuilder();
sb.AppendLine(@“
using System.Collections.Generic;
namespace Countries {
  public partial class DirectoryOfCountries {
    private static List<Country> _europe = new List<Country> {“);

foreach (var country in countries.Where(c => c.Location == "Европа"))
{
    sb.Append($"new Country (\"{country.Name}\", \"{country.Code}\"),");
}

sb.AppendLine(@“
    };
    public static partial IReadOnlyList<Country> Europe() => _europe;
  }}");
```

Examples

Advanced

```
var sb = new StringBuilder();
sb.AppendLine(@“
using System.Collections.Generic;
namespace Countries {
 public partial class DirectoryOfCountries {
   private static List<Country> _europe = new List<Country> {“);

foreach (var country in countries.Where(c => c.Location == "Европа"))
{
   sb.Append($"new Country (\"{country.Name}\", \"{country.Code}\"),");
}

sb.AppendLine(@“
   };
   public static partial IReadOnlyList<Country> Europe() => _europe;
  }}");
```

Examples

Advanced

```
var sb = new StringBuilder();
sb.AppendLine(@“
using System.Collections.Generic;
namespace Countries {
  public partial class DirectoryOfCountries {
   private static List<Country> _europe = new List<Country> {“);

foreach (var country in countries.Where(c => c.Location == "Европа"))
{
    sb.Append($"new Country (\"{country.Name}\", \"{country.Code}\"),");
}

sb.AppendLine(@“
    };
    public static partial IReadOnlyList<Country> Europe() => _europe;
  }}");
```

Examples

Advanced

```
var sb = new StringBuilder();
sb.AppendLine(@“
using System.Collections.Generic;
namespace Countries {
  public partial class DirectoryOfCountries {
    private static List<Country> _europe = new List<Country> {“);

foreach (var country in countries.Where(c => c.Location == "Европа"))
{
    sb.Append($"new Country (\"{country.Name}\", \"{country.Code}\"),");
}

sb.AppendLine(@“
    };
    public static partial IReadOnlyList<Country> Europe() => _europe;
  }}");
```

Examples

Advanced

```
var sb = new StringBuilder();
sb.AppendLine(@“
using System.Collections.Generic;
namespace Countries {
  public partial class DirectoryOfCountries {
    private static List<Country> _europe = new List<Country> {“);

foreach (var country in countries.Where(c => c.Location == "Европа"))
{
    sb.Append($"new Country (\"{country.Name}\", \"{country.Code}\"),");
}

sb.AppendLine(@“
    };
    public static partial IReadOnlyList<Country> Europe() => _europe;
  }}”);
```

Debugging

View generated code

```
<Project Sdk="Microsoft.NET.Sdk">

  …

  <PropertyGroup>
    <EmitCompilerGeneratedFiles>true</EmitCompilerGeneratedFiles>
    <CompilerGeneratedFilesOutputPath>
      $(BaseIntermediateOutputPath)\GeneratedFiles
    </CompilerGeneratedFilesOutputPath>
  </PropertyGroup>

</Project>
```

Debugging

View generated code

```
<Project Sdk="Microsoft.NET.Sdk">

  …

  <PropertyGroup>
    <EmitCompilerGeneratedFiles>true</EmitCompilerGeneratedFiles>
    <CompilerGeneratedFilesOutputPath>
      $(BaseIntermediateOutputPath)\GeneratedFiles
    </CompilerGeneratedFilesOutputPath>
  </PropertyGroup>

</Project>
```

Debugging

View generated code

```
<Project Sdk="Microsoft.NET.Sdk">

  …

  <PropertyGroup>
    <EmitCompilerGeneratedFiles>true</EmitCompilerGeneratedFiles>
    <CompilerGeneratedFilesOutputPath>
      $(BaseIntermediateOutputPath)\GeneratedFiles
    </CompilerGeneratedFilesOutputPath>
  </PropertyGroup>

</Project>
```

Debugging

Debug generated code

GeneratedClass.cs [generated]

This file is auto-generated by the generator 'Generators.CustomGenerator' and cannot be edited.

Miscellaneous Files | GeneratedNamespace.GeneratedClass | GeneratedMethod()

```
1
2   namespace GeneratedNamespace
3   {
4       public class GeneratedClass
5       {
6           public static void GeneratedMethod()
7           {
8               System.Console.WriteLine("Hello, generated, World!");
9           }
10      }
11  }
```

Debugging

Debug a source generator

CustomGenerator.cs

Generators | Generators.CustomGenerator

```
4
5   namespace Generators
6   {
7       [Generator]
        0 references
8       public class CustomGenerator : ISourceGenerator
9       {
            3 references
10          public void Initialize(GeneratorInitializationContext context)
11          {
12          }
13
            3 references
14          public void Execute(GeneratorExecutionContext context)
15          {
16              System.Diagnostics.Debugger.Launch();
```

Unit testing

Prepare code model

```
private static Compilation CreateCompilation(string source)
      => CSharpCompilation.Create("compilation",
        new[] {CSharpSyntaxTree.ParseText(source)},
        new[] {MetadataReference.CreateFromFile(
               typeof(Binder).GetTypeInfo().Assembly.Location)},
               new CSharpCompilationOptions(OutputKind.ConsoleApplication));

var inputCompilation = CreateCompilation(@"
namespace MyCode
{
    public class Program
    {
        public static void Main(string[] args)
        {
        }
    }
}”);
```

Unit testing

Compile

```
// Directly create an instance of the generator
var generator = new CustomGenerator();

// Create the driver that will control the
// generation, passing in our generator
GeneratorDriver driver = CSharpGeneratorDriver.Create(generator);

// Run the generation pass
driver = driver.RunGeneratorsAndUpdateCompilation(
            inputCompilation,
            out var outputCompilation,
            out var diagnostics);
```

Unit testing

Assert compilation results

```
// There were no diagnostics created by the generators
Assert.True(diagnostics.IsEmpty);

// We have two syntax trees, the original 'user'
// provided one, and the one added by the generator
Assert.True(outputCompilation.SyntaxTrees.Count() == 2);

// Verify the compilation with the added source has no diagnostics
Assert.True(outputCompilation.GetDiagnostics().IsEmpty);
```

Unit testing

Assert the compilation results directly

```
// Or we can look at the results directly:
var runResult = driver.GetRunResult();

// The runResult contains the combined results
// of all generators passed to the driver
Assert.True(runResult.GeneratedTrees.Length == 1);
Assert.True(runResult.Diagnostics.IsEmpty);

// Or you can access the individual results on a by-generator basis
var generatorResult = runResult.Results[0];
Assert.True(generatorResult.Generator == generator);
Assert.True(generatorResult.Diagnostics.IsEmpty);
Assert.True(generatorResult.GeneratedSources.Length == 1);
Assert.True(generatorResult.Exception is null);
```

Publishing as NuGet package

```
<Project Sdk="Microsoft.NET.Sdk">

  …

  <ItemGroup>
      <None
          Include="$(OutputPath)\$(AssemblyName).dll"
          Pack="true"
          PackagePath="analyzers/dotnet/cs"
          Visible="false" />
  </ItemGroup>

</Project>
```

Publishing as NuGet package

```
<Project Sdk="Microsoft.NET.Sdk">

  …

  <ItemGroup>
      <None
          Include="$(OutputPath)\$(AssemblyName).dll"
          Pack="true"
          PackagePath="analyzers/dotnet/cs"
          Visible="false" />
  </ItemGroup>

</Project>
```

Problems

- Not easy to work with Roslyn code model (AST)
- Performance affects compilation time
- Silent work and IDE support needed
- Security of third-party nuget packages

References

Docs

- https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-9#support-for-code-generator
- https://github.com/dotnet/roslyn/blob/master/docs/features/source-generators.cookbook.md
- https://devblogs.microsoft.com/dotnet/introducing-c-source-generators/
- https://blog.jetbrains.com/dotnet/2020/11/12/source-generators-in-net-5-with-resharper/
- https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/proposals/csharp-9.0/extending-partial-methods
- https://khalidabuhakmeh.com/module-initializers-in-csharp-9

References

Examples

- https://github.com/trampster/JsonSrcGen
- https://github.com/devlooped/ThisAssembly
- https://github.com/ufcpp/StringLiteralGenerator
- https://devblogs.microsoft.com/dotnet/new-c-source-generator-samples/
- https://github.com/daredever/SourceGenerators/tree/main/src