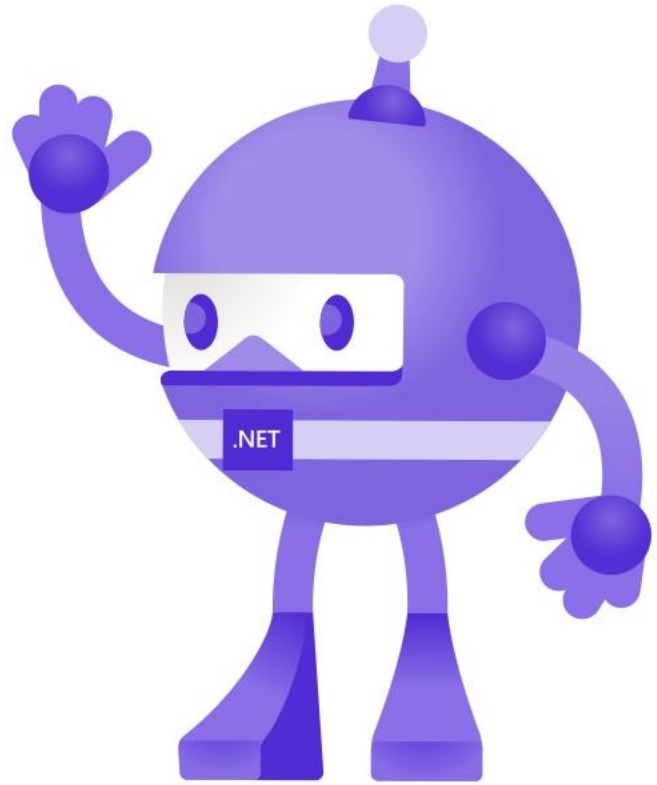
Source Generator in .NET 8: Oltre la Reflection, Verso la Compilazione AOT

Marco Bortolin



SPONSOR


Microsoft


avanade


BC Soft
consulenza sistemi informativi


randstad


UNIKEY
Bringing IT knowledge to the people


Reti
Società Benefit

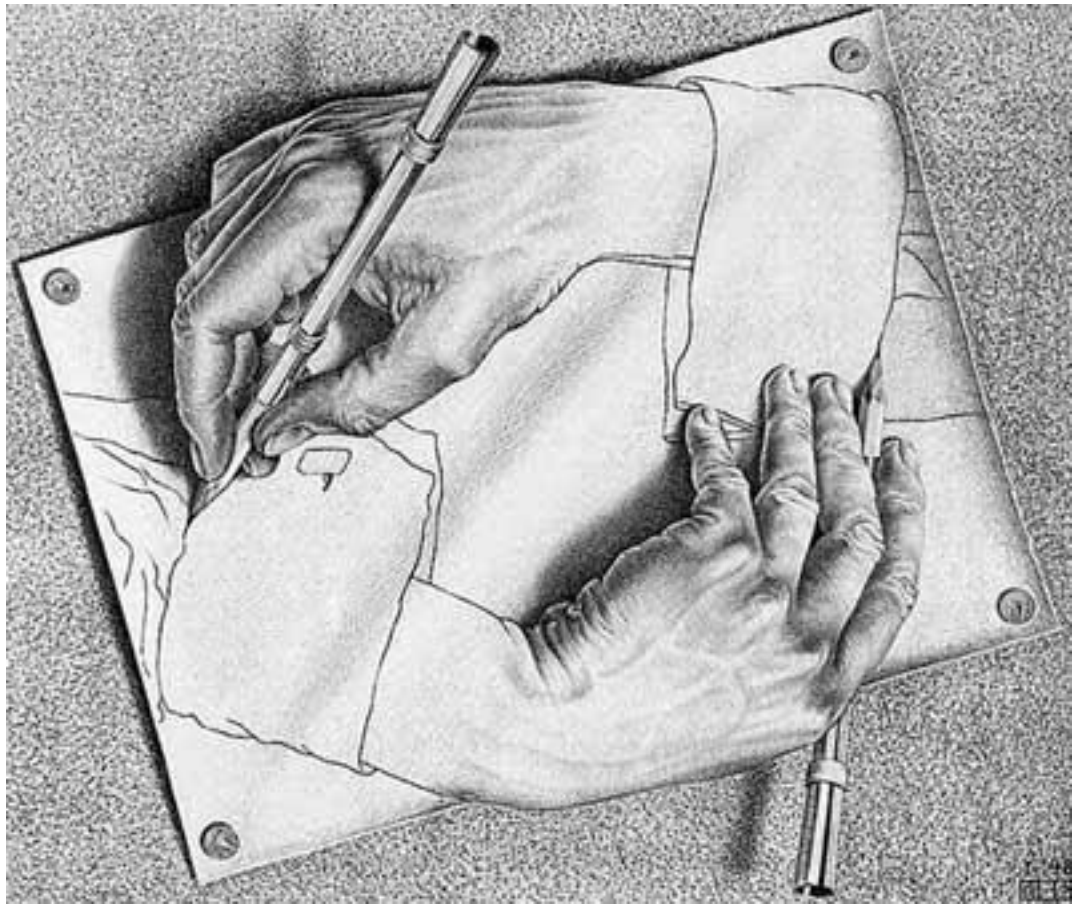

JET BRAINS


Packt


NET CODE


NET CODE

Source Generator



Escher, Mani che disegnano

"A Source Generator is a piece of code that runs during compilation and can inspect your program to produce additional source files that are compiled together with the rest of your code"

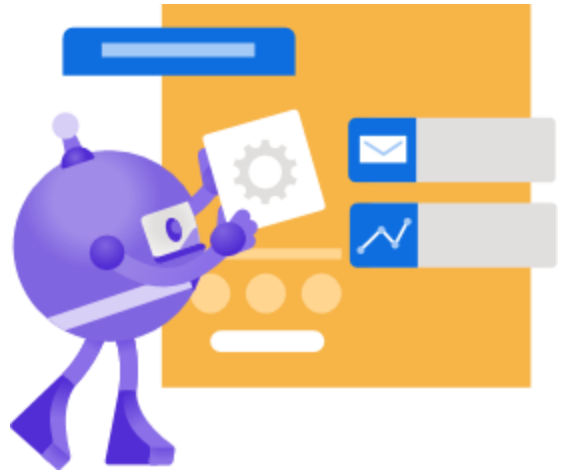
from docs.microsoft.com

Cosa sono ?

Un **Source Generator** è una componente del compilatore Roslyn che ti consente di:

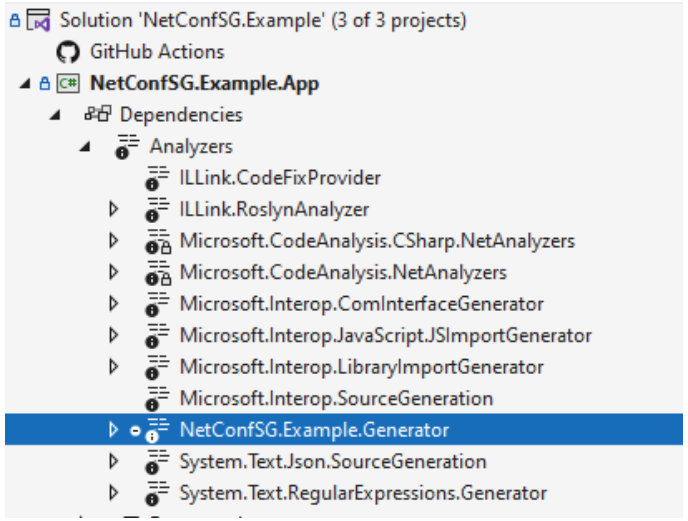
- Analizzare il codice sorgente del tuo progetto (sintassi e modelli semantici) **in fase di compilazione**, proprio come già fanno gli analizzatori di codice.
- Generare file sorgenti che possono essere aggiunti al contesto del compliatore **durante la compilazione stessa**.
- In altre parole, puoi fornire codice sorgente aggiuntivo come input per una compilazione mentre il codice viene compilato
- Puoi così spostare attività che tipicamente faresti a **runtime** tramite reflection, a **compile time.**

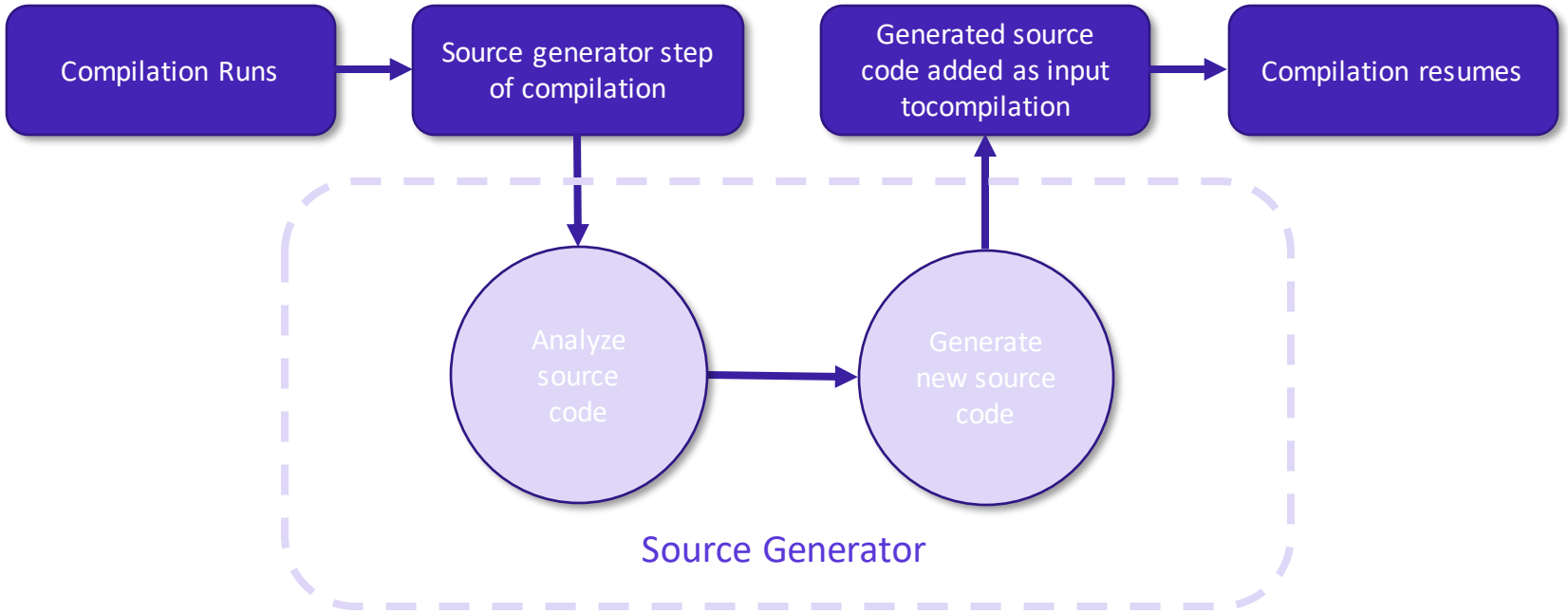
Performance



Come funzionano

Un Source Generator non è altro che un assembly **.NET Standard 2.0** caricato dal compilatore assieme agli altri analizzatori di codice sorgente.





https://devblogs.microsoft.com/dotnet/introducing-c-source-generators/

System.Text.Json & Source Generators

Da .NET 6 è possibile un nuovo approccio alla serializzazione JSON per che consiste nel spostare a compile-time l'ispezione dei tipi serializzabili JSON, tramite Source Generator.

Serialization

	Elapsed time (ms)	Allocated (KB)
Serializer	28.25	1110.00
SrcGenSerializer	12.75	563.00

Deserialization

	Elapsed time (ms)	Allocated (KB)
Serializer	24.75	1457.00
SrcGenSerializer	15.50	1025.00

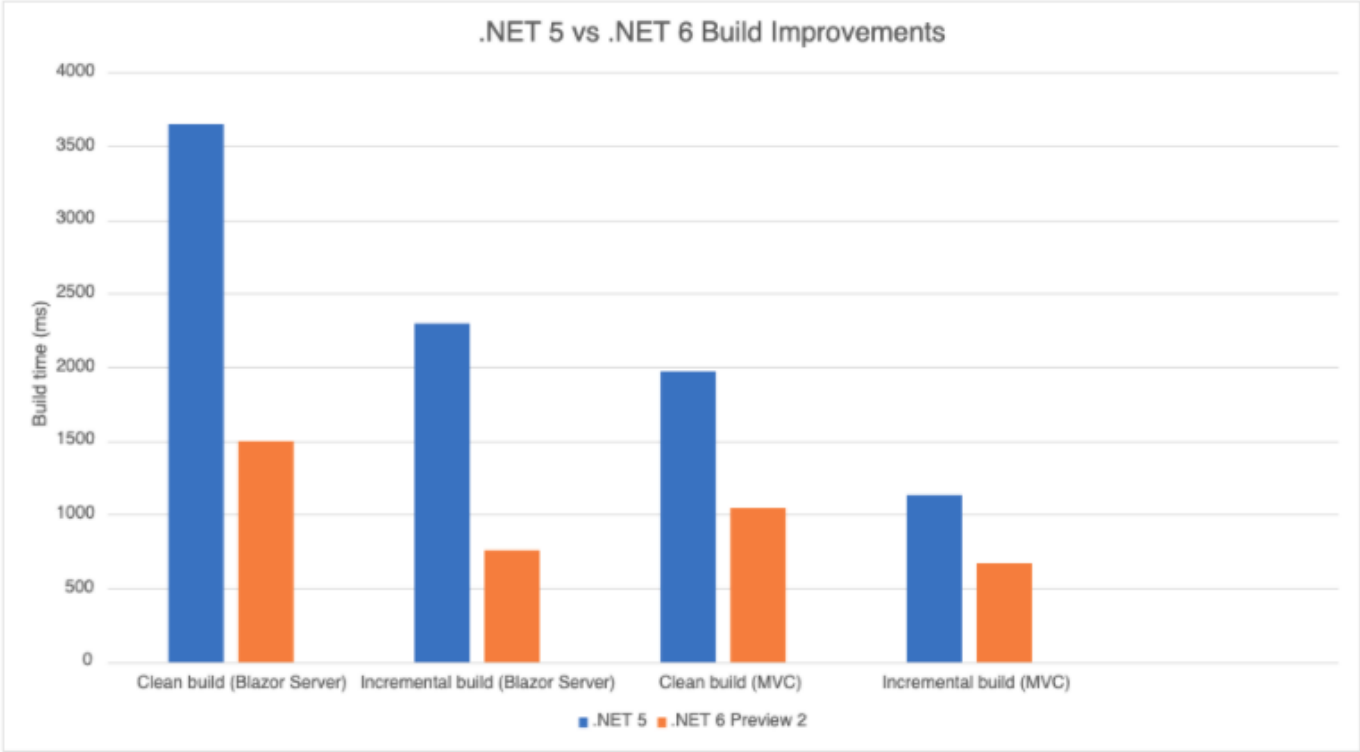
	Requests/sec	Requests
net5.0	243,000	3,669,151
net6.0	260,928	3,939,804
net6.0 + JSON source gen	364,224	5,499,468

Razor & Source Generators

Da .NET 6 il compilatore Razor è stato implementato utilizzando generatori di sorgenti C#.

L'uso dei generatori di sorgenti semplifica il compilatore Razor e accellera notevolmente i tempi di compilazione.

The following graph shows the build time improvements when using the new Razor compiler to build the default Blazor Server and MVC templates:



SG & AOT

Limitations of Native AOT deployment

Native AOT apps have the following limitations:

- No dynamic loading, for example, `Assembly.LoadFile`.
- No run-time code generation, for example, `System.Reflection.Emit`.
- No C++/CLI.
- Windows: No built-in COM.
- Requires trimming, which has limitations.
- Implies compilation into a single file, which has known incompatibilities.
- Apps include required runtime libraries (just like self-contained apps, increasing their size as compared to framework-dependent apps).
- System.Linq.Expressions always use their interpreted form, which is slower than run-time generated compiled code.
- Not all the runtime libraries are fully annotated to be Native AOT compatible. That is, some warnings in the runtime libraries aren't actionable by end developers.

Trimming: .NET SDK analizza l'intera applicazione e rimuove tutto il codice inutilizzato.

.Net 8 SDK

Compilazione AOT

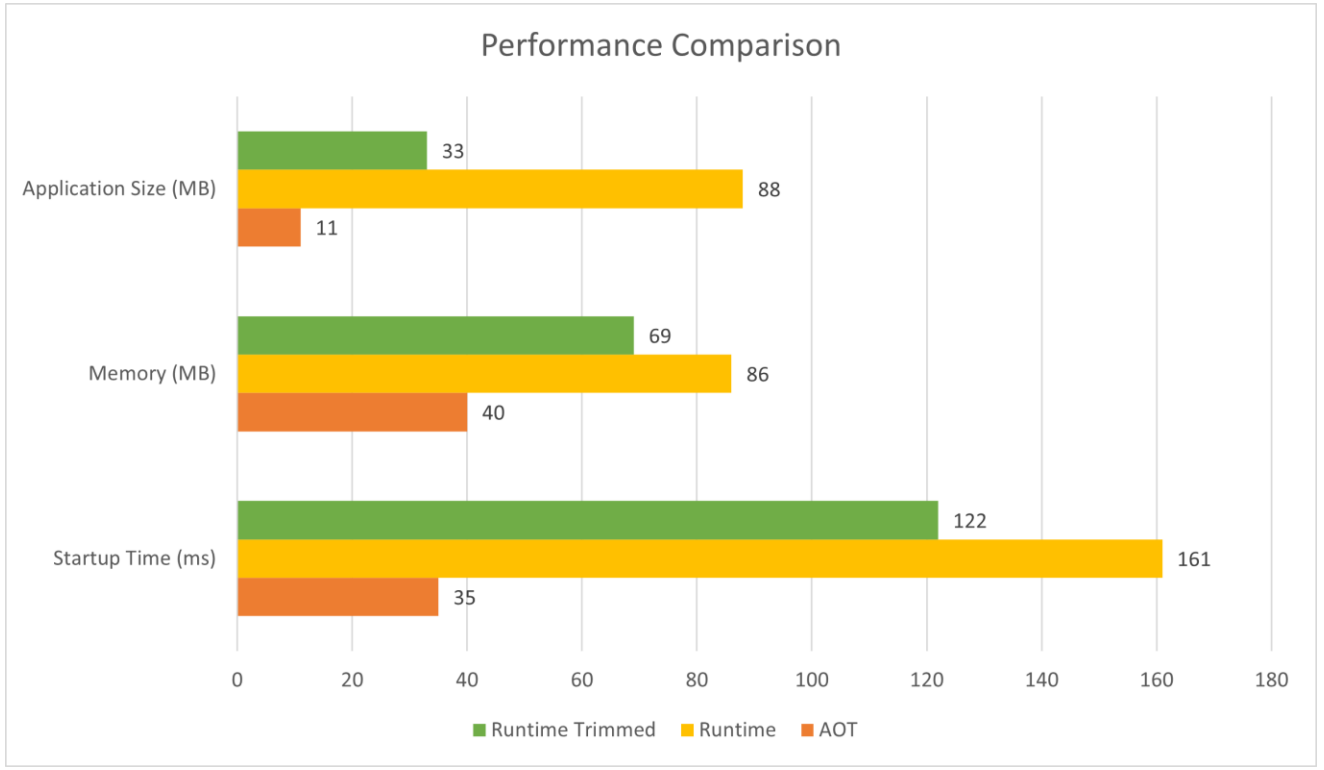
Reflection  Trimming

ASP.NET Core support for native AOT

Grazie all'utilizzo dei Source Generator, con .NET 8 alcune features di ASP.NET Core sono adesso compatibili con la compilazione AOT:

- gRpc
- Minimal APIs (partial)

...

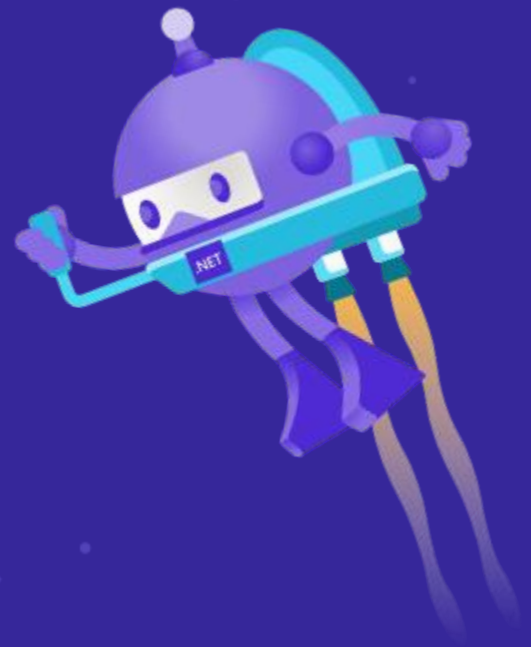


https://learn.microsoft.com/en-us/aspnet/core/fundamentals/native-aot?view=aspnetcore-8.0

https://andrewlock.net/exploring-the-dotnet-8-preview-exploring-the-new-minimal-api-source-generator/

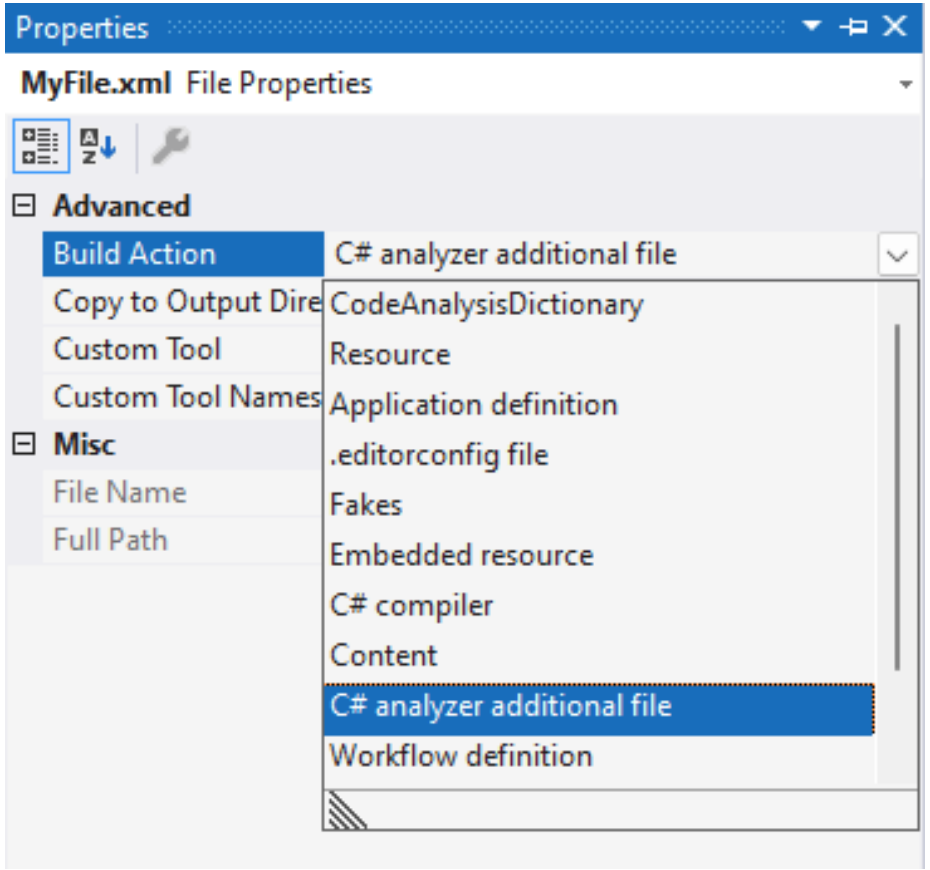
DEMO

Source Generator



Additional Files

- Tramite il provider **AdditionalTextsProvider** posso analizzare anche file di testo e non solo il codice.
- I file (Content) di un progetto non sono automaticamente disponibili nei Source Generator.
- I file devono essere contrassegnati come **AdditionalFile** per essere forniti ai Source Generator.




```
<ItemGroup>
  <AdditionalFiles Include="OtherFiles\MyFile.xml" />
  <AdditionalFiles Include="OtherFiles\MyFile.csv" />
</ItemGroup>
```

Prerequisiti e Limiti

Prerequisiti:

- C# 9.0+ (SDK 5.0.100+)
- Microsoft Visual Studio 16.8.0+

Limiti:

- I source generator non consentono di riscrivere/modificare il codice sorgente dell'utente.
- È possibile solo aggiungere nuovo sorgente C# anche sfruttando le «partial class» e i «partial method».
- Esecuzione non ordinata, ogni generatore vedrà la stessa compilazione di input, senza accesso ai file creati da altri generatori di origine.

Debug

- E’ possibile usare il debugger tramite il metodo Debugger.Launch().
  (Viene aperta un’altra istanza di VS)

- Con VS 2022 e il setting a livello di progetto
  <IsRoslynComponent>true</IsRoslynComponent>
  si può creare un profilo di debug specifico per gli Analyzer ed effettuare il debug del progetto SG nella stessa istanza

- E’ possibile forzare la generazione effettiva dei file sorgente e la cartella di destinazione

```
<PropertyGroup>
    <EmitCompilerGeneratedFiles>true</EmitCompilerGeneratedFiles>
    <CompilerGeneratedFilesOutputPath>DirectoryOutput</CompilerGeneratedFilesOutputPath>
</PropertyGroup>
```

Deploy

- E’ possibile distribuire il nostro Source Generator come pacchetto NuGet

```
<PropertyGroup>
    <GeneratePackageOnBuild>true</GeneratePackageOnBuild> <!-- Generates a package at build -->
    <IncludeBuildOutput>false</IncludeBuildOutput> <!-- Do not include the generator as a lib dependency -->
</PropertyGroup>

<ItemGroup>
    <!-- Package the generator in the analyzer directory of the nuget package -->
    <None Include="$(OutputPath)\$(AssemblyName).dll" Pack="true" PackagePath="analyzers/dotnet/cs" Visible="false" />
</ItemGroup>
```

- Eventuali dipendenze vanno gestire come private a meno che non siano utilizzate anche dal codice generato

```
<ItemGroup>
    <!-- Take a private dependency on Newtonsoft.Json (PrivateAssets=all) Consumers of this generator will not reference it.
     Set GeneratePathProperty=true so we can reference the binaries via the PKGNewtonsoft_Json property -->
    <PackageReference Include="Newtonsoft.Json" Version="12.0.1" PrivateAssets="all" GeneratePathProperty="true" />

    <!-- Package the Newtonsoft.Json dependency alongside the generator assembly -->
    <None Include="$(PkgNewtonsoft_Json)\lib\netstandard2.0\*.dll" Pack="true" PackagePath="analyzers/dotnet/cs" Visible="false" />
</ItemGroup>
```

https://github.com/dotnet/roslyn/blob/main/docs/features/source-generators.cookbook.md#use-functionality-from-nuget-packages

Risorse utili

- Source Generators (Official)
  https://learn.microsoft.com/en-us/dotnet/csharp/roslyn-sdk/source-generators-overview

  roslyn/docs/features/source-generators.cookbook.md at main · dotnet/roslyn (github.com)

- A list of C# Source Generators (Thanks to Amadeusz Sadowski)
  https://github.com/amis92/csharp-source-generators

- Demo Project Source Generator
  https://github.com/bortolin/NetConfSG.Example

- Example without Incremental Source
  https://github.com/bortolin/XeDotNet.SourceGeneratorExample

Marco Bortolin

email: m.bortolin@hunext.com

twitter: @marcobortolin

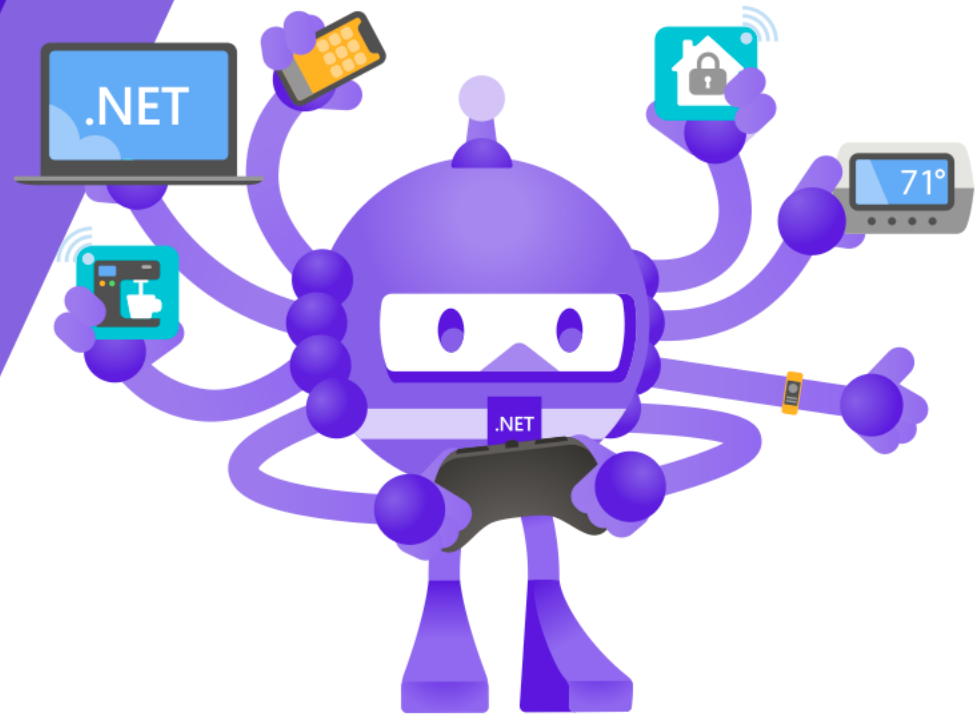
https://github.com/bortolin

https://www.linkedin.com/in/marcobortolin

.NET Conference 2024

VOTA LA SESSIONE

Facci sapere se questa sessione ti piace.
Inquadra il QR code e esprimi una tua opinione.
Ci aiuterai a migliorare.



Microsoft avanade BC Soft consulenza sistemi informativi randstad UNIKEY Packt Reti Società Benefit JetBrains