

Creating Source Code with Source Code Generators

Christian Nagel

<https://www.cninnovation.com>

<https://csharp.christiannagel.com>

Christian Nagel

- Training
 - Coaching
 - Consulting
 - Development
-
- Microsoft MVP
 - www.cninnovation.com
 - csharp.christiannagel.com
 - [@christiannagel](https://twitter.com/christiannagel)
-



Professional **C# and .NET** 2021 Edition

Christian Nagel

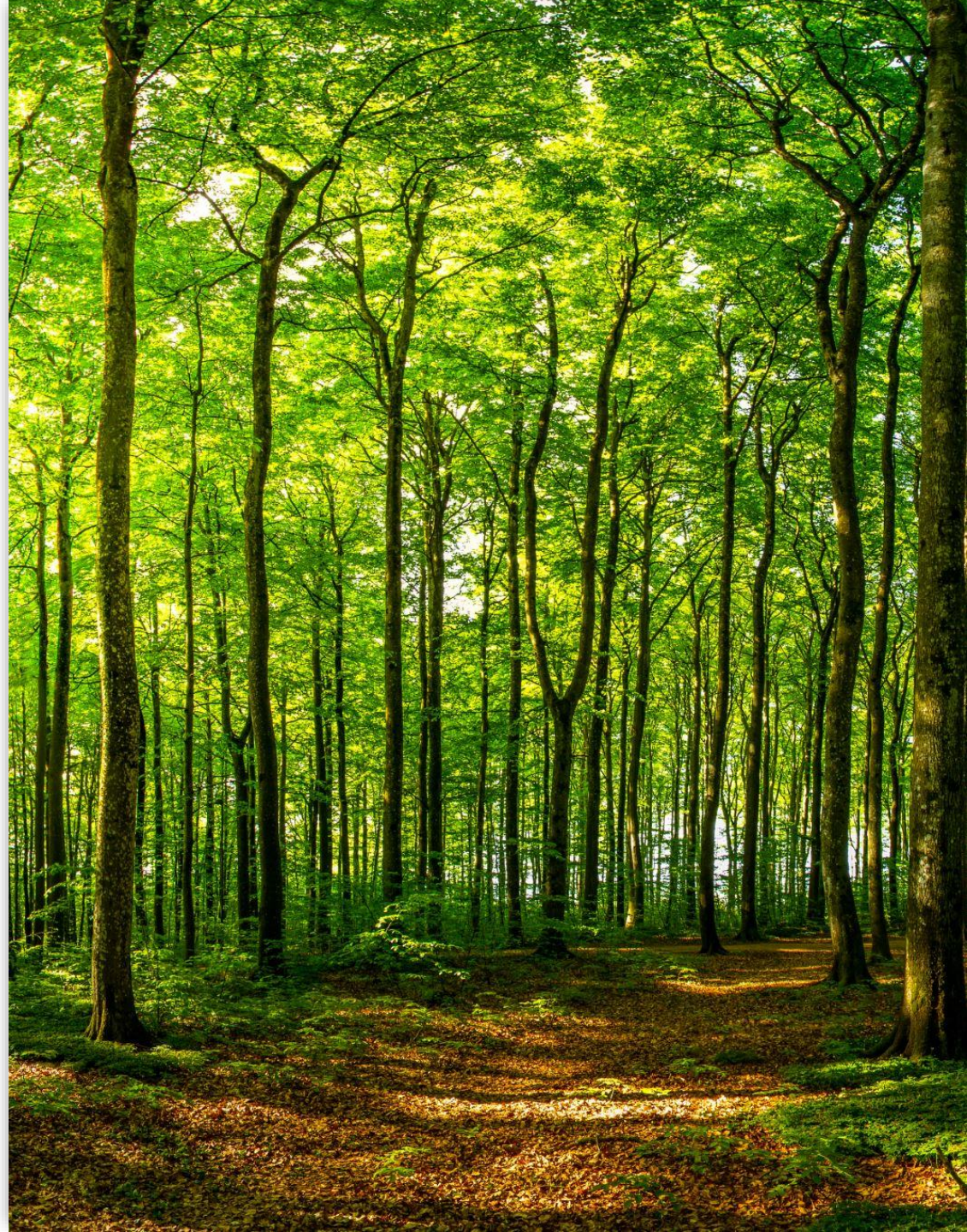
A close-up, low-key photograph of a vintage film projector. The image focuses on the intricate mechanical parts, including a large, spoked film reel on the left and a series of horizontal metal strips or sprockets in the center. A small, bright light source is visible on the right, casting a warm glow on the metallic surfaces. The background is dark, emphasizing the mechanical details.

Agenda

- Intro to source generators
- Using source generators available with .NET
- Creating a custom source generator
- More source generators available

What are source generators?

- Create source code during (pre) compilation
- Access syntax trees of the project's source code
- Remove barriers to linker-based and AOT compilation

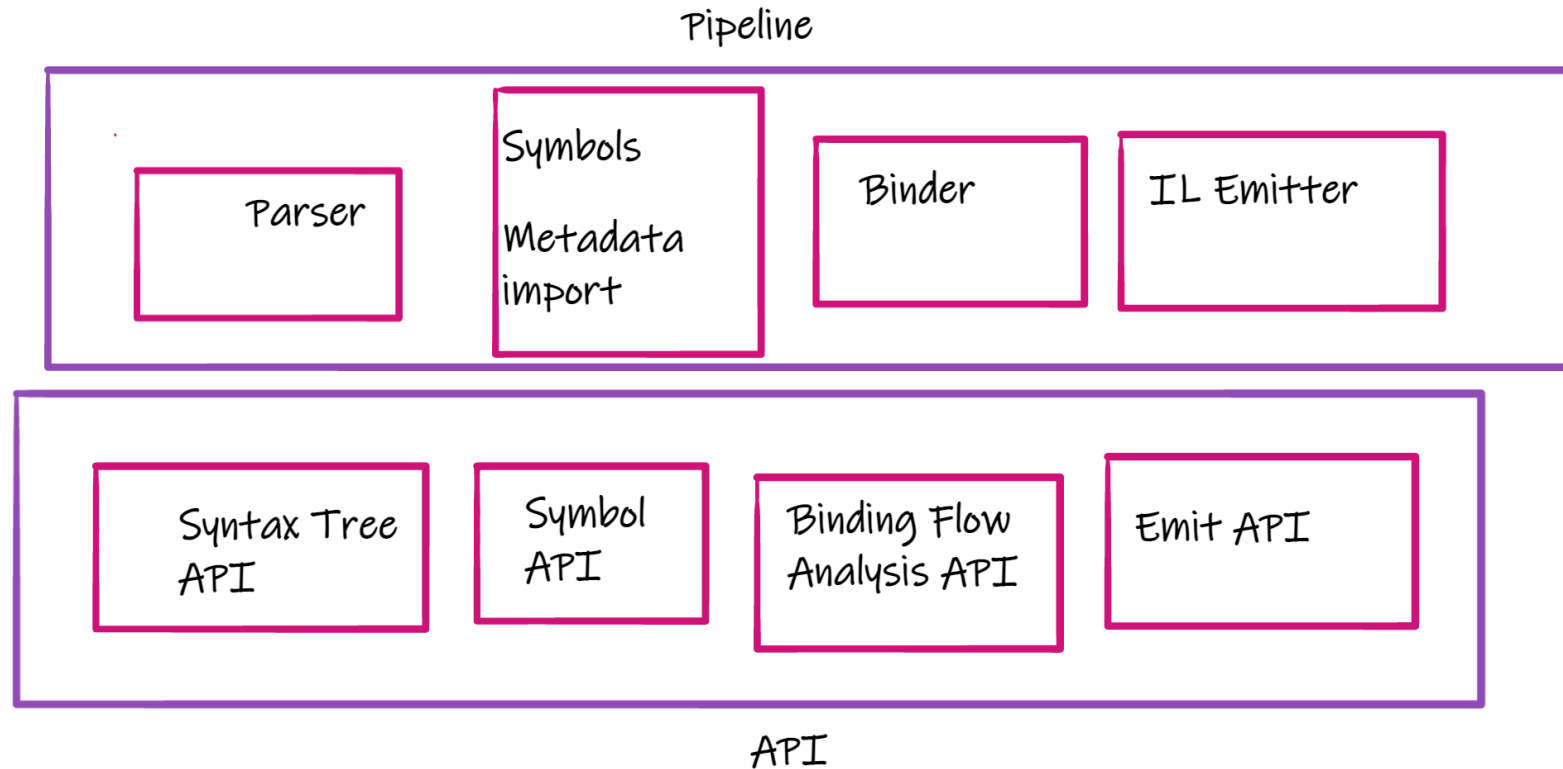




Source Generators History

- C# 6 introduced Roslyn - **The .NET Compiler Platform**
- Object Models for the compilation pipeline
- Analyzers: inspect code quality
- Source generators are based on analyzers

Compiler Pipeline

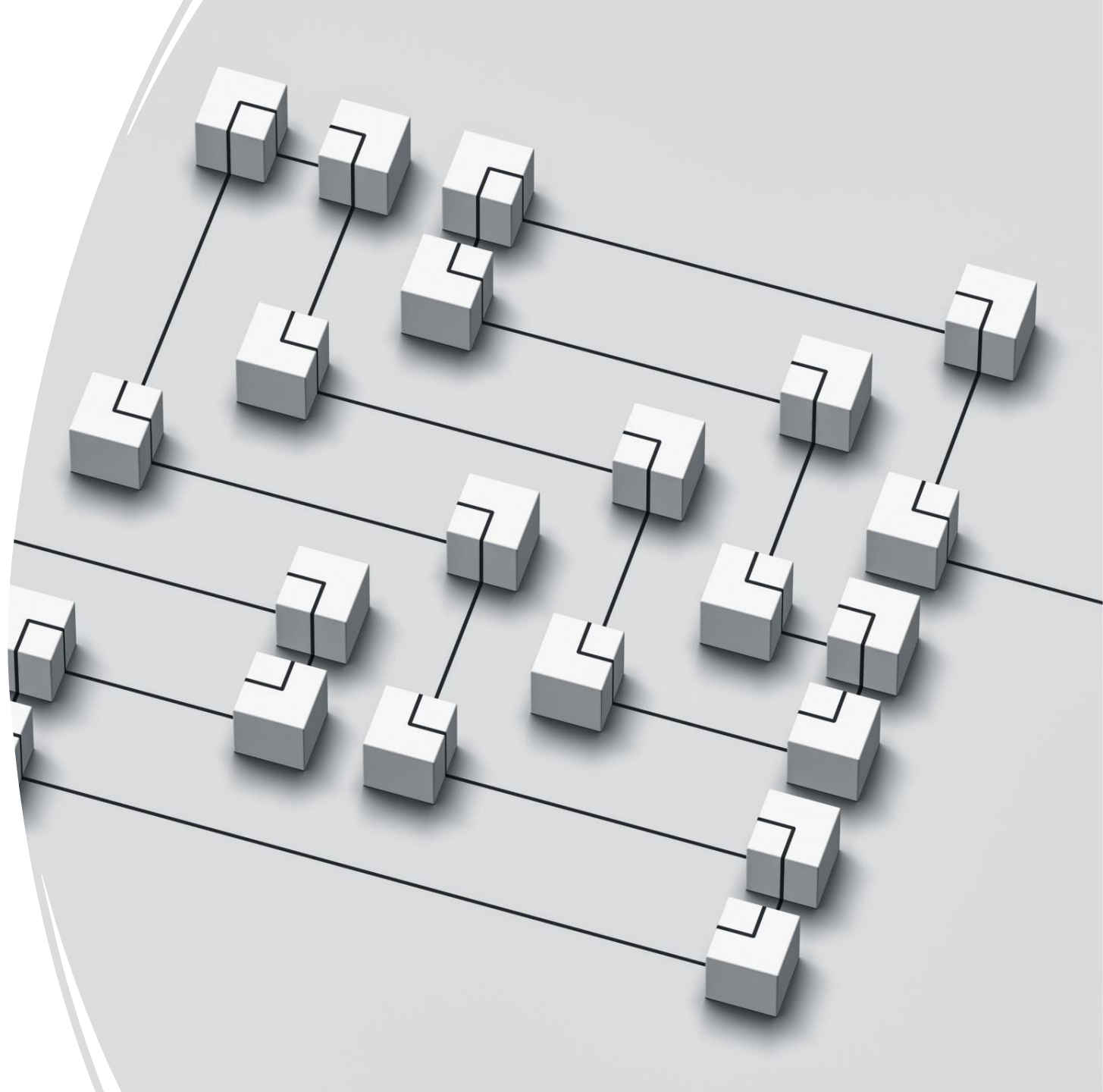


Syntax analysis

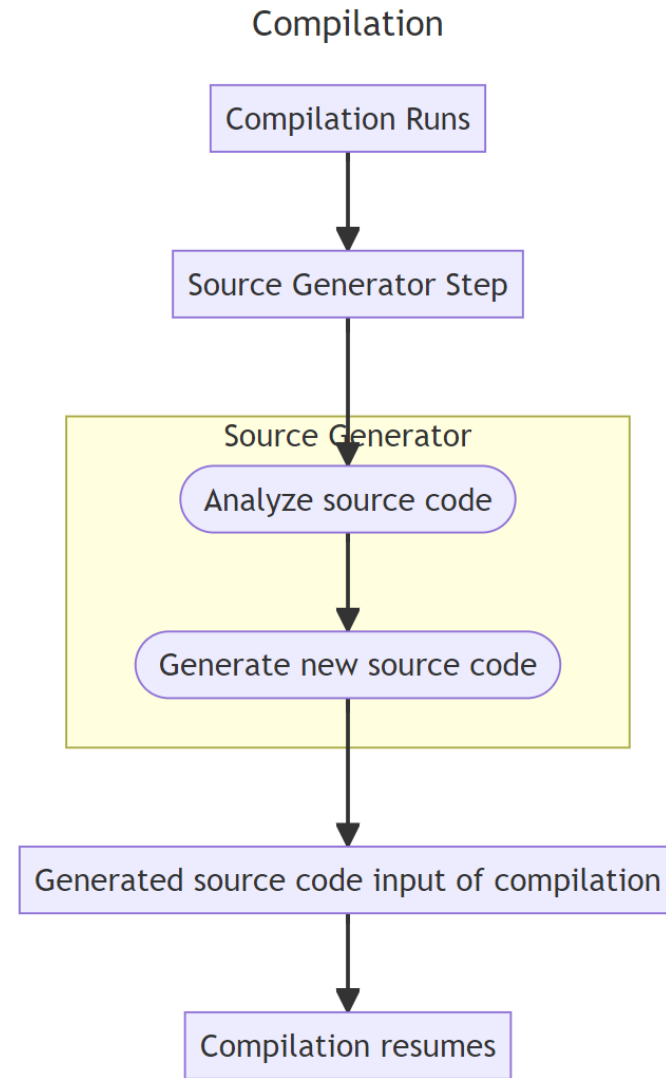
- Syntax API
- Tree structure of the source code
- Query for specific code
- Walk the tree


Semantic Analysis

- Semantic meaning of a program
- Symbols, bindings
- Use the *syntax tree* and a **Compilation** to create a **SemanticModel**



Phases of compilation (with source generator)





Source generators available with .NET (Part 1)

Regular Expressions

- Throughput of `RegexOptions.Compiled`
- Startup benefit – no parsing, analysis, compilation at runtime
- AOT Compilation with code generation
- Debuggability
- Trim app

Regular Expressions Benchmark

- Just a sample with a simple expression!

Method	Mean	Error	StdDev
TestWithGenerator	210.7 ns	2.88 ns	2.69 ns
TestWithReflection	403.8 ns	1.27 ns	1.19 ns

JSON Serialization


- Improve performance (Utf8JsonWriter)
- Reduce runtime dependencies (avoid reflection)
- Customization for serialization options

Platform Invoke

- Reduce runtime overhead using IL stubs
- Supports customization with attributes
- Faster startup

Logging

- Compile-time diagnostics
- Faster startup times
- C# extension with caching delegates



Creating your Source Generator



2 Generations

- Source Generators (C# 9, .NET 5)
 - `ISourceGenerator` Interface, now deprecated
 - `Generator` attribute
- Incremental Generators (.NET 6)
 - `IncrementalGenerator` Interface

The background of the image shows three wind turbines silhouetted against a sunset sky with soft orange and purple hues. The turbines are positioned at different heights and angles, creating a sense of depth. The water in the foreground is represented by stylized blue waves.

Hello,
Generator!

Testing Providers

- Test projects
- Best way to debug source generators
- Use a *CSharpGeneratorDriver* to create and run a source generator
- **Snapshot testing** makes it easy to test expected results

Incremental Generators

- Finer grained approach
- Scale to support Roslyn/CoreCLR scale projects
- Cache between fine grained steps
- Generate more than text
- *IncrementalGenerator* Interface

Incremental Value Provider

- `IncrementalValueProvider<T>`
- `IncrementalValuesProvider<T>`
- Provider supplies transformation
 - `Select<TSource, TResult>`
- Chain multiple transformations

Cache Friendly Generator

- Extract out information early
- Use value types where possible
- Use multiple transformations
- Start with syntax APIs before using semantic analysis
- Build a data model

Incremental Value Providers

- **CompilationProvider**
 - Access the complete compilation
- **AdditionalTextsProvider**
 - Add other text files
- **AnalyzerConfigOptionsProvider**
 - Provide build-time settings, generate code as configured by e.g. editorconfig
- **MetadataReferencesProvider**
 - Reference other packages that are not part of the project
- **ParseOptionsProvider**
 - Customize the options of the source generator, e.g. language version



More source generators (Parts 2)

JavaScript Interop

- Import JavaScript Functions
 - Use them with .NET
- Export .NET Methods
 - Use them with JavaScript

MVVM

- `Communitytoolkit.Mvvm`
- Source generators for `INotifyPropertyChanged`, `ICommand`...
- Removes large chunks boilerplate code

$$F = G \frac{m_1 m_2}{d^2}$$

$$i\hbar \frac{\partial}{\partial t} \psi = \hat{H} \psi$$

$$\phi(x) =$$

$$-E + V = 2$$

$$E = mc^2$$

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}$$

$$\frac{d}{dt}$$

Replace Code

- C# 12 Interceptors (preview in C# 12 / .NET 8)
- Used by .NET 8 Native AOT Minimal API Source Generators

Take away



Use built-in source generators



Think about creating your own source generators



<https://github.com/cnilearn/bastamainz2023>

Thank you for coming!

Questions?

<https://www.cninnovation.com>

<https://csharp.christiannagel.com>