main 12 Branches 32 Tags Go to file Code

AArnott Enable push of daily builds to CI feed ac50cb9 · last month 2,176 Commits

.azuredevops	Enable dependabot for Azure Repos	last year
.config	Bump powershell from 7.4.2 to 7.4.3 (#275)	2 months ago
.devcontainer	Bump .NET SDK to 8.0.300	3 months ago
.github	Merge remote-tracking branch 'libtemplate/microbuild' in...	7 months ago
.vscode	Merge remote-tracking branch 'origin/main' into microbuild	5 months ago
azure-pipelines	Enable push of daily builds to CI feed	last month
doc	Fix typo in ArchSpecificAPIs.md	last year
integration-tests	Fix-up integration tests	2 months ago
src	Fix non-signed build	last month
test	Disable test that fails on microbuild agents	last month
tools	Install .NET 7 runtime for some tests	7 months ago
.editorconfig	Merge remote-tracking branch 'libtemplate/microbuild' in...	2 months ago
.gitattributes	Use LF line endings for plist files	last year
.gitignore	Ignore .sarif files	2 months ago
.prettierrc.yaml	Recommend prettier	6 months ago
CODE_OF_CONDUCT.md	Add files from MS-created repo	4 years ago
CONTRIBUTING.md	Document where the win32metadata gems are	last year
COPYRIGHT	Create COPYRIGHT file	last year
Directory.Build.props	Fix linux build	last month
Directory.Build.rsp	Add Directory.Build.rsp file	4 years ago
Directory.Build.targets	Merge branch 'main' into microbuild	2 months ago
Directory.Packages.props	Bring back the microbuild stuff we deleted before	last month
LICENSE	Fix template expansion	4 years ago
Microsoft.Windows.CsWin32.sln	Remove broken and unused test project	last year
NOTICE.txt	Add NOTICE.txt file	3 years ago
README.md	Avoid `Unsafe.SkipInit` API when it isn't available	2 months ago
SECURITY.md	Merge branch 'main' into microbuild	2 years ago
SUPPORT.md	Merge latest Library.Template	2 years ago
azure-pipelines.yml	Merge remote-tracking branch 'libtemplate/microbuild' in...	5 months ago
global.json	Sign the `.VSInsertionMetadata` optprof package	last month
init.cmd	Fix up init.cmd to set env vars from .ps1 child process	4 years ago
init.ps1	Bring back the microbuild stuff we deleted before	last month
nuget.config	Merge latest Library.Template	2 years ago
settings.VisualStudio.json	Enable auto-format on save in VS and VS Code	9 months ago
stylecop.json	Merge remote-tracking branch 'libtemplate/main' into vall...	2 years ago
version.json	Build stable package version	3 months ago

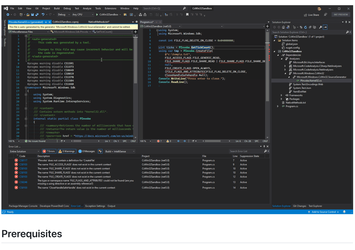
README Code of conduct MIT license Security

C#/Win32 P/Invoke Source Generator

A source generator to add a user-defined set of Win32 P/Invoke methods and supporting types to a C# project.

Features

- Rapidly add P/Invoke methods and supporting types to your C# project.
- No bulky assemblies to ship alongside your application.
- `SafeHandle` -types automatically generated.
- Generates xml documentation based on and links back to docs.microsoft.com



Prerequisites

The .NET 5 SDK or Visual Studio 2019 Update 8 (16.8) for the C# compiler that added support for Source Generators. The experience with source generators in Visual Studio is still improving, and is noticeably better in VS 16.9. WPF projects have additional requirements.

In addition, some generated code may require use of the C# 9 language version (`<LangVersion>9</LangVersion>`) in your project file. See issue #4 for more on this.

See dotnet/pinvoke for precompiled NuGet packages with Win32 P/Invokes.

Usage

Install the `Microsoft.Windows.CsWin32` package:

```
dotnet add package Microsoft.Windows.CsWin32 --prerelease
```

You should also install the `System.Memory` and `System.Runtime.CompilerServices.Unsafe` packages when targeting .NET Framework 4.5+ or .NET Standard 2.0, as these add APIs that significantly improve much of the code generated by CsWin32:

```
dotnet add package System.Memory
dotnet add package System.Runtime.CompilerServices.Unsafe
```

Projects targeting .NET Core 2.1+ or .NET 5+ do not need to add these package references, although it is harmless to do so.

Note that while the `System.Memory` package depends on the `System.Runtime.CompilerServices.Unsafe` package, referencing the latter directly is still important to get the latest version of the APIs it provides.

Your project must allow unsafe code to support the generated code that will likely use pointers. This does *not* automatically make all your code unsafe. Use of the `unsafe` keyword is required anywhere you use pointers. The source generator NuGet package sets the default value of the `AllowUnsafeBlocks` property for your project to `true`, but if you explicitly set it to `false` in your project file, generated code may produce compiler errors.

Create a `NativeMethods.txt` file in your project directory that lists the APIs to generate code for. Each line may consist of one of the following:

- Exported method name (e.g. `CreateFile`). This *may* include the `A` or `W` suffix, where applicable. This *may* be qualified with a namespace but is only recommended in cases of ambiguity, which CsWin32 will prompt where appropriate.
- A macro name (e.g. `HRESULT_FROM_WIN32`). These are generated into the same class with extern methods. Macros must be hand-authored into CsWin32, so let us know if you want to see a macro added.
- A namespace to generate all APIs from (e.g. `Windows.Win32.Storage.FileSystem` would search the metadata for all APIs within that namespace and generate them).
- Module name followed by `.*` to generate all methods exported from that module (e.g. `Kernel32.*`).
- The name of a struct, enum, constant or interface to generate. This *may* be qualified with a namespace but is only recommended in cases of ambiguity, which CsWin32 will prompt where appropriate.
- A prefix shared by many constants, followed by `*`, to generate all constants that share that prefix (e.g. `ALG_SID_MD4*`).
- A comment (i.e. any line starting with `//`) or white space line, which will be ignored.

When generating any type or member, all supporting types will also be generated.

Generated code is added directly in the compiler. An IDE may make this generated code available to view through code navigation commands (e.g. Go to Definition) or a tree view of source files that include generated source files.

Assuming default settings and a `NativeMethods.txt` file with content that includes `CreateFile`, the P/Invoke methods can be found on the `Windows.Win32.PInvoke` class, like this:

```
using Windows.Win32;

PInvoke.CreateFile(/*args*/);
```

Constants are defined on the same class as the p/invoke methods (by default, the `Windows.Win32.PInvoke` class).

Other supporting types are defined within or under the `Windows.Win32` namespace. Discovery of the namespace for a given type can be done with the Go To All feature (Ctrl+T) in Visual Studio with the type name as the search query.

A project may include many NativeMethods.txt files (each one necessarily in its own directory). CsWin32 will read them all to generate APIs, provided these files are included as `AdditionalFiles` in the project. A `NativeMethods.txt` file directly in the project directory is added automatically to `AdditionalFiles`. Files in other directories must be added to the project file manually.

Whether API requests are all in a single NativeMethods.txt file or split across many makes no difference to the generated result. We recommend using just one NativeMethods.txt file and keeping it sorted for easy bookkeeping. Multiple files perhaps makes the most sense in a Shared Project scenario where several API requests will be common across many projects, so sharing a NativeMethods.txt file with those same projects that contain all the necessary APIs for the set of shared source files make maintenance easier.

Some APIs require targeting a specific architecture and are not available when your C# project compiles as "Any CPU". Learn more about how this manifests and what your options are.

Customizing generated code

Several aspects of the generated code can be customized, including:

- The name of the class(es) that declare p/invoke methods
- Whether to emit interop types as `public` or `internal`
- Whether to emit ANSI functions as well where Wide character functions also exist
- Set `PreserveSig` for COM interfaces or individual members
- Force generation of blittable structs, COM structs instead of interfaces (for super high performance with 0 GC pressure), etc.

To configure these settings, create a `NativeMethods.json` file in your project directory. Specifying the `$schema` property that points to the schema adds completions, descriptions and validation in many JSON editors, and in fact is where all the documentation for the available settings is found.

```
{
  "$schema": "https://aka.ms/CsWin32.schema.json",
  "emitSingleFile": false
}
```

Most generated types include the `partial` modifier so you can add your own members to that type within your code.

When you need to *replace* a generated type, simply copy and paste it from generated code into your own source files and remove the `partial` modifier. Be sure to keep the name and namespace exactly the same. CsWin32 will notice that your project already declares the type and skip generating it, but generate everything else. Note that if that type is the only thing that references some other generated type, CsWin32 will stop generating that type too. To keep CsWin32 generating the referred types you need, add them explicitly to `NativeMethods.txt`.

Newer metadata

To update the metadata used as the source for code generation, you may install a newer `Microsoft.Windows.SDK.Win32Metadata` package:

```
dotnet add package Microsoft.Windows.SDK.Win32Metadata --prerelease
```

CsWin32 also consumes the WDK from a similarly named package: `Microsoft.Windows.WDK.Win32Metadata`.

Consuming daily builds

Can't wait for the next release to try out a bug fix? Follow these steps to consume directly from our daily build.

Just add this package feed to your nuget.config file:

```
<add key="winsdk" value="https://pkgs.dev.azure.com/azure-public/winsdk/_packaging/CI/nuget/
```

About
A source generator to add a user-defined set of Win32 P/Invoke methods and supporting types to a C# project.
Readme
MIT license
Code of conduct
Security policy
Activity
Custom properties
2k stars
42 watching
85 forks
Report repository

Releases 31
v0.3.106 Latest
on May 11
+ 30 releases

Packages
No packages published

Used by 1.2k
+ 1,186

Contributors 31
+ 17 contributors

Languages
C# 97.5% PowerShell 8.3%
Other 0.2%

© 2024 GitHub, Inc. Terms Privacy Security Status Docs Contact Manage cookies Do not share my personal information