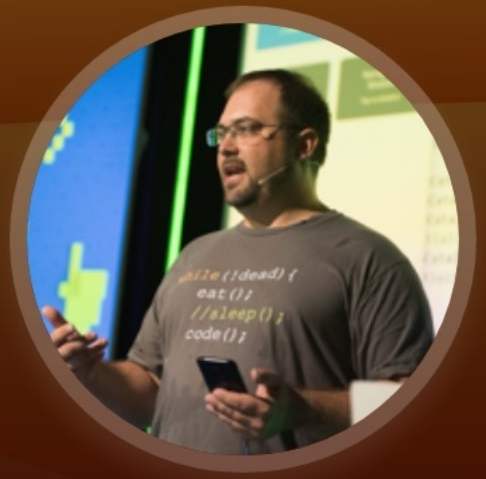
MSBuild Unleashed: Breaking Free From Complex CI Pipelines

Thu, 1/11/2024 9:15 am — Sandusky, Ohio, United States



Dan Siegel

2024 CODEMASH PRIME

WHO IS THIS GUY

Dan Siegel

- Owner – AvantiPoint
- Owner / Maintainer – Prism Library
- Member – Uno Platform Core Maintainers

WHEN I'M NOT WRITING CODE…

1ST QUESTION

- Have you ever found yourself needing to write sometimes complex scripts in order to build your app as part of your CI/CD processes


LIKE
DUH
makeameme.org

AGENDA

1. Introduction to MSBuild
2. PropertyGroups & ItemGroups
3. Common Properties and Items
4. Writing Inline Tasks
5. Writing Compiled Tasks
6. Props and Targets Files
7. Tools for working with MSBuild & NuGet
8. Questions

WHAT IS MSBUILD?

- MSBuild (Microsoft Build Engine) is a build platform for managed code as well as native C++ code.
- It's the engine behind building, publishing, and packaging .NET applications.
- It’s what powers your build from Visual Studio, Rider, and dotnet build

KEY FEATURES

- XML-based project files
- Extensible and customizable
- Supports building in parallel
- Can resolve dependencies automatically

WHY WOULD I EVER NEED TO EXTEND MSBUILD?

WEB DEVELOPERS

- We want to customize the build to perform tasks like npm restore
- We want to compile our SCSS and make sure our compile output is copied to the correct output directory

ANY APP DEVELOPER

- Image Processing – Use a single source image to generate multiple output files for different screen resolutions

MOBILE (iOS)

- Update Entitlements – Apple Push Services update to production environment on Release builds…

THE SKY IS THE LIMIT

PROPERTY GROUPS & ITEM GROUPS

Property Groups

- Think of this like variables you can use or declare

Item Groups

- Think of this like a collection of objects that you will do something with

PROPERTIES

- TargetFramework
- Configuration
- BaseIntermediateOutputPath
- BaseOutputPath

- xam.dev/msbuild-common-props
- xam.dev/msbuild-well-known
- xam.dev/msbuild-nuget-pack

USING PROPERTIES

```
<Product>$(AssemblyName) ($(TargetFramework))</Product>
```

CONDITIONALLY UPDATING A PROPERTY

```
<Product Condition="'$(Product)'==''">$(AssemblyName) ($(TargetFramework))</Product>
```

USING C# CLASSES & STATIC METHODS

```
<Copyright>Copyright (C) 2015-$([System.DateTime]::Now.ToString(`yyyy`)) Uno Platform inc</Copyright>

    <PackageIcon>$([System.IO.Path]::GetFileName($(UnoPackageDownloadedIcon)))</PackageIcon>
```

COMMON ITEMS

- Compile
- None
- EmbeddedResource
- Content
- Page
- Include
- Remove
- Exclude

HOW TO CUSTOMIZE…

IT'S REALLY THAT SIMPLE

```
<FooBar Include="SomeFile.json" />
```

ITEM METADATA - COMMON

- Link="%(Filename)%(Extension)"
- LinkBase="SomeDirectory"

ADVANCED

```
<Project>
    <ItemGroup>
        <!-- Removes native usings to avoid Ambiguous reference -->
        <Using Remove="@(Using->HasMetadata('Platform'))" />
    </ItemGroup>
</Project>
```

CONVENTIONS

- Names that start with a Letter a “Public”
- Names that start with an Underscore are “Private/Internal”
  - Yes you can still use them… it’s convention not actual scoping…

WRITING A CUSTOM TARGET

```
<Target Name="DownloadAndSetPackageIcon" AfterTargets="Build" Condition=" '$(PackageIcon)'=='' ">
    <PropertyGroup>
        <_IconUrl>https://uno-assets.platform.uno/logos/uno.png</_IconUrl>
    </PropertyGroup>

    <DownloadFile SourceUrl="$(_IconUrl)" DestinationFolder="$(IntermediateOutputPath)">
        <Output TaskParameter="DownloadedFile" PropertyName="_UnoPackageDownloadedIcon" />
    </DownloadFile>

    <PropertyGroup>
        <PackageIcon>$([System.IO.Path]::GetFileName($(_UnoPackageDownloadedIcon)))</PackageIcon>
    </PropertyGroup>

    <ItemGroup>
        <None Include="$(_UnoPackageDownloadedIcon)" Pack="true" PackagePath="\" Visible="false" />
    </ItemGroup>
</Target>
```


WRITING INLINE TASKS

CREATE AN INLINE TASK

```
<UsingTask TaskName="ReplaceFileText"
           TaskFactory="RoslynCodeTaskFactory"
           AssemblyFile="$(MSBuildToolsPath)\Microsoft.Build.Tasks.Core.dll">
    <!-- Your Task Here… -->
</UsingTask>
```

ADD TASK PARAMETERS

```
<UsingTask TaskName="ReplaceFileText"
           TaskFactory="RoslynCodeTaskFactory"
           AssemblyFile="$(MSBuildToolsPath)\Microsoft.Build.Tasks.Core.dll">
    <ParameterGroup>
        <Filename ParameterType="System.String" Required="true" />
        <MatchExpression ParameterType="System.String" Required="true" />
        <ReplacementText ParameterType="System.String" Required="true" />
    </ParameterGroup>
    <!- Your Task Here… -->
</UsingTask>
```

ADD THE TASK CODE

```
<UsingTask TaskName="ReplaceFileText"
           TaskFactory="RoslynCodeTaskFactory"
           AssemblyFile="$(MSBuildToolsPath)\Microsoft.Build.Tasks.Core.dll">
    <ParameterGroup>
        <Filename ParameterType="System.String" Required="true" />
        <MatchExpression ParameterType="System.String" Required="true" />
        <ReplacementText ParameterType="System.String" Required="true" />
    </ParameterGroup>

    <Task>
        <Using Namespace="System" />
        <Using Namespace="System.IO" />
        <Using Namespace="System.Text.RegularExpressions" />
        <Code Type="Fragment" Language="cs">
            <![CDATA[
                File.WriteAllText(
                    Filename,
                    Regex.Replace(File.ReadAllText(Filename), MatchExpression, ReplacementText)
                    );
            ]]>
        </Code>
    </Task>
</UsingTask>
```


WRITING COMPILED TASKS

CREATE A CLASS LIBRARY

- Install the ‘Microsoft.Build.Tasks.Core’ NuGet with the version matching the version of Visual Studio you would like to support… (This may not be the latest package available)
- Based on the version of Microsoft.Build.Tasks.Core you may want to update your class library TargetFramework to netstandard2.0, net472, net6.0 or net7.0 if targeting 17.4.0 or later

SETUP THE CSPROJ

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

  <PropertyGroup>
    <TargetFramework>netstandard2.0</TargetFramework>
    <DevelopmentDependency>true</DevelopmentDependency>
    <BuildOutputTargetFolder>build</BuildOutputTargetFolder>
  </PropertyGroup>


  <ItemGroup>
    <None Include="build\Package.props"
          Pack="true"
          PackagePath="build\$(PackageId).props" />
    <None Include="build\Package.targets"
          Pack="true"
          PackagePath="build\$(PackageId).targets" />
  </ItemGroup>

</Project>
```

CREATE A CLASS LIBRARY

```
using Microsoft.Build.Framework;
using Microsoft.Build.Utilities;

namespace MyAwesomeProject;

public class HelloWorld : Task
{
    [Required]
    public string Message { get; set; }

    public override bool Execute()
    {
        Log.LogMessage(MessageImportance.High, Message);
        return true;
    }
}
```

CREATE OUR PACKAGE.PROPS

```
<Project>

  <PropertyGroup>
    <_MyProjectDllPath>$(MSBuildThisFileDirectory)netstandard2.0\MyProject.dll</_MyProjectDllPath>
  </PropertyGroup>

</Project>
```

CREATE OUR PACKAGE.TARGETS

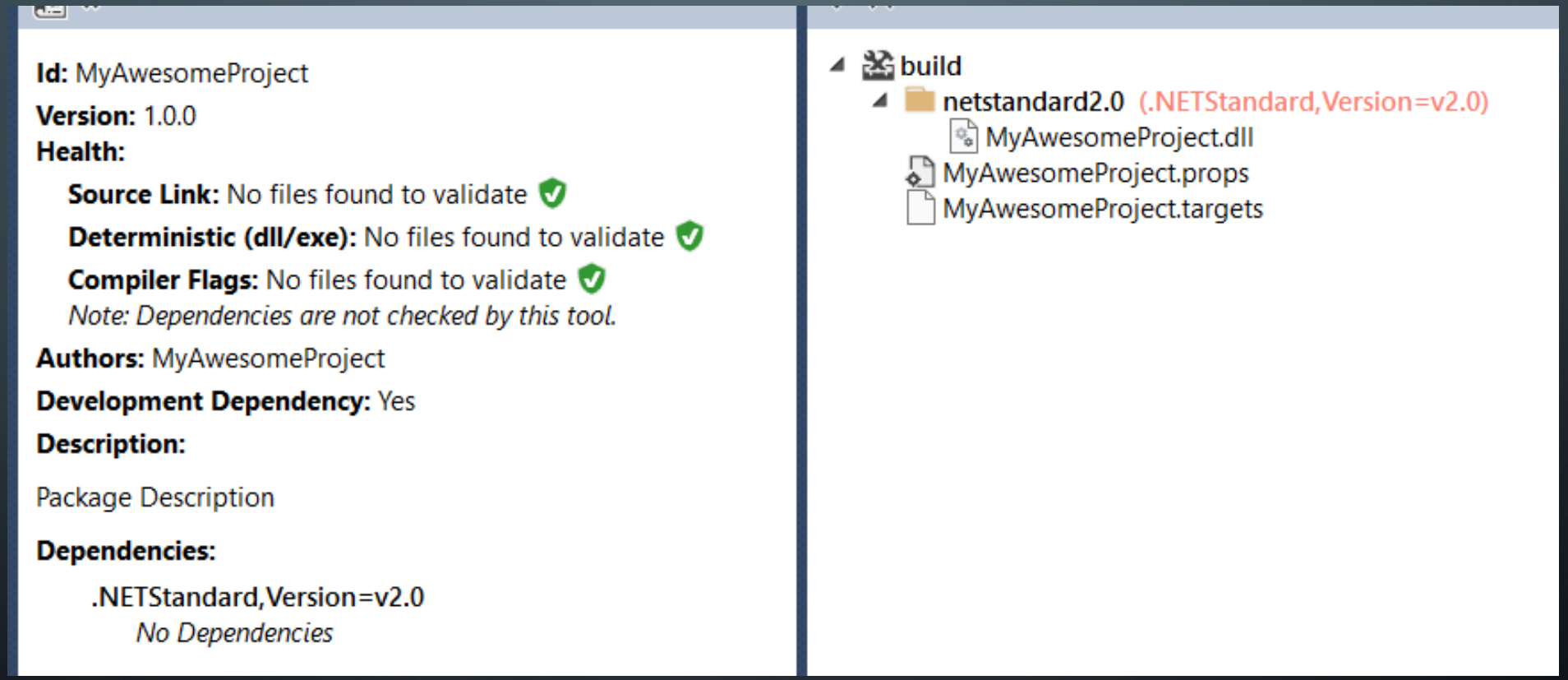
```
<Project>

  <UsingTask TaskName="MyAwesomeProject.HelloWorld“
             AssemblyFile="$(MyProjectDllPath)"/>


  <Target Name="HelloWorld" AfterTargets="Build">
    <HelloWorld Message="Hello from my Custom Target" />
  </Target>

</Project>
```

WHAT'S IN THE GENERATED NUGET



INPUTS & OUTPUTS

```
<Project>

  <Target Name="_CollectScss">
    <ItemGroup>
      <_AllScss Include="**\*.scss" />
      <_PartialScss Include="@(_AllScss)"
                    Condition="$([MSBuild]::ValueOrDefault('%(Filename)', '').StartsWith('_'))" />
      <_Scss Include="@(_AllScss)" Exclude="@(_PartialScss)" />
    </ItemGroup>
  </Target>

  <Target Name="ProcessScss"
          BeforeTargets="CssG"
          AfterTargets="_CollectScss"
          DependsOnTargets="MobileBuildToolsInit"
          Inputs="@(_Scss);@(_PartialScc)"
          Outputs="@(_Scss -> '$(IntermediateOutputPath)%(RecursiveDir)%(Filename).css')"
          Condition=" '@(ReferencePath->WithMetadataValue('NuGetPackageId', 'Xamarin.Forms'))' != '' ">
  </Target>

</Project>
```


PROPS & TARGET FILES

PROPS FILES

- Props files are loaded at the beginning.
- They cannot contain Targets
- There are 3 special types of props files
  - Directory.Build.props
  - Directory.Package.props
  - {PackageId}.props – This must match the NuGet Package Id it is contained in

TARGETS FILES

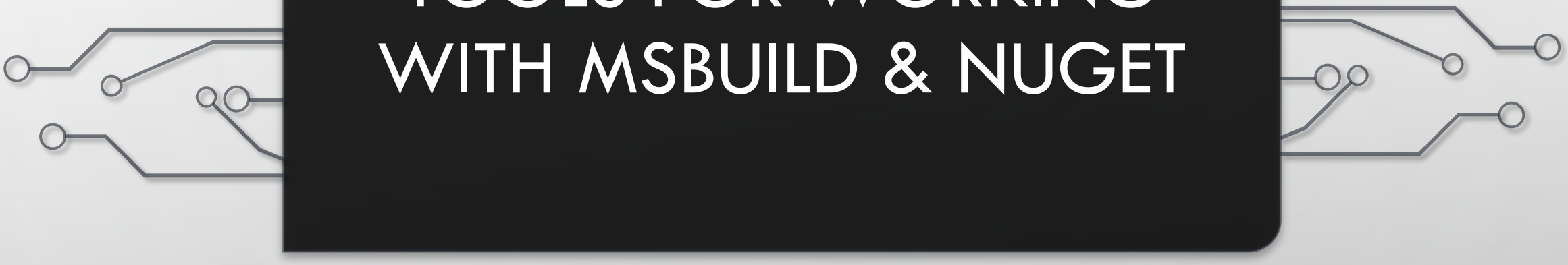
- Targets Files are loaded last.
- You can include Property Groups, ItemGroups, Inline Tasks, & Targets
- There are 2 types of special targets files
  - Directory.Build.targets
  - {PackageId}.targets – This must match the NuGet Package Id it is contained in

NAMING

- Directory.Build.props / Directory.build.props / directory.build.props – Casing matters depending on where you build… It is Directory.Build.props

MANUALLY IMPORTING PROPS & TARGETS

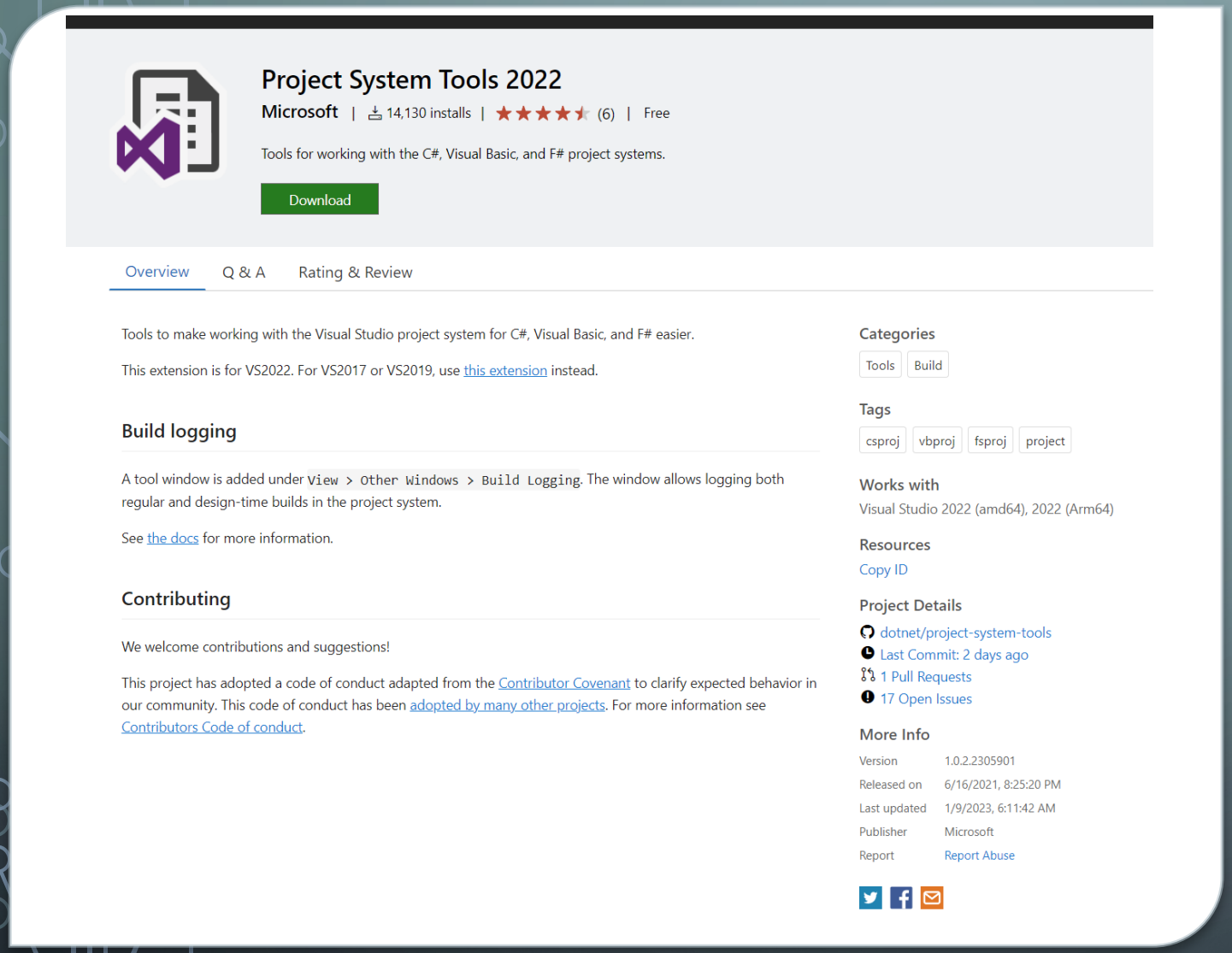
```
<Import Project="../Path/Some.props" />
```


TOOLS FOR WORKING
WITH MSBUILD & NUGET

BINLOG

- Binlogs give us a complete picture of what is happening in our build
- Binlogs can be created from the command line using msbuild or dotnet build by adding /bl
- A less complete binlog can be created in Visual Studio by installing the Project System Tools 2022 extension

PROJECT SYSTEM TOOLS



Project System Tools 2022

Microsoft | 14,130 installs | ★★★★½ (6) | Free

Tools for working with the C#, Visual Basic, and F# project systems.

Download

Overview Q & A Rating & Review

Tools to make working with the Visual Studio project system for C#, Visual Basic, and F# easier.

This extension is for VS2022. For VS2017 or VS2019, use this extension instead.

Build logging

A tool window is added under `View > Other Windows > Build Logging`. The window allows logging both regular and design-time builds in the project system.

See the docs for more information.

Contributing

We welcome contributions and suggestions!

This project has adopted a code of conduct adapted from the Contributor Covenant to clarify expected behavior in our community. This code of conduct has been adopted by many other projects. For more information see Contributors Code of conduct.

Categories

Tools Build

Tags

csproj vbproj fsproj project

Works with

Visual Studio 2022 (amd64), 2022 (Arm64)

Resources

Copy ID

Project Details

dotnet/project-system-tools
Last Commit: 2 days ago
1 Pull Requests
17 Open Issues

More Info

Version	1.0.2.2305901
Released on	6/16/2021, 8:25:20 PM
Last updated	1/9/2023, 6:11:42 AM
Publisher	Microsoft
Report	Report Abuse

NUGET PACKAGE EXPLORER

- This is great for anyone creating a NuGet package
- You can verify files are packed correctly
- You can inspect 3rd party packages prior to adding them
- You can verify tasks such as Deterministic Builds and Source Link were properly enabled
- Install via Chocolatey or the Windows Store
- Use the Uno Platform PWA – https://nuget.info


QUESTIONS

LINKS

- dan@prismlibrary.com
- twitter.com/DanJSiegel
- youtube.com/dansiegel
- xam.dev/msbuild-common-props
- xam.dev/msbuild-well-known
- xam.dev/msbuild-nuget-pack
- nuget.info