Create Xamarin NuGet on MacOS

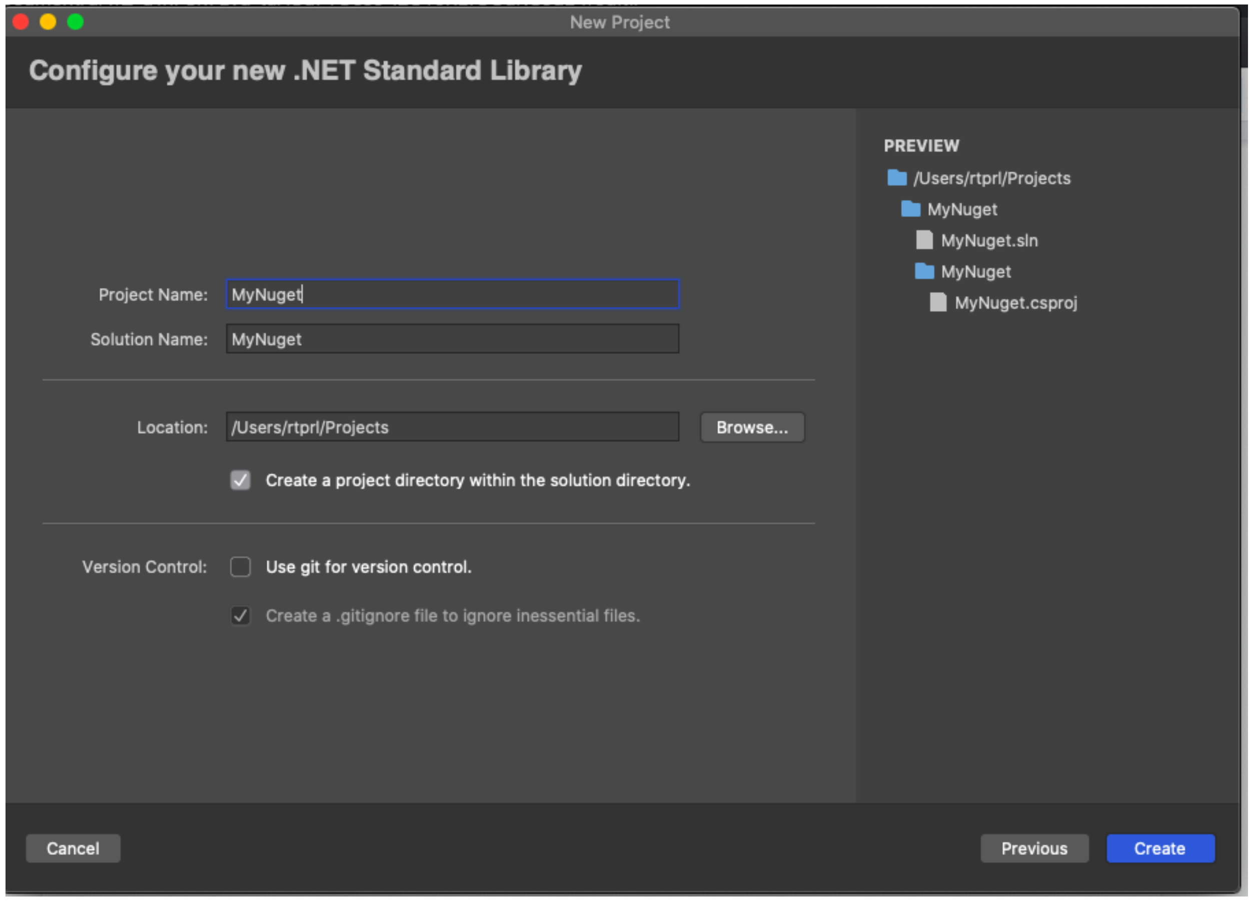
Ritesh Chatterjee

NuGet is like a small packet of code which works like a library and can be used in .net projects to help us (developers) do good things with the smallest amount of coding. In Xamarin, a developer can target multiple platforms. NuGet helps abstract out the platform specific details, so that the developer can focus on the logical coding more than resolving the target dependencies. It is really easy to use nuggets with Visual Studio NuGet package manager.

Besides abstraction, NuGet is a great way to package frequently reused functionality in your other projects. In Windows machines, it is rather straightforward to develop a cross-platform .net library or a Xamarin Nuget. Here is the link to a very informative step-by-step tutorial by James Montemagno for building a .net library in Windows. The problem is that visual studio for Mac doesnt have such a template. After several days of failed attempts to build a Nuget in MacOS following Motemagno's tutorial, I finally got it running. Realizing the lack of a good tutorial on how to build a cross-platform Nuget in MacOS, I decided to write one.

In this post, I will describe how to create a Nuget on MacOS. I will focus on Xamarin NuGet here (Creating just a .net Standard library and pack and use it for web projects is very much straightforward.) In the following steps, I will focus on Android and iOS platforms but you can target any other platform as well, as long as you have the SDKs. Lets go ahead and get the hands dirty (in code) ...

1. First, create a .net standard project. (You can select any target but better to keep it latest select 2.0 or 2.1 in this part, just to get started. We will modify the project later by changing the csproj file.)



New Project

Configure your new .NET Standard Library

PREVIEW
- /Users/rtprl/Projects
  - MyNuget
    - MyNuget.sln
    - MyNuget
      - MyNuget.csproj

Project Name: MyNuget

Solution Name: MyNuget

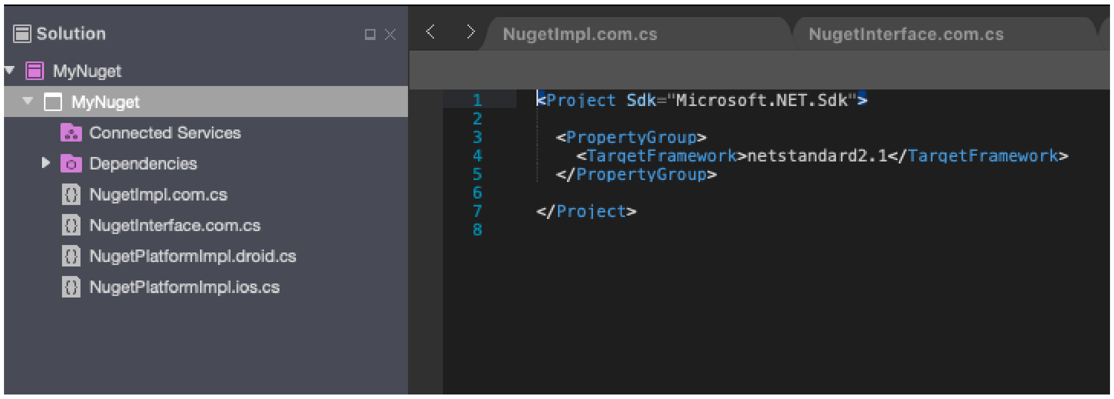
Location: /Users/rtprl/Projects

[✓] Create a project directory within the solution directory.

Version Control: [ ] Use git for version control.

[✓] Create a .gitignore file to ignore inessential files.

2. Create a file structure. We will create 3 types of files. As we are targeting two platforms, so we need three types of files. In general, if you target for n platforms, you will be needing n+1 files. We must decide on the suffix name for each platform, as the naming convention will be ***Filename.PlatformSuffix.cs*** for any platform
    1. There will be one set for common files which should be compiled always for any target framework. We need two files to start with:
        1. ***InterfaceName.CommonSuffix.cs.*** This is an interface to maintain the features that you want to provide in that as method signatures.
        2. ***MasterFilename.CommonSuffix.cs.*** This class will be responsible for returning the implementation class based on the target platform.
    2. Create an implementation file for the Android project to define the features with the help of android libraries. (***ImplementationClassname.AndroidSuffix.cs***)
    3. Create an Implementation file for ios project to define the features with the help of android libraries. (***ImplementationClassname.IOSSuffix.cs***). Remember this ***ImplementationClassname*** should be the same as the one for Android.



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

  <PropertyGroup>
    <TargetFramework>netstandard2.1</TargetFramework>
  </PropertyGroup>

</Project>
```

3. As our structure is ready now, we need to change the csproj file.
    1. Open the csproj file of the project. Now lets break down the csproj file that you will have now.

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

This is the top level sdk declaration for the project. But we now want Multi Platform Target and for that we have one awesome support from developers. We need to change that sdk to the new one to support multiple libraries. So long story short we will edit this as,

```
<Project Sdk="MSBuild.Sdk.Extras/2.1.2”>
```

    2. Now the next part will look like,

```
<PropertyGroup>
    <TargetFramework>netstandard2.0</TargetFramework>
</PropertyGroup>
```

Now we have to modify it to support multiple target frameworks. Just by adding the frameworks one after another separated by semicolon. Something like this below(iOS, Android, .netstandard supports). You can add other library supports too. Replace the above section with the below one. Be cautious about the monoandroid version you are adding cause this will require your machine to have a certain android API version. Check for the API version you have installed and add the corresponding mono version. MonoAndroid90 means Android 9 or API level 28.

```
<PropertyGroup>
<TargetFrameworks>netstandard2.0;netstandard1.0;Xamarin.iOS10;MonoAndroid90</TargetFrameworks>
</PropertyGroup>
```

3. Now add a few things to the file within the same <PropertyGroup> tag. PackOnBuild,GeneratePackageOnBuild is required to create the NuGet on build, EnableDefaultCompileItems set to false means to stop compiler auto include files for compilation. Assembly name you can keep anything but best practice keep it the same as the root namespace(which is actually the solution name) for your project.

```
          <AssemblyName>MYNuGet</AssemblyName>
            <PackOnBuild>true</PackOnBuild>
           <EnableDefaultNoneItems>false</EnableDefaultNoneItems>
           <EnableDefaultCompileItems>false</EnableDefaultCompileItems>
           <DisableExtraReferences>true</DisableExtraReferences>
          <GeneratePackageOnBuild>true</GeneratePackageOnBuild>
           <PackageId>MYNuGet</PackageId>
           <PackageVersion>1.0.0</PackageVersion>
           <Product>$(AssemblyName) ($(TargetFramework))</Product>
           <RootNamespace>MYNuGet</RootNamespace>
```

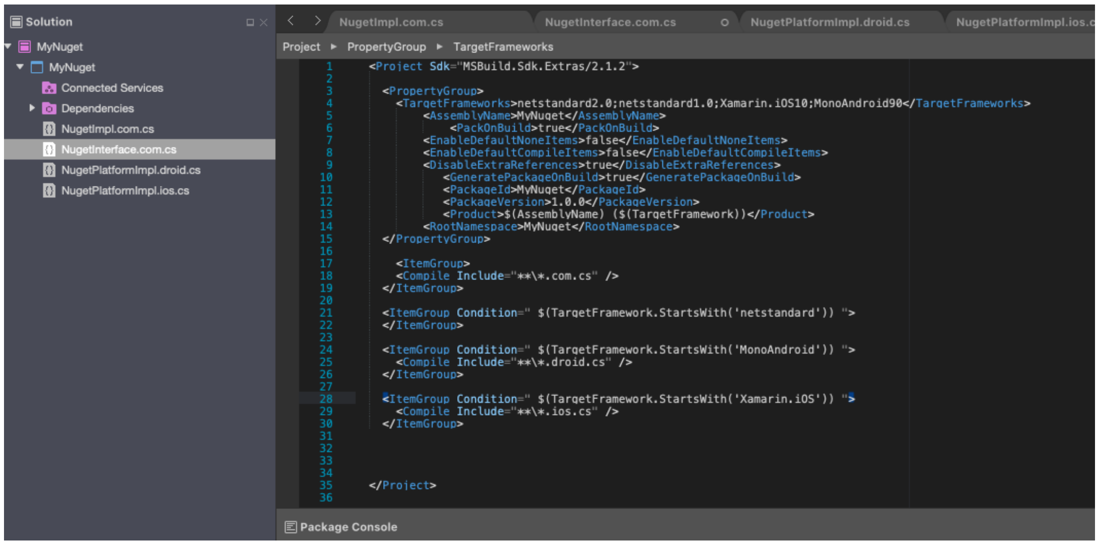
4. Now we need to add the platform specific compile paths to tell the compiler if the target framework is this then target these.

```
<ItemGroup>
      <!-- FOR COMMON SPECIFIC NAME CONVENTION CHANGE “com” TO COMMONSUFFIX, SHOULD BE COMPILED FOR ALL TARGET FRAMEWORKS -->
    <Compile Include="**\*.com.cs" />
  </ItemGroup>

  <ItemGroup Condition=" $(TargetFramework.StartsWith('netstandard')) ">
  </ItemGroup>

  <ItemGroup Condition=" $(TargetFramework.StartsWith('MonoAndroid')) ">
     <!-- FOR ANDROID SPECIFIC NAME CONVENTION CHANGE “droid” TO SOMETHING ELSE -->
    <Compile Include="**\*.droid.cs" />
  </ItemGroup>

  <ItemGroup Condition=" $(TargetFramework.StartsWith('Xamarin.iOS')) ">
    <!-- FOR ANDROID SPECIFIC NAME CONVENTION CHANGE “ios” TO SOMETHING ELSE -->
    <Compile Include="**\*.ios.cs" />
  </ItemGroup>
```

5. Last but not the least remove all the other lines except the bottom </project> tag(mostly due to newly added files for point 2 there will be few lines added in the compiler, just remove those). You can try saving the file without removing it but you might get an error or at least a warning specifying *the same file specified for compilation multiple times* or something of that sort. So better remove the lines and save and rebuild the project once. If it fails then somewhere in the csproj you have messed up. For reference adding the whole csproj file below and your one should look somewhat like this. There are few more commented out lines which we need to take care while publishing the nuget. For this follow the nuspec creation and publish steps as mentioned here.

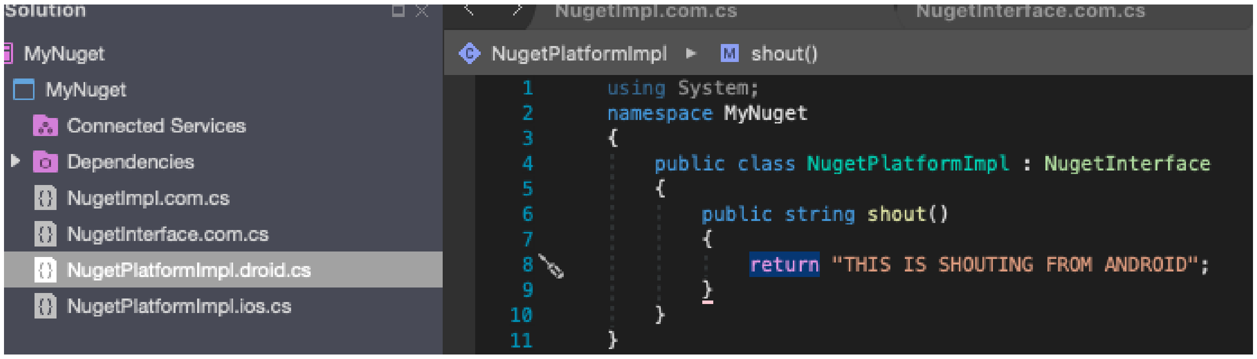
```
<Project Sdk="MSBuild.Sdk.Extras/2.1.2">

  <PropertyGroup>
      <!--netstandard support package and corresponding target os version(API 28[android 9.0] means monoandroid90, IOS 13 means Xamarin.iOS10 etc) add them-->

<TargetFrameworks>netstandard1.0;netstandard2.0;Xamarin.iOS10;MonoAndroid90</TargetFrameworks>

     <!-- PackOnBuild,GeneratePackageOnBuild is required to create the NuGet on build, EnableDefaultCompileItems set to false means to stop compiler auto include files for compilation -->
    <RootNamespace>MYNuget</RootNamespace>
    <AssemblyName>MYNuGet</AssemblyName>
    <PackOnBuild>true</PackOnBuild>
    <EnableDefaultNoneItems>false</EnableDefaultNoneItems>
    <EnableDefaultCompileItems>false</EnableDefaultCompileItems>
    <DisableExtraReferences>true</DisableExtraReferences>
    <GeneratePackageOnBuild>true</GeneratePackageOnBuild>
    <!--Package definition edit as you want ,things that you can skip is already commented out at bottom but add them as to publish them these fields are required-->
    <PackageId>MYNuGet</PackageId>
    <PackageVersion>1.0.0</PackageVersion>
  </PropertyGroup>

  <ItemGroup>
    <!-- FOR ANDROID SPECIFIC NAME CONVENTION CHANGE “com” TO SOMETHING ELSE -->
    <Compile Include="**\*.com.cs" />
  </ItemGroup>

  <ItemGroup Condition=" $(TargetFramework.StartsWith('netstandard')) ">
  </ItemGroup>

  <ItemGroup Condition=" $(TargetFramework.StartsWith('MonoAndroid')) ">
     <!-- FOR ANDROID SPECIFIC NAME CONVENTION CHANGE “droid” TO SOMETHING ELSE -->
    <Compile Include="**\*.droid.cs" />
  </ItemGroup>

  <ItemGroup Condition=" $(TargetFramework.StartsWith('Xamarin.iOS')) ">
```

```
    <!-- FOR ANDROID SPECIFIC NAME CONVENTION CHANGE “ios” TO SOMETHING ELSE -->
    <Compile Include="**\*.ios.cs" />
  </ItemGroup>

     <!-- Fill in these fields when you want to create publish the nuget with
proper data not required while development -->
    <!-- <NeutralLanguage>en</NeutralLanguage>
    <LangVersion>default</LangVersion>
    <PackageLicenseUrl>LINK TO LICENSE</PackageLicenseUrl>
    <PackageProjectUrl>LINK TO PROJECT</PackageProjectUrl>
    <RepositoryUrl>LINK TO PROJECT</RepositoryUrl>
    <PackageReleaseNotes>RELEASE NOTES</PackageReleaseNotes>
    <PackageIconUrl>ICON URL</PackageIconUrl>
             <PackageTags>xamarin,   ios,   android,   xamarin.forms,   plugin,
MYNuGet</PackageTags>
    <Title>Test Nuget Plugin for Xamarin</Title>
    <Summary>Summary of nuget</Summary>
    <Description>Plugin Description</Description>
    <Owners>YOU</Owners>
    <Authors>YOU</Authors>
    <Copyright>Copyright 2019</Copyright>
    <Version>1.0.0</Version>
    <LangVersion>latest</LangVersion>
    <DebugType>portable</DebugType> -->

</Project>
```

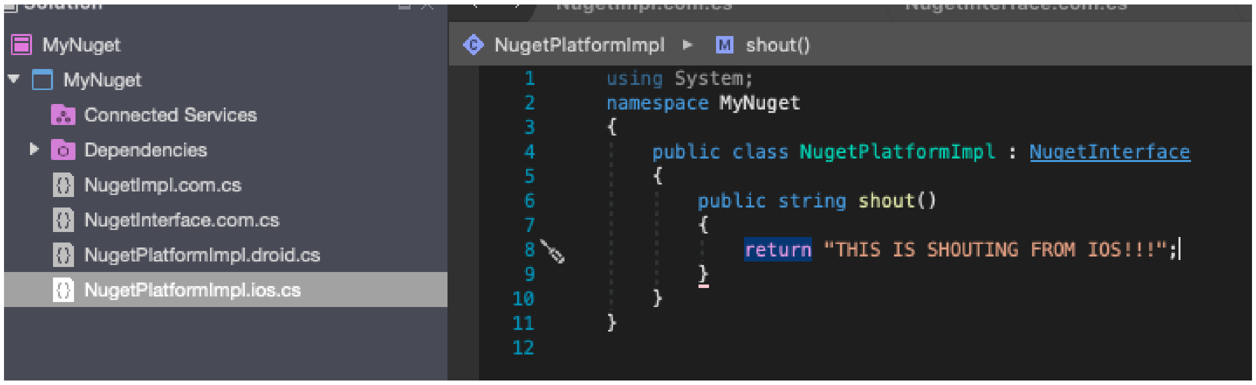


```
<Project Sdk="MSBuild.Sdk.Extras/2.1.2">

  <PropertyGroup>
    <TargetFrameworks>netstandard2.0;netstandard1.0;Xamarin.iOS10;MonoAndroid90</TargetFrameworks>
        <AssemblyName>MyNuget</AssemblyName>
            <PackOnBuild>true</PackOnBuild>
        <EnableDefaultNoneItems>false</EnableDefaultNoneItems>
        <EnableDefaultCompileItems>false</EnableDefaultCompileItems>
        <DisableExtraReferences>true</DisableExtraReferences>
           <GeneratePackageOnBuild>true</GeneratePackageOnBuild>
           <PackageId>MyNuget</PackageId>
           <PackageVersion>1.0.0</PackageVersion>
           <Product>$(AssemblyName) ($(TargetFramework))</Product>
        <RootNamespace>MyNuget</RootNamespace>
  </PropertyGroup>

    <ItemGroup>
    <Compile Include="**\*.com.cs" />
  </ItemGroup>

  <ItemGroup Condition=" $(TargetFramework.StartsWith('netstandard')) ">
  </ItemGroup>

  <ItemGroup Condition=" $(TargetFramework.StartsWith('MonoAndroid')) ">
    <Compile Include="**\*.droid.cs" />
  </ItemGroup>

  <ItemGroup Condition=" $(TargetFramework.StartsWith('Xamarin.iOS')) ">
    <Compile Include="**\*.ios.cs" />
  </ItemGroup>




</Project>
```

4. Now Add a basic method, say to return just a string in the interface to test if we are on the right track. Make the implementation classes for android and iOS implement them with some return statements different for the platforms. Also do check if you are able to corresponding platform libraries in those implementation classes.



```
using System;
namespace MyNuget
{
    public class NugetPlatformImpl : NugetInterface
    {
        public string shout()
        {
            return "THIS IS SHOUTING FROM ANDROID";
        }
    }
}
```



```
using System;
namespace MyNuget
{
    public class NugetPlatformImpl : NugetInterface
    {
        public string shout()
        {
            return "THIS IS SHOUTING FROM IOS!!!";
        }
    }
}
```

5. Our frame is ready, so just need MasterFile to do the trick. First of all make the class static and remove the constructor.
    1. Then create a private member of ***Lazy<Interfacename>*** like this.

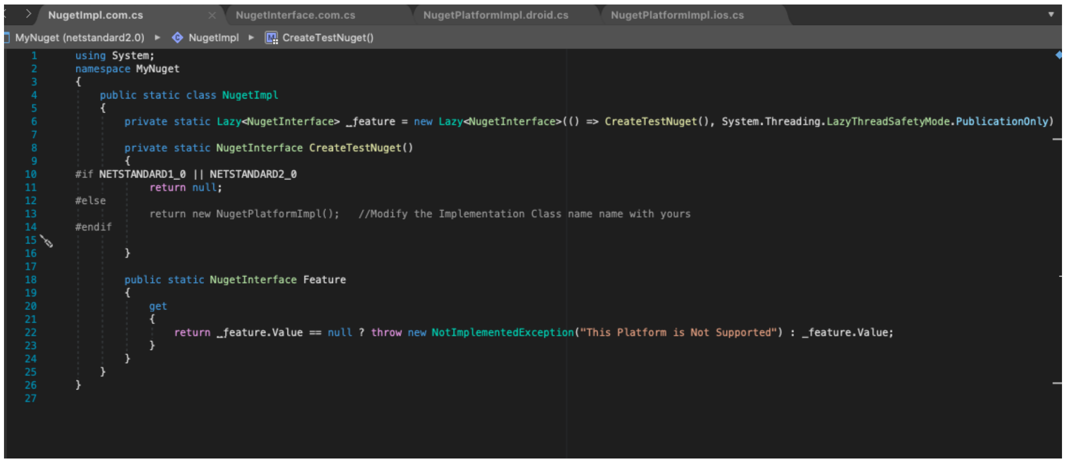
       private static Lazy<*Interfacename*> _feature = new Lazy<*Interfacename*>(() => CreateTestNuget(), System.Threading.LazyThreadSafetyMode.PublicationOnly);

    2. Implement the static method mentioned in the above line which will create the instance for the interface based on target framework. Use precompiled derivatives. Remember here the NETSTANDARD_$(VERSION) should match the one you are targeting as per the csproj modification in point 2 above.

```
static Interfacename CreateTestNuget() {
#if NETSTANDARD1_0 || NETSTANDARD2_0
            return null;
#else
             return new ImplementationClassname();   //Modify the Implementation
Class name name with yours
#endif
}
```

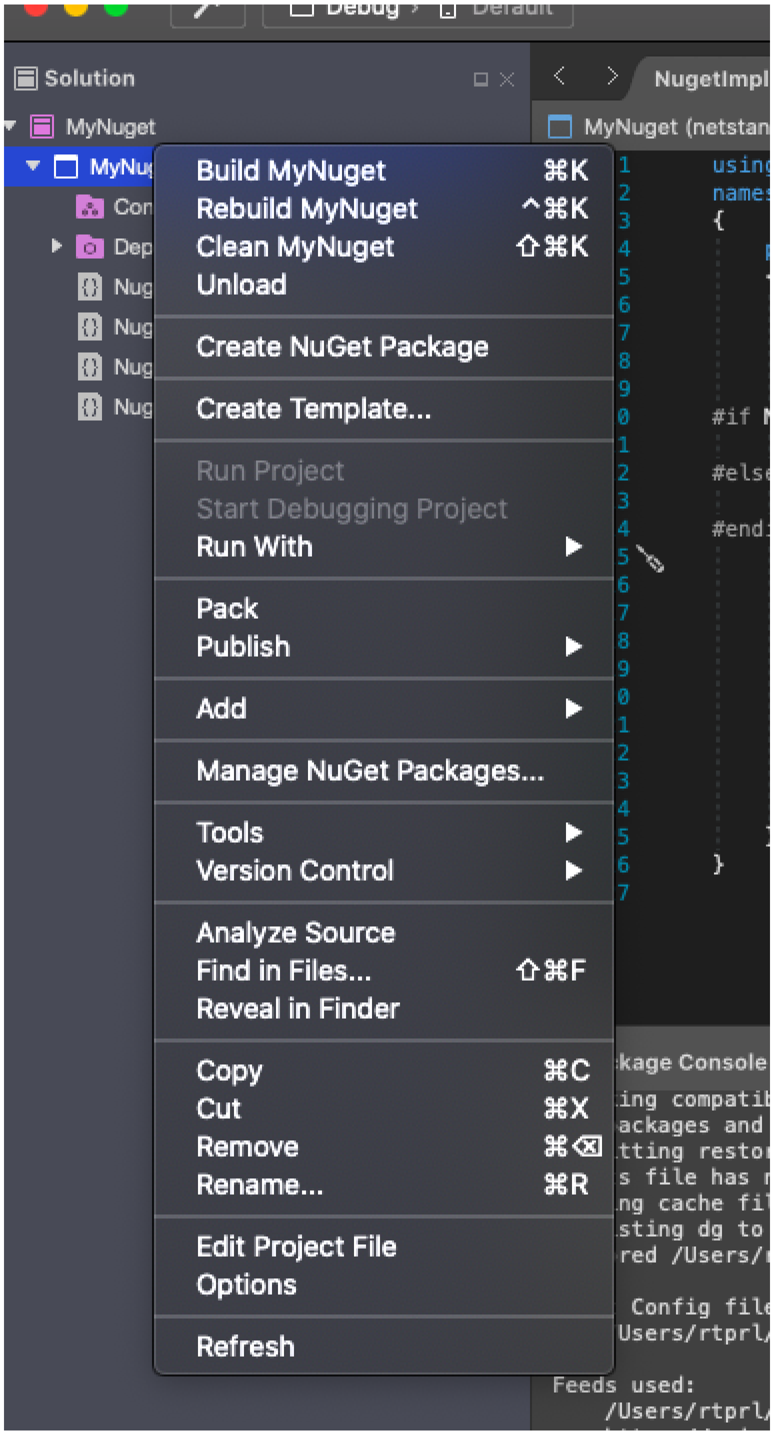
    3. Now create a public member to access the implementation and use the methods.

```
public static Interfacename Feature
        {
            get
            {
                                    return _feature.Value == null ? throw new
NotImplementedException("This Platform is Not Supported") : _feature.Value;
            }
        }
```


NugetImpl.com.cs
NugetInterface.com.cs
NugetPlatformImpl.droid.cs
NugetPlatformImpl.ios.cs
MyNuget (netstandard2.0) ▸ NugetImpl ▸ CreateTestNuget()

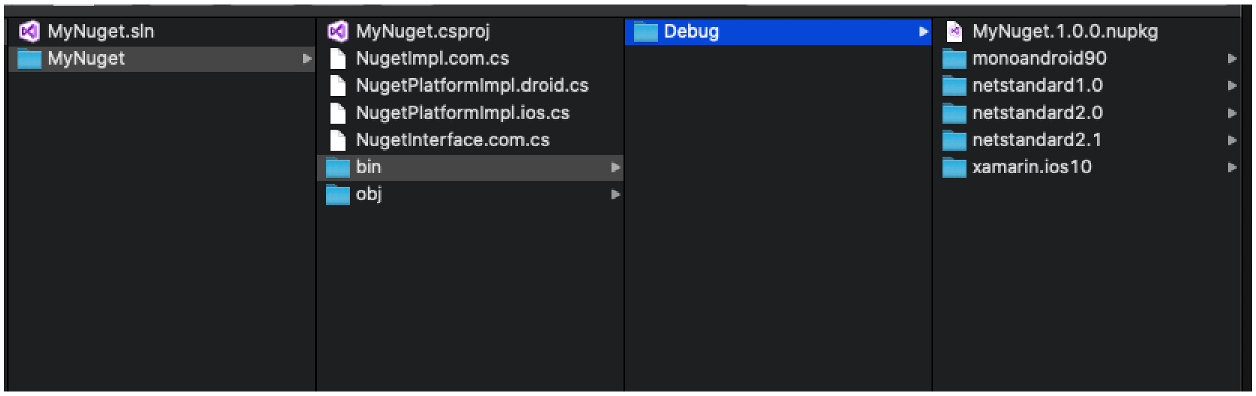
```
using System;
namespace MyNuget
{
    public static class NugetImpl
    {
        private static Lazy<NugetInterface> _feature = new Lazy<NugetInterface>(() => CreateTestNuget(), System.Threading.LazyThreadSafetyMode.PublicationOnly)

        private static NugetInterface CreateTestNuget()
        {
#if NETSTANDARD1_0 || NETSTANDARD2_0
            return null;
#else
            return new NugetPlatformImpl();   //Modify the Implementation Class name name with yours
#endif

        }

        public static NugetInterface Feature
        {
            get
            {
                return _feature.Value == null ? throw new NotImplementedException("This Platform is Not Supported") : _feature.Value;
            }
        }
    }
}
```

6. You are ready. Go ahead and clean, rebuild your project. Your nugget will be created. To change versions while development just modify the PackageVersion in csproj file. Now go ahead and add it to any project and check in android and iOS if it's returning the correct string or not. To add this package or any local nuget packages to NuGet repo for your project, do this…..
    1. Open nuget package manager.
    2. Click on the left top drop down and select Configure sources.
    3. Click on Add and give name and folder location where the nugget is generated, click ok.
    4. Close nuget package manager and reopen the new repo name should show in the left top dropdown.
    5. Remember the Nuget name you will find will be actually the package id you did put in csproj and while using in your project the namespace by which you will use is the root namespace or in simple words the nuget solution name you had. So try to keep the rootnamespace, packageid, Assembly name same as the solution name.
    6. Or if you want to do this in a fancy way follow this.



```
using
names
{
#if N
#else
#endi
}
```

```
ing compatib
ackages and
tting resto
s file has
ng cache fi
sting dg to
red /Users/

 Config file
Users/rtprl/

Feeds used:
    /Users/rtprl/
```



SPECIAL NOTES

1. **Now this is your basic simplest nuget structure ready but if you have other dependencies to add to your nuget and that too platform specific you can easily do so, follow this and check the add dependency section.**.
2. **If you are getting an error like could not find Resource.designer.cs error then maybe you need to upgrade .netcore runtime, xamarin.android sdk. Usually I will recommend doing all the stable release updates that visual studio suggests.**

Sample codebase can be found here.

That's all Folks. I hope this helps. Feel free to shoot questions. Thanks for reading. Happy Coding.