# NuGet in C# projects

This page describes how to deal with NuGet packages within C# projects.

In spite of there is quite large [NuGet documentation](https://docs.microsoft.com/en-us/nuget/) (including shorten [Quick Start](https://docs.microsoft.com/en-us/nuget/quickstart/create-and-publish-a-package)), this does not cover straightforward manual how to deal with NuGets during the whole Continuous Integration process. This page tries to fill this gap.

Contents

[NuGet in C# projects 1](#_Toc514747893)

[Intro 1](#_Toc514747894)

[Examples 1](#_Toc514747895)

[NuGet usage prerequisities 1](#_Toc514747896)

[Consume NuGet packages 1](#_Toc514747897)

[Generate NuGet packages 1](#_Toc514747898)

[Generate NuGet package for several platforms 1](#_Toc514747899)

[Multi-targets in the new "Microsoft.NET.Sdk" csproj file format 1](#_Toc514747900)

[Multi-targets in the "classic" csproj file format 1](#_Toc514747901)

[Publish NuGet packages 1](#_Toc514747902)

[Shared folder for hosting NuGet packages 1](#_Toc514747903)

[Further reading 1](#_Toc514747904)

# Intro

Three **basic activities with NuGet packages** are discussed here:

* Consume NuGet packages - i.e. how to refer to the NuGet packages from the project and how to download them from the NuGet host to the local computer
* Generate NuGet packages - i.e. how to create a NuGet package containing the assembly(ies) which are build out of the C# project
* Publish NuGet packages - i.e. how to publish generated NuGet package into the specific NuGet host

These activities are described from **the different point of views**:

* Used tool point of view - What needs to be done if the NuGet package is Consumed / Generated / Published
  + from the Visual Studio
  + outside of the Visual Studio (e.g. on the CI server such as Jenkins)
* Used Project format - What needs to be done if the NuGet package is Consumed / Generated / Published
  + with the new "Microsoft.NET.Sdk" csproj file format
  + with the "classic" csproj file format

(see the page "New "Microsoft.NET.Sdk" csproj file format - comparison, advantages, disadvantages" for details on project formats)

# Examples

The text below refers to the live examples.

#### Source codes

The examples source codes can be found in the **GIT repository** <https://github.com/net-ba/NuGetInCSharp/tree/master/example>

* The folder "src/LoggingComponent" contains examples of two csproj files with the new "Microsoft.NET.Sdk" csproj file format
* The folder "src/LoggingComponent.csProjToolsVersion15" contains examples of two csproj files with the "classic" format

#### Source code building and related NuGet packages consuming, generating and publishing

The projects located in the GIT repository (as soon as you clone the repository) can be built (and the related NuGet packages be consumed, generated and published) by the Visual Studio on your local computer.

# NuGet usage prerequisities

|  |  |  |
| --- | --- | --- |
|  | **the new "Microsoft.NET.Sdk" csproj file format** | **the "classic" csproj file format** |
| **from the Visual Studio** | install Visual Studio 2017 | install Visual Studio 2013 or 2015 or 2017 |
| **outside of the Visual Studio  (e.g. on the CI server such as Jenkins)** | * install Visual Studio Build Tools 2017  via [Visual Studio Installer](https://www.visualstudio.com/thank-you-downloading-visual-studio/?sku=BuildTools&rel=15) * Install  .NET Core Sdk (<https://www.microsoft.com/net/download/core>)   + For details see "[Get Started with .NET Core - Install for Windows - Command Line](https://www.microsoft.com/net/core#windowscmd)")   + (not sure if really needed) Set env. Variable "MSBuildSDKsPath" to the value like "C:\Program Files\dotnet\sdk\1.0.4\Sdks" | Download the newest nuget.exe (it was version 4.1 in April 2017) - <https://dist.nuget.org/index.html> |

(related official documentation: <https://docs.microsoft.com/en-us/nuget/guides/install-nuget>)

# Consume NuGet packages

 Let's assume there is C# project which depends on some external NuGet package (e.g. on [NLog NuGet package](https://www.nuget.org/packages/NLog/)).

To be able to compile the C# project, the external NuGet package needs to be downloaded into the computer where the project is built.  
This operation is called **Restore**.

|  |  |  |
| --- | --- | --- |
|  | **the new "Microsoft.NET.Sdk" csproj file format** | **the "classic" csproj file format** |
| **(prerequisites)** | the project refers to given external NuGet package via Dependencies  d59e8607fd0daa3840660adf89f92e70 | the project refers to given external NuGet package   * it has to contain "packages.config" file * csproj has to refer to the downloaded dll in "packages" subfolder   3282314554db5192e295e4d4e6ccf65b |
| **(examples)** | [LoggingComponent/Kistler.Example.NLogService](https://github.com/net-ba/NuGetInCSharp/tree/master/example/src/LoggingComponent/Kistler.Example.NLogService)   * Kistler.Example.NLogService.csproj | [LoggingComponent.csProjToolsVersion15/Kistler.Example.NLogService/](https://github.com/net-ba/NuGetInCSharp/tree/master/example/src/LoggingComponent.csProjToolsVersion15/Kistler.Example.NLogService)   * Kistler.Example.NLogService.csproj * packages.config |
| **from the Visual Studio** | see the chapter "Enabling and disabling package restore" in the [NuGet documentation](https://docs.microsoft.com/en-us/nuget/consume-packages/package-restore).  e5972ba72b3777c54c74df2f8cf23465 | the same as at the new "Microsoft.NET.Sdk" csproj file format. |
| **outside of the Visual Studio  (e.g. on the CI server such as Jenkins)** | dotnet.exe restore <path to \*.sln>  Example for Jenkins:  eeb189444e2d15985fa4fe1b846f19f1 | nuget.exe restore <path to \*.sln>  Example for Jenkins:  152bc6292a7e038620b6f2da6d4463d4 |

(related official documentation: <https://docs.microsoft.com/en-us/nuget/consume-packages/overview-and-workflow>)

# Generate NuGet packages

Let's assume there is C# project which produces some assembly (dll) we would like to share with others as a NuGet package.

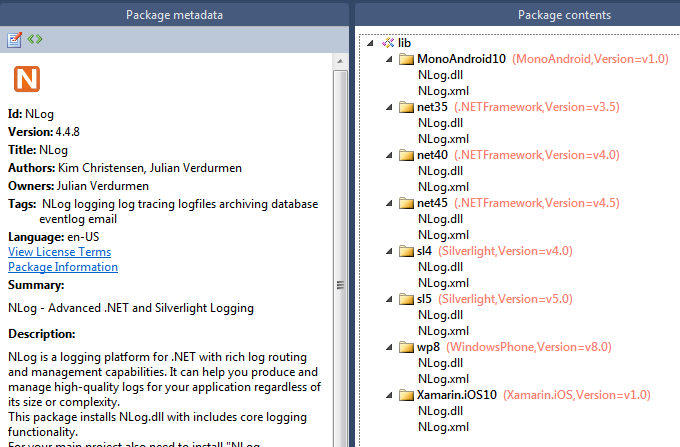
To create such NuGet package, following needs to be done.

|  |  |  |
| --- | --- | --- |
|  | **the new "Microsoft.NET.Sdk" csproj file format** | **the "classic" csproj file format** |
| **(prerequisites)** | The NuGet metadata are filled in the csproj file and  the check box "Generate NuGet package on build" is checked | 1. The NuGet metadata are filled in the csproj file, nuspec file and AssemblyInfo.cs file     2. A post build step to generate NuGet package needs to be established in the project  0bf88a7edf0a74c1c03d78b14dc40605  nuget pack $(ProjectPath) -Prop Platform=$(PlatformName) -Prop Configuration=$(ConfigurationName) |
| **(examples)** | [LoggingComponent/Kistler.Example.NLogService](https://github.com/net-ba/NuGetInCSharp/tree/master/example/src/LoggingComponent/Kistler.Example.NLogService)   * Kistler.Example.NLogService.csproj | [LoggingComponent.csProjToolsVersion15/Kistler.Example.NLogService/](https://github.com/net-ba/NuGetInCSharp/tree/master/example/src/LoggingComponent.csProjToolsVersion15/Kistler.Example.NLogService)   * Kistler.Example.NLogService.csproj * Kistler.Example.NLogService.nuspec * AssemblyInfo.cs |
| **from the Visual Studio** | Nothing special, the NuGet package is generated as part of the success build of the project.  Be aware there is **difference between Build and Rebuild** in the Visual Studio 2017. Whereas **Build** operation generated NuGet package(s), the **Rebuild** operation does not. | Nothing special, the NuGet package is generated as part of the success build of the project |
| **outside of the Visual Studio  (e.g. on the CI server such as Jenkins)** | Nothing special, the NuGet package is generated as part of the success build of the project  dotnet.exe build --configuration "Release" src/LoggingComponent/LoggingComponent.sln  Example for Jenkins:  16d90df8a80c245727ca563796dd0d13 | Nothing special, the NuGet package is generated as part of the success build of the project  Example for Jenkins:  3b467411db24497d2aef6c7e506bd888 |

(related official documentation: <https://docs.microsoft.com/en-us/nuget/create-packages/overview-and-workflow>)

## Generate NuGet package for several platforms

Single NuGet package can contain (and usually contains) assembles (DLLs) for several different platforms.

For instance, NLog.4.4.8 NuGet package contains assemblies for different versions of .NET Framework, for Mono, for Xamarin and so on.  


### Multi-targets in the new "Microsoft.NET.Sdk" csproj file format

To make your C# project be able to target several different platforms is easy to achieve with the new "Microsoft.NET.Sdk" csproj file format.  
Only thing you need to do is to list the platforms you would like to target into the csproj file, into the tag "<TargetFrameworks>".  
In the example below, the two frameworks are targeted:  ".NET Standard 1.3" and "Full .NET Framework 4.6.2".

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

<PropertyGroup>

<TargetFrameworks>netstandard1.3;net462</TargetFrameworks>

</PropertyGroup>

</Project>

 That's all.

If you build such csproj, it generates several DLLs for you and creates NuGet package containing all these DLLs in proper structure.  
The previous example with the two targets resuls in this build output and this NuGet package:

|  |  |
| --- | --- |
| dfdfa6bf4455cd6be04c750f9fae2f0d | 74203de044b7a3e93ccfde479c053e1d |

Here are the links with further information:

* [Targeting multiple .NET platforms in a single NuGet package with Visual Studio 2017](https://www.bartwolff.com/Blog/2017/03/20/targeting-multiple-net-platforms-in-a-single-nuget-package-w)
* **The list of** the possible targeted **platforms** can be found [here](https://docs.microsoft.com/en-us/nuget/schema/target-frameworks) or [here](https://docs.microsoft.com/en-us/dotnet/standard/frameworks).

### Multi-targets in the "classic" csproj file format

In case of the "classic" csproj file format, be able to target several different platforms, there is not much support, almost everything needs to be done manually.

For details see the documentation "[Supporting multiple .NET framework versions](https://docs.microsoft.com/en-us/nuget/create-packages/supporting-multiple-target-frameworks)".

# 

# Publish NuGet packages

Let's assume there is a NuGet package available which was generated out of a C# project.

Now, it is time to share this NuGet package with other developers.

Beside the possibility to share it in the public [nuget.org](http://nuget.org) server (see [this documentation](https://docs.microsoft.com/en-us/nuget/create-packages/publish-a-package) for details), it is possible **to host the package on a private server** (related official documentation: <https://docs.microsoft.com/en-us/nuget/hosting-packages/overview>).

Also for private NuGet package hosting there is several possibilities, nevertheless, here we will show only one simple example.

## Shared folder for hosting NuGet packages

The solution does not differ between the new "Microsoft.NET.Sdk" and  the "classic" csproj file format.

The solution is based on a call of the NuGet.exe CLI, so there is also no differ between approach "from" and "outside" the Visual Studio.

There is quite good documentation of this solution available here: "[Local feeds](https://docs.microsoft.com/en-us/nuget/hosting-packages/local-feeds)".

The solution is based on shared network folder (for example: "\\someFileServer\NugetPackages").

This folder contains shared NuGet packages.

Single NuGet packages can be copied into this folder by the NuGet CLI command:

nuget add <.nupkg file> -source \\someFileServer\NugetPackages

**To copy all NuGet packages generated by building process into the shared folder** on the CI server (on Jenkins, for instance), one can use following Windows batch script:

@echo Publishing generated NuGet packages into '\\someFileServer\NugetPackages'

@echo List of available generated NuGet packages:

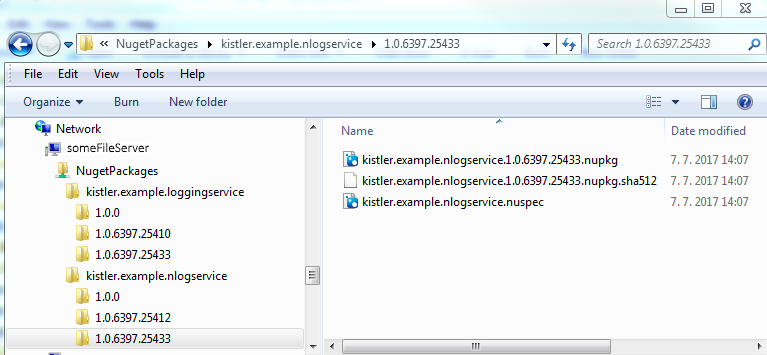
@dir src\LoggingComponent\bin\Release\\*.nupkg /b /s

@echo Publishing start

@for /f %%f in ('dir src\LoggingComponent\bin\Release\\*.nupkg /b /s') do @nuget add %%f -Source \\someFileServer\NugetPackages

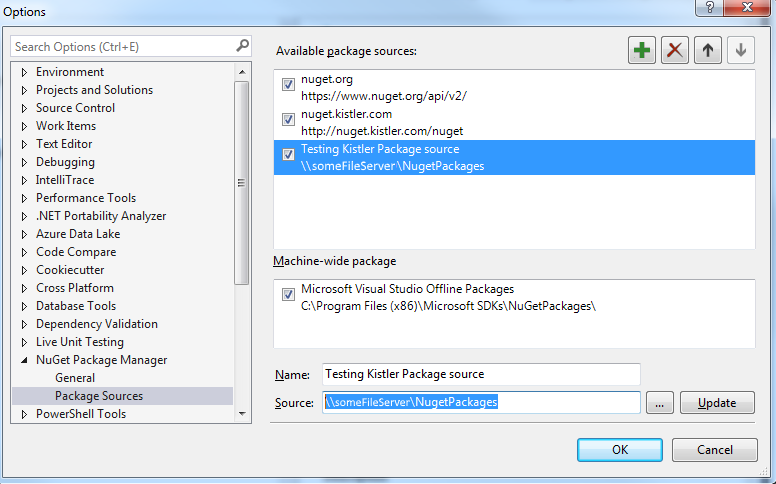
@echo Publishing end

After several builds and publishing results, the **content of this share folder** can look like this:

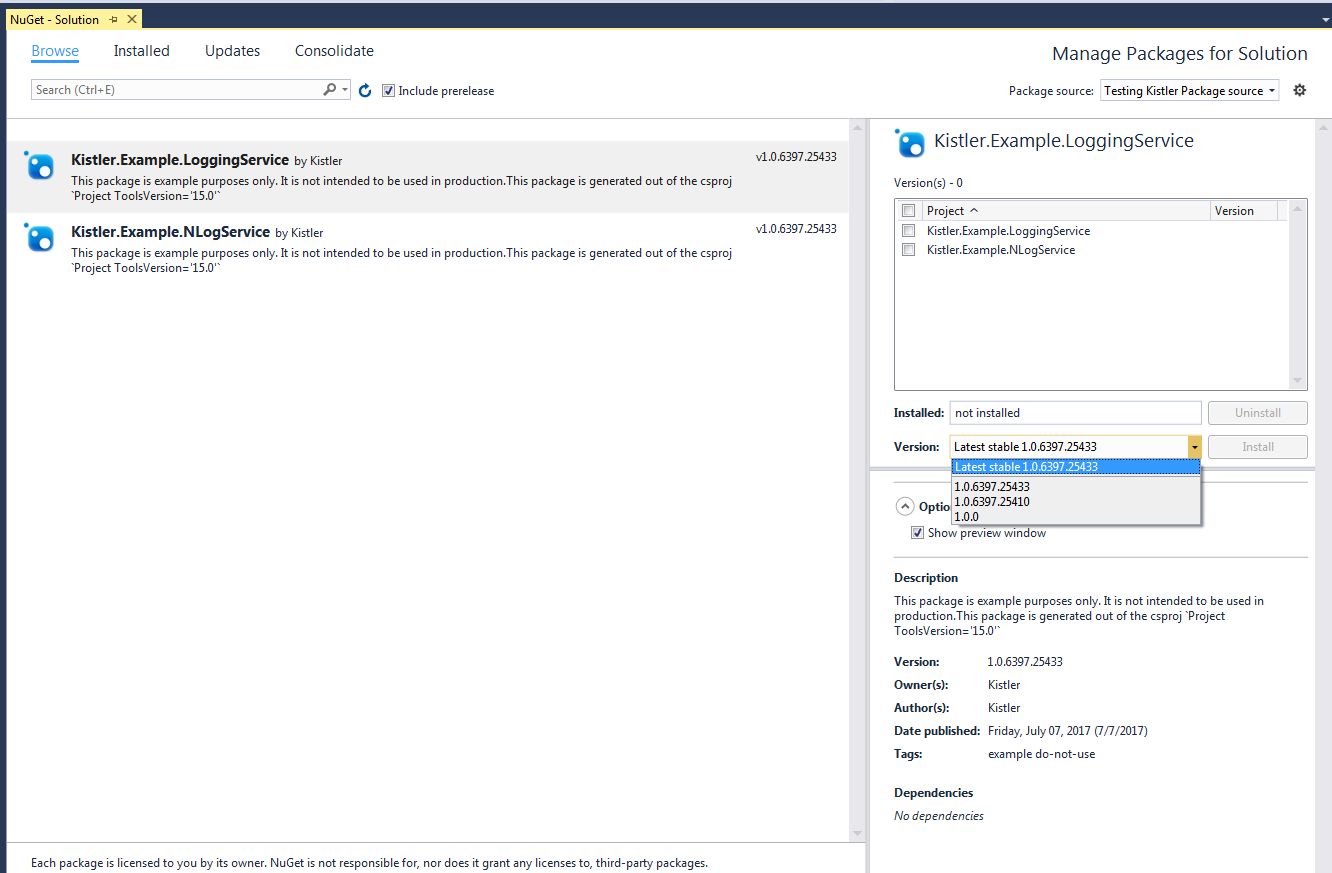


|  |
| --- |
|  |

**To be able to consume the NuGet packages** out of this share folder from Visual Studio, the NuGet Package Manager in the Visual Studio needs to be configured properly:



Now, it is possible in any of your solution to consume any NuGet package published in the shared folder.  
You can browse and install the packages via NuGet Package Manager for Solution in the Visual Studio:



# Further reading

* Documentation root: <https://docs.microsoft.com/en-us/nuget/>
* Quick start: <https://docs.microsoft.com/en-us/nuget/quickstart/create-and-publish-a-package>
* Create Package: <https://docs.microsoft.com/en-us/nuget/create-packages/creating-a-package>
* Publishing packages: <https://docs.microsoft.com/en-us/nuget/create-packages/publish-a-package>
* Hosting your own NuGet feeds: <https://docs.microsoft.com/en-us/nuget/hosting-packages/overview>
* Command line reference: <https://docs.microsoft.com/en-us/nuget/tools/nuget-exe-cli-reference>
* Targeting multiple .NET platforms in a single NuGet package with Visual Studio 2017: <https://www.bartwolff.com/Blog/2017/03/20/targeting-multiple-net-platforms-in-a-single-nuget-package-w>
* NuGet is now fully integrated into MSBuild: <http://blog.nuget.org/20170316/NuGet-now-fully-integrated-into-MSBuild.html>
* NuGet pack and restore as MSBuild targets: <https://docs.microsoft.com/en-us/nuget/schema/msbuild-targets>
* Installing and reinstalling packages with package restore: <https://docs.microsoft.com/en-us/nuget/consume-packages/package-restore>
* How to build and host a NuGet package using VS2017 and .NET Standard in VSTS: <https://liftcodeplay.com/2017/03/23/how-to-build-and-host-a-nuget-package-using-vs2017-and-net-standard-in-vsts/>
* Use Jenkins to restore and publish packages: <https://www.visualstudio.com/en-us/docs/package/build/jenkins>
* Building a .Net 4.6.1 project with Jenkins: <http://www.andyfrench.info/2015/12/building-net-461-project-with-jenkins.html>
* Build & deploy ASP.NET Core applications with Jenkins and Octopus Deploy: <https://www.benscobie.com/build-deploy-asp-net-core-applications-with-jenkins-and-octopus-deploy/>
* Installing and reinstalling packages with package restore - Migrating to automatic restore: <https://docs.microsoft.com/en-us/nuget/consume-packages/package-restore#migrating-to-automatic-restore>