# New "Microsoft.NET.Sdk" csproj file format - comparison, advantages, disadvantages

Together with .NET Core, Microsoft introduced a **new format of the csproj file**.

This page compare this new format with the original / "class" csproj format, lists advantages and disadvantages of the new format and refers articles worth to read on this topic.

Contents

[New "Microsoft.NET.Sdk" csproj file format - comparison, advantages, disadvantages 1](#_Toc514686418)

[Intro 1](#_Toc514686419)

[Examples 1](#_Toc514686420)

[Advantages / Disadvantages of the new "Microsoft.NET.Sdk" csproj file 1](#_Toc514686421)

[Further reading 1](#_Toc514686422)

# Intro

Together with .NET Core, Microsoft introduced a **new format of the csproj file**.

* Its structure is XML (its temporally used Json structure was rejected).
* its root XML tag is <Project Sdk="Microsoft.NET.Sdk"> ... </Project>
* this csproj file format can be used from the Visual Studio 2017

There is still used and usable the original, a **"clasic" format of the csproj file**

* Its structure is XML
* its root XML tag is <Project ToolsVersion="15.0" xmlns="<http://schemas.microsoft.com/developer/msbuild/2003>"> ... </Project>
* this csproj file format can be used with the Visual Studio 2013, 2015 as well as Visual Studio 2017

# Examples

Here are **examples of the "classic" and new format of the csproj file**. In spite of the new one is really short, both examples have the same functionality.

Precisely speaking, the new one has one feature in addition which is not achievable in the old one  - the new one is able to generate assemblies for **more targeted frameworks** (in this example it is ".NET Standard 1.3" and "Full .NET Framework 4.6.2")

|  |  |
| --- | --- |
| **the "clasic" structure of the csproj file**  <?xml version="1.0" encoding="utf-8"?>  <Project ToolsVersion="15.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">  <PropertyGroup>  <Configuration Condition=" '$(Configuration)' == '' ">Debug</Configuration>  <Platform Condition=" '$(Platform)' == '' ">AnyCPU</Platform>  <ProjectGuid>{E5F3BBEB-7AF9-4646-BA74-52914E1F0E50}</ProjectGuid>  <OutputType>Library</OutputType>  <AppDesignerFolder>Properties</AppDesignerFolder>  <RootNamespace>Kistler.Example.LoggingService</RootNamespace>  <AssemblyName>Kistler.Example.LoggingService</AssemblyName>  <TargetFrameworkVersion>v4.6.2</TargetFrameworkVersion>  <FileAlignment>512</FileAlignment>  <TargetFrameworkProfile />  </PropertyGroup>  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|AnyCPU' ">  <DebugSymbols>true</DebugSymbols>  <DebugType>full</DebugType>  <Optimize>false</Optimize>  <OutputPath>..\bin\Debug\</OutputPath>  <DefineConstants>DEBUG;TRACE</DefineConstants>  <ErrorReport>prompt</ErrorReport>  <WarningLevel>4</WarningLevel>  </PropertyGroup>  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Release|AnyCPU' ">  <DebugType>pdbonly</DebugType>  <Optimize>true</Optimize>  <OutputPath>..\bin\Release\</OutputPath>  <DefineConstants>TRACE</DefineConstants>  <ErrorReport>prompt</ErrorReport>  <WarningLevel>4</WarningLevel>  </PropertyGroup>  <ItemGroup>  <Reference Include="System" />  <Reference Include="System.Core" />  <Reference Include="System.Xml.Linq" />  <Reference Include="System.Data.DataSetExtensions" />  <Reference Include="Microsoft.CSharp" />  <Reference Include="System.Data" />  <Reference Include="System.Net.Http" />  <Reference Include="System.Xml" />  </ItemGroup>  <ItemGroup>  <Compile Include="ILoggingService.cs" />  <Compile Include="LoggingLevel.cs" />  <Compile Include="Properties\AssemblyInfo.cs" />  </ItemGroup>  <ItemGroup>  <None Include="Kistler.Example.LoggingService.nuspec" />  <None Include="Readme.md" />  </ItemGroup>  </Project> | **the new structure of the csproj file**  <Project Sdk="Microsoft.NET.Sdk">  <PropertyGroup>  <TargetFrameworks>netstandard1.3;net462</TargetFrameworks>  </PropertyGroup>  </Project> |

The **live examples of the old and new csproj file formats**, you can find in the GIT repository <https://github.com/net-ba/NuGetInCSharp/tree/master/example>

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

# Advantages / Disadvantages of the new "Microsoft.NET.Sdk" csproj file

**Advantages:**

* The csproj file is simple
* Multi-target support
* NuGet restoring and generation is integrated inside the csproj file (no redundant specifications of versions, package content, and so on)
  + No "packages.config" file needed
  + No ".nuspec" file needed (so no need to keep it in sync with the csproj file (as regards dependencies for example))
* csproj file can be edited in the Visual Studio without a need to be unload from the Visual Studio
* No redundant info in AssemblyInfo.cs  and csproj file needed
* To build and nuget stuff you can use single tool "dotnet.exe" CLI, need not to use directly "msbuild.exe" and "nuget.exe"

**Disadvantages:**

* If you are building WPF, Universal Windows, Xamarin, or ASP.NET 4 projects, you’ll have to stick with the old format (for now… follow <https://github.com/dotnet/sdk/issues/491> for updates).
  + But, in the solution, you can combine the new and the "classic" csproj files
* No auto-increment version at build (if it is needed, it can be realized by script)

# 

# Further reading

* If you are interested how to convert your csproj file in the "classic" format into the new format,   
  read the article [Old csproj to new csproj: Visual Studio 2017 upgrade guide](http://www.natemcmaster.com/blog/2017/03/09/vs2015-to-vs2017-upgrade/)
* [Organizing your project to support .NET Framework and .NET Core](https://docs.microsoft.com/en-us/dotnet/core/porting/project-structure)  
  This article helps project owners who want to compile their solution against .NET Framework and .NET Core side-by-side. It provides several options to organize projects to help developers achieve this goal.
* [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)
* See also the page “Porting your .NET Framework projects to .NET Core”
* See also the page “NuGet in C# projects”