Debugging, source code and symbols for NuGet packages

MyGet supports NuGet symbols packages to help MyGet users debug their NuGet packages, step through their source code, and integrate with Visual Studio and tools like WinDbg.

MyGet comes with its own symbol server that supports .snupkg and legacy .symbols.nupkg symbols, as well as consuming symbols and source indexing. Both managed and native assemblies and symbols are supported.

Creating symbols packages

Using the nuget pack command, you can create a normal .nupkg and a symbol file for debugging. On MyGet you can push both your NuGet package and its symbol package to the same feed for easier management.

New .snupkg format

To create your symbol package in the new .snupkg format, you must specify the `SymbolPackageFormat` property. The SymbolPackageFormat property can have one of two values: symbols.nupkg (the default) or snupkg. If this property is not specified, a legacy symbol package will be created.

If you're using dotnet CLI or MSBuild, you can specify `SymbolPackageFormat` in your .csproj file.

- Either add this property to your .csproj file:

```
<PropertyGroup>
    <IncludeSymbols>true</IncludeSymbols>
    <SymbolPackageFormat>snupkg</SymbolPackageFormat>
</PropertyGroup>
```

- Or in the command line:

```
dotnet pack MyPackage.csproj -p:IncludeSymbols=true -
p:SymbolPackageFormat=snupkg
```

or

```
msbuild MyPackage.csproj /t:pack /p:IncludeSymbols=true
/p:SymbolPackageFormat=snupkg
```

If you're using the NuGet.exe., you can use the following commands to create a .snupkg file in addition to the .nupkg file by specifying in the .csproj or .nuspec file, including the `-Symbols`

option:

```
nuget pack MyPackage.nuspec -Symbols -SymbolPackageFormat snupkg

nuget pack MyPackage.csproj -Symbols -SymbolPackageFormat snupkg
```

Legacy .symbols.nupkg format

Using the `nuget pack` command, you can create a legacy .symbols.nupkg file with the `-Symbols` option. ***Legacy symbols packages must contain both the .dll and .pdb file, together with the source files***.

Check the NuGet documentation for new .snupkg (https://docs.microsoft.com/en-us/nuget/create-packages/symbol-packages-snupkg) symbols or legacy .symbols.nupkg (https://docs.microsoft.com/en-us/nuget/create-packages/symbol-packages) symbols for more information and an example of a .nuspec file.

Pushing symbols packages to MyGet

When working with NuGet.org (https://www.nuget.org) , the NuGet client automatically recognizes symbols packages and pushes them to NuGet server if they are in the new .snupkg format, or the default SymbolSource feed if they are legacy .symbols.nupkg packages. Since we want to ensure our packages end up on our own feed and securely host debugger symbols, we must explicitly push symbols to the MyGet symbol server.

New .snupkg format

The publish workflow to publish the SamplePackage.1.0.0.nupkg to a MyGet feed with new .snupkg symbols depends on whether your NuGet package and symbol package are located in the same folder (replace the GUID with your MyGet API key).

a) If .nupkg and .snupkg packages are in the same folder, you only need to us `push` command for nupkg (we will detect the .snupkg and push it for you):

```
nuget push SamplePackage.1.0.0.nupkg 00000000-0000-0000-0000-00000000000 -Source h
ttps://www.myget.org/F/somefeed/api/v3/index.json
```

b) If .nupkg and .snupkg packages are in different folders, you will need to manually push both the NuGet package and symbol package separately:

```
nuget push SamplePackage.1.0.0.nupkg 00000000-0000-0000-0000-00000000000 -Source h
ttps://www.myget.org/F/somefeed/api/v3/index.json
nuget push SamplePackage.1.0.0.snupkg 00000000-0000-0000-0000-00000000000 -Source
https://www.myget.org/F/somefeed/api/v3/index.json
```

c) Or if your .nupkg package is already in a MyGet feed and you need to add a .snupkg package:

```
nuget push SamplePackage.1.0.0.snupkg 00000000-0000-0000-0000-00000000000 -Source
```

```
https://www.myget.org/F/somefeed/api/v3/index.json
```

Legacy .symbols.nupkg format

The publish workflow to publish the SamplePackage.1.0.0.nupkg to a MyGet feed with legacy .symbols.nupkg symbols would be issuing the following two commands from the console (replace the GUID with your MyGet API key):

```
nuget push SamplePackage.1.0.0.nupkg 00000000-0000-0000-0000-00000000000 -Source h
ttps://www.myget.org/F/somefeed/api/v2/package
nuget push SamplePackage.1.0.0.Symbols.nupkg 00000000-0000-0000-0000-00000000000 -
Source https://www.myget.org/F/somefeed/symbols/api/v2/package
```

Note: Starting with NuGet.exe 3.5, regular packages and symbols packages can be pushed with one single command:
nuget push SamplePackage.1.0.0.nupkg ooooooo-oooo-oooo-oooo-ooooooooooo -Source https://www.myget.org/F/somefeed/api/v2/package (https://www.myget.org/F/somefeed/api/v2/package) *-SymbolSource https://www.myget.org/F/somefeed/symbols/api/v2/package* (https://www.myget.org/F/somefeed/symbols/api/v2/package) *-SymbolApiKey ooooooo-oooo-oooo-oooo-ooooooooooo*

Consuming symbol packages in Visual Studio

To debug a NuGet package for which symbols are available, we will need the symbols URL for use in Visual Studio. After logging in to MyGet, we can find this URL under the ***Feed Details*** tab of our feed.

Tip: MyGet provides two symbol server URLs: one that requires authentication and one that contains an authentication token in the URL. The first will prompt for credentials when used (Visual Studio 2015 and beyond). The latter one will not, as it contains an authentication token. It is recommended to keep the symbols URL to yourself at all time: it's a personal URL in which security information is embedded in the form of a guid. If for some reason this gets compromised, please reset your API key (https://www.myget.org/profile/Me#!/AccessTokens) .

Visual Studio typically will only debug our own source code, the source code of the project or projects that are currently opened in Visual Studio. To disable this behavior and to instruct Visual Studio to also try to debug code other than the projects that are currently opened, use the ***Tools | Options*** menu and find the *Debugging* node. Configure the following options:

- *Enable Just My Code* should be disabled.
- *Enable source server support* should be enabled. This may trigger a warning message but it is safe to just click *Yes* and continue with the settings specified.

Keep the Options dialog open and find the ***Debugging | Symbols*** node on the left. In the dialog shown, add the symbol server URL for your MyGet feed, for example
`https://www.myget.org/F/somefeed/auth/11111111-1111-1111-1111-`

`11111111111/api/v2/symbolpackage` (or for legacy .symbols.nupkg symbols, `https://www.myget.org/F/somefeed/auth/11111111-1111-1111-1111-11111111111/symbols`).

Browsing symbols in MyGet

The package details page for a package that has symbols available comes with a nice utility that enables us to browse the source code embedded in a symbols package.

Quick command cheatsheet

Here's a quick cheatsheet of the commands related to symbol feeds:

New .snupkg symbols

- Create package and new symbols

```
nuget.exe pack <path_to_project_or_nuspec> -Symbols -SymbolPackageFormat snupkg
```

- Pushing a package to MyGet (we will detect the .snupkg and push it):

```
nuget.exe push <package-file> <myget-key> -Source
https://www.myget.org/F/<feed-name>/api/v3/index.json
```

- Pushing a new symbols package to MyGet (if nupkg is already in MyGet feed):

```
nuget.exe push <symbols-package-file> <myget-key> -Source
https://www.myget.org/F/<feed-name>/api/v3/index.json
```

Legacy .symbols.nupkg.symbols

- Create package and symbols

```
nuget.exe pack <path_to_project_or_nuspec> -symbols
```

- Pushing a package to MyGet:

```
nuget.exe push <package-file> <myget-key> -Source
https://www.myget.org/F/<feed-name>/api/v2/package
```

- Pushing a symbols package to MyGet:

```
nuget.exe push <package-file> <myget-key> -Source
https://www.myget.org/F/<feed-name>/symbols/api/v2/package
```

Troubleshooting

The following list of tips might be useful to you if you hit any issues when configuring the debugger. If you have some other tips to share, contact MyGet support or submit a pull request for this page.

Note: MyGet does not index any binaries found in the package's `\tools` folder.

A symbols package was pushed but does not provide source stepping (legacy symbols format)

The way Visual Studio and other debugging tools match an assembly and PDB file is by using the assembly hash. This hash is stored in the `.dll` and `.pdb` file and must match for debugging and source steping to work.

On the package details page on MyGet, we can verify if for a given assembly debugging and source stepping is possible. MyGet shows the assembly name, the assembly hash and whether source stepping will be available for it or not.

Note: To verify this match when creating packages on our system, we can use the ChkMatch (https://www.debuginfo.com/tools/chkmatch.html) tool. The author of this tool also provides a comprehensive article about matching assemblies and symbols (https://www.debuginfo.com/articles/debuginfomatch.html) .

In the Visual Studio Debugger options, you can disable the *Require source files to exactly match the original version* toggle if you are confident the `.dll` and `.pdb` can be different. We advise against this as the debugging symbols that will be loaded come from a different build than the one you are tying to debug, but it can be a last resort to get at least an idea of the source code for the assembly being debugged.

Portable PDB hashes will look different from Windows PDB hashes, because of the SSQP Key Conventions for Portable PDB's (https://github.com/dotnet/symstore/blob/master/docs/specs/SSQP_Key_Conventions.md#portable-pdb-signature) .

Note for users generating Portable PDB's (.NET Standard / .NET Core): MyGet *can* serve the `.pdb` file to Visual Studio, Rider, WinDBG etc., but we can not rewrite the PDB file to allow linking to source code. Portable PDB's work differently in this regard. To support source stepping, we recommend using SourceLink (https://github.com/dotnet/sourcelink/blob/master/README.md) .

The source files are still in their original location

When stepping into source code, Visual Studio or WinDbg uses the `.pdb` file to link the assembly with corresponding code. MyGet indexes the `.pdb` file after uploading a package. This indexing process adds a second path to the `.pdb` file, telling Visual Studio where to find the source files.

If the source code is still on the machine where a symbols package was created, and we try to debug on that machine, the `.pdb` file that is downloaded will reference both the local path to

sources as well as the path on MyGet. Since the sources are still available locally, Visual Studio will never reach out to MyGet to download source code files. If this is undesired, make sure to rename the local folder containing the original sources (or use a different machine to perform debugging).

Make sure to push both .nupkg and .snupkg (or .symbols.nupkg) to your feed

Symbols packages contain the `.pdb` files that link the assembly with source code. When only `.symbols.nupkg` packages are pushed to a feed, we can consume the package like any normal package and MyGet will properly recognize the package as a symbols package. When trying to debug using such package, Visual Studio will find the `.pdb` on disk instead of reaching out to MyGet to download it, and will fail stepping into code because of that.

The solution here is to always push `.nupkg` as well as `.symbols.nupkg` packages to a feed so that Visual Studio has to reach out to MyGet for fetching debugging information.

Make sure the .nupkg does not contain .pdb files

When a `.nupkg` contains `.pdb` files, Visual Studio will ***never*** reach out to MyGet to download symbols and sources. When trying to debug using such package, Visual Studio will find the `.pdb` on disk instead of reaching out to MyGet to download it, and will fail stepping into code because of that.

Ensure proper packaging:

- `.nupkg` **must** contain assemblies, content files, ..., but **never** `.pdb` files or a `src` folder.
- `.snupkg` may contain the following file extensions: .pdb (portable), .nuspec, .xml, .psmdcp, .rels, .p7s.
- `.symbols.nupkg` ***may*** contain assemblies, content files, ..., and **must** contain `.pdb` files and a `src` folder.

Veryifying symbols package contents

There is a useful plug-in for NuGet Package Explorer (https://npe.codeplex.com) which allows us to validate our symbols packages.

To install the plug-in, open NuGet Package Explorer and use the ***Tools | Plugin Manager...*** menu. In the dialog that opens, click the ***Add Feed Plugin...*** button double-click the SymbolSource plug-in from the MyGet feed.

This plugin enhances the package analysis tools with additional rules that should help detect 99% of the problems with symbols packages.

Once installed, open a symbols package and validate its contents by selecting `Tools > Analyze Package` or hit `CTRL-Q`.

A common root cause for symbols missing in the symbols package originates from a too restrictive `.nuspec` file. The one below will filter out all non-DLL files from the package.

```
<file src="C:\src\AwesomeLib\bin\Release\AwesomeLib.dll" target="lib\net45" />
```

If you have a nuspec file which contains a similar line as the one above, you might want to change it to the following:

```
<file src="C:\src\AwesomeLib\bin\Release\AwesomeLib.*" target="lib\net45" />
```

The NuGet client tools are smart enough to filter out PDB files from non-symbols packages (unless you explicitly include them).