Nuget Deploy Documentation

Xyanid

Version 1.4.1.0

Table of Contents

Illustration Index

Index of Tables

1 Visual Studio Integration

The tool adds a two new menu commands which are only available when right clicking on a single project. Furthermore a new option will be available under Tools -> Options -> NuGet Deploy, with a number of sub categories, which are explained further down the description. Below is a screenshot showing the menu command.

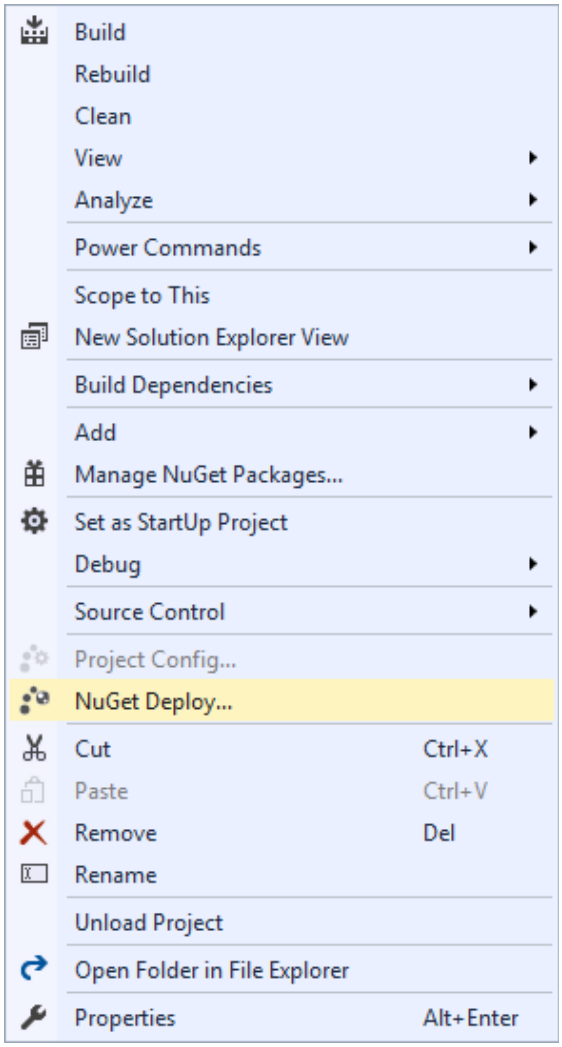

Illustration 1.1: new menu commands

2 Menu Commands

The Menu commands provide the main functionality of the tool, as said there are two commands. Both will open a new Dialogues, which are explained down below.

2.1 Nuget Deploy...

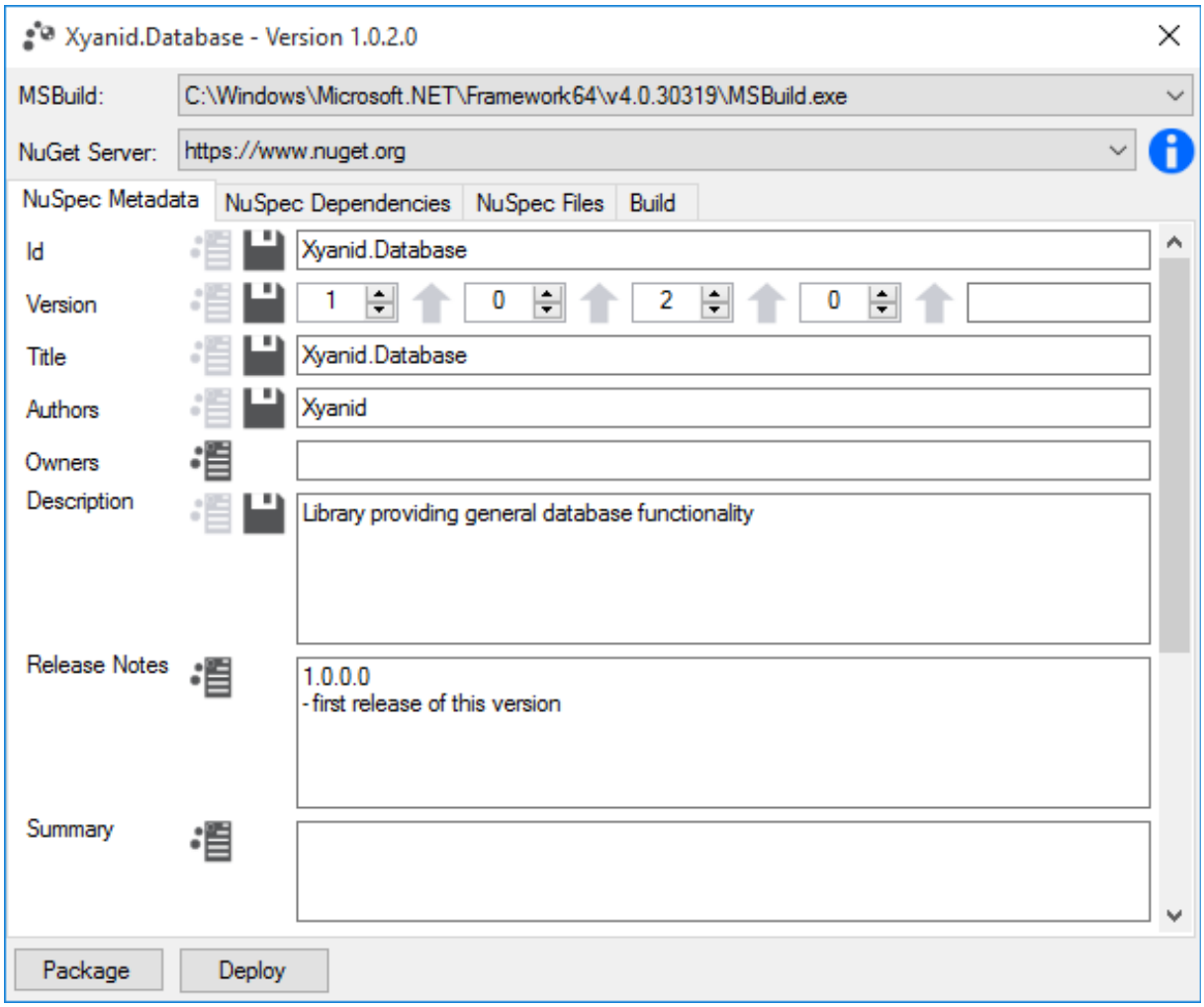

Illustration 2.1: NuGet deploy... Dialogue

This dialogue is shown when selecting the ***NuGet deploy...***, allowing you to select a msbuild.exe which will be used to build the project as well as the NuGet server on which to deploy the package.

If you have not set a NuGet server or msbuild.exe yet, you can choose the **<new...>** entry in the combo box, which will prompt you select a msbuild.exe on your system or provide a NuGet Server. Those information will also be stored in the configuration.

As you can see there are three sub pages which provide an overview of the deployment that is about to take place. The option available in those pages are explained in the following section.

2.1.1 NuSpec Metadata

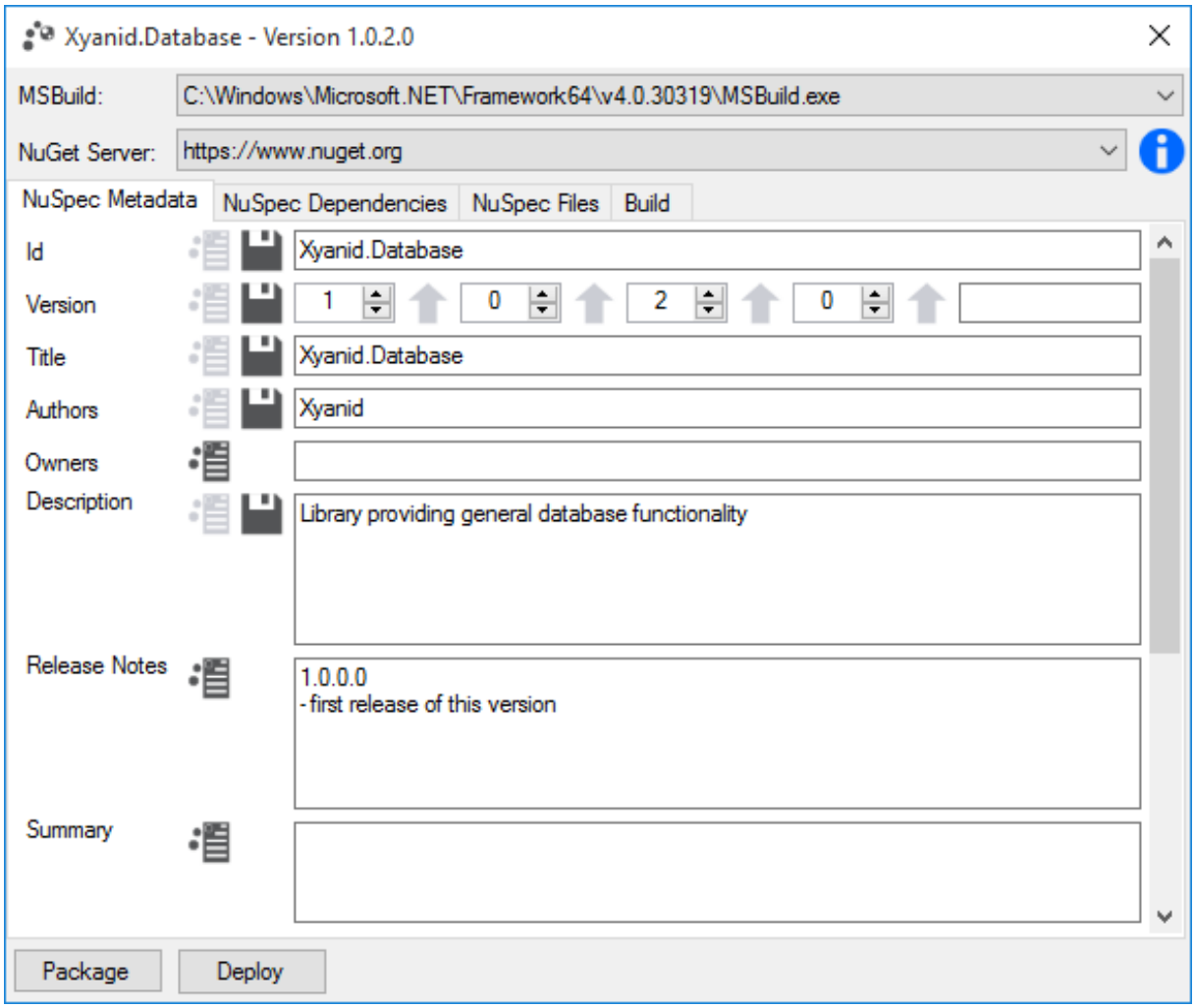

Illustration 2.2: NuSpec Metadata for the project

In this page you can see the actual NuSpec Metadata information that will be used, so you have one last chance to alter them if they do not match.

Each element has two icons which indicate your current settings for this option, e.G if the file looking icon is enabled it means that the information for this element is saved and retrieved from a NuSpec file. If the save icon is enabled it mean that this information is saved back into the project when you deploy or package it.

Furthermore each of the icons has a tooltip which will also provide information about the setting.

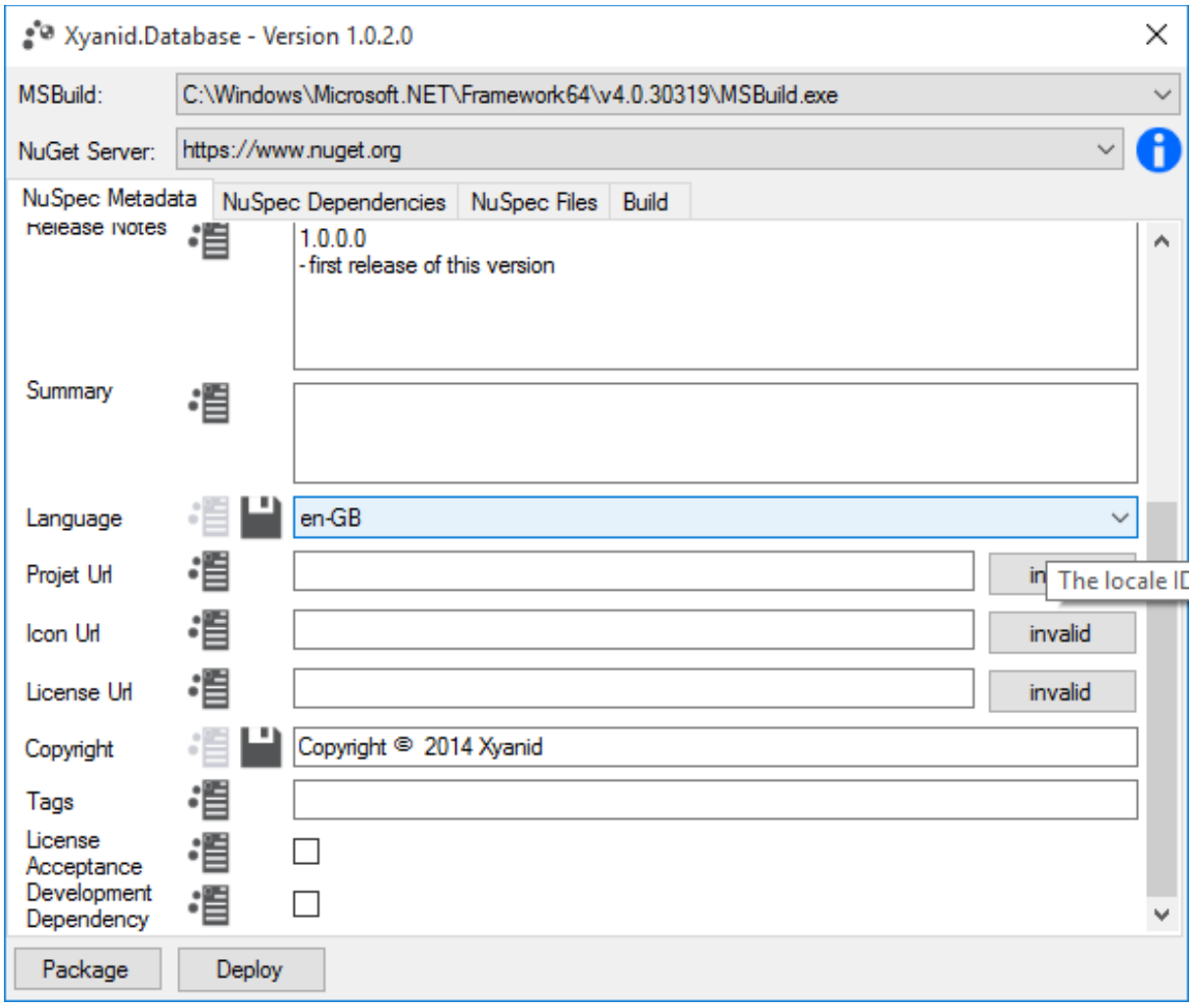

Illustration 2.3: The rest of the NuSpec Metadata for the project

The buttons next to the project URL, icon URL and License URL enable you to make a simple http head check if the value you have entered is also available.

The result will be written in the button:

- invalid → means the URL is invalid because a http error occurred
- valid → means the URL is valid
- checking... → meaning the URL is checked, this is done in a separate thread for each URL to not block the UI.

2.1.1.1 Invalid Value

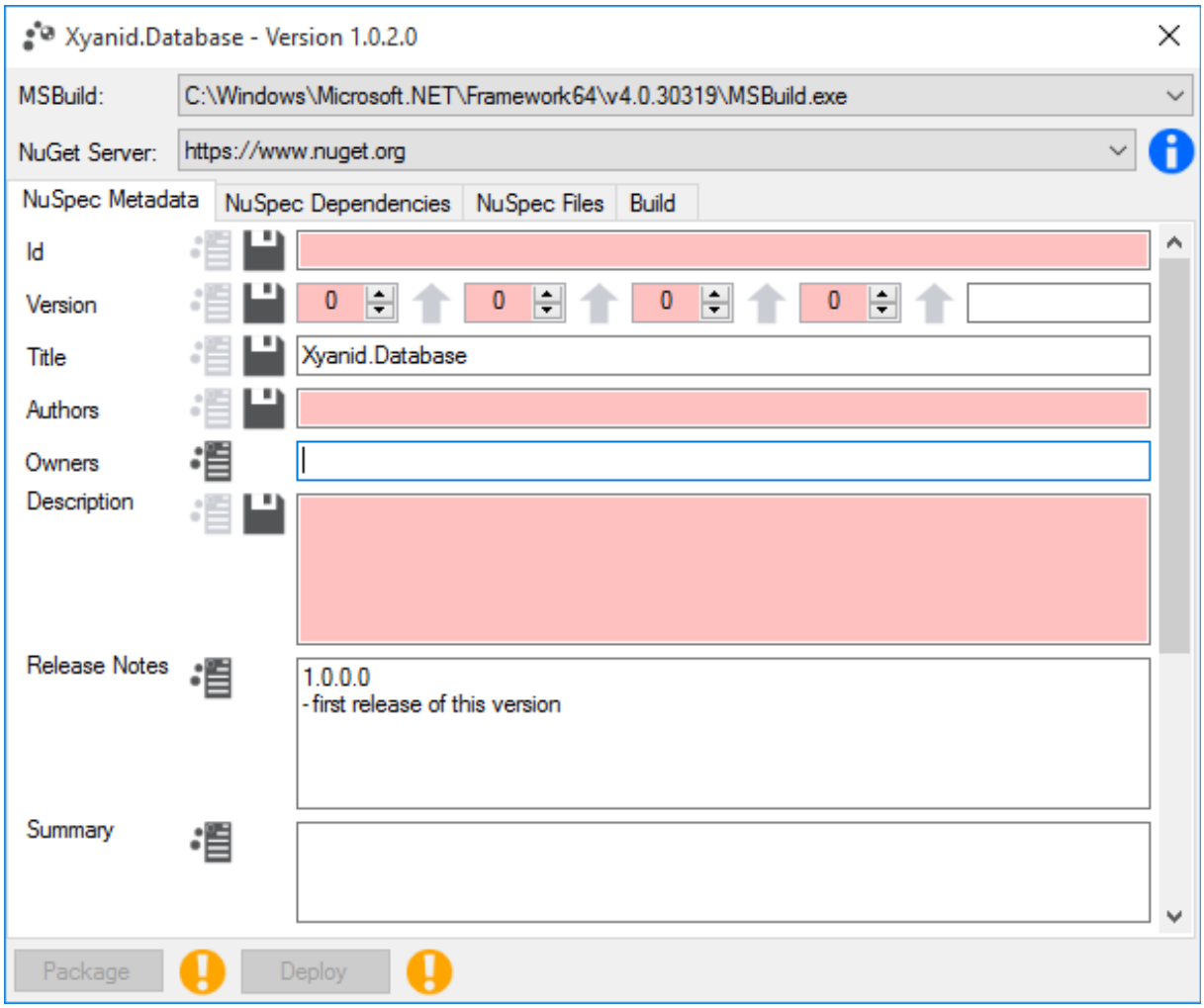

Illustration 2.4: NuGet deploy... dialogue with invalid elements

The fields Id, Version, Authors and Descriptions are mandatory, which means those option need to be provided. If they are not (e.g. they are empty or invalid), then the fields will be marked with a light red background colour, a warning icon will appear in the bottom of the dialogue and the deployment will not be possible as long as the issue is not resolved.

If you hover over the icon, you will also see a tool tip, providing information about what the issue is.

2.1.2 NuSpec Dependencies

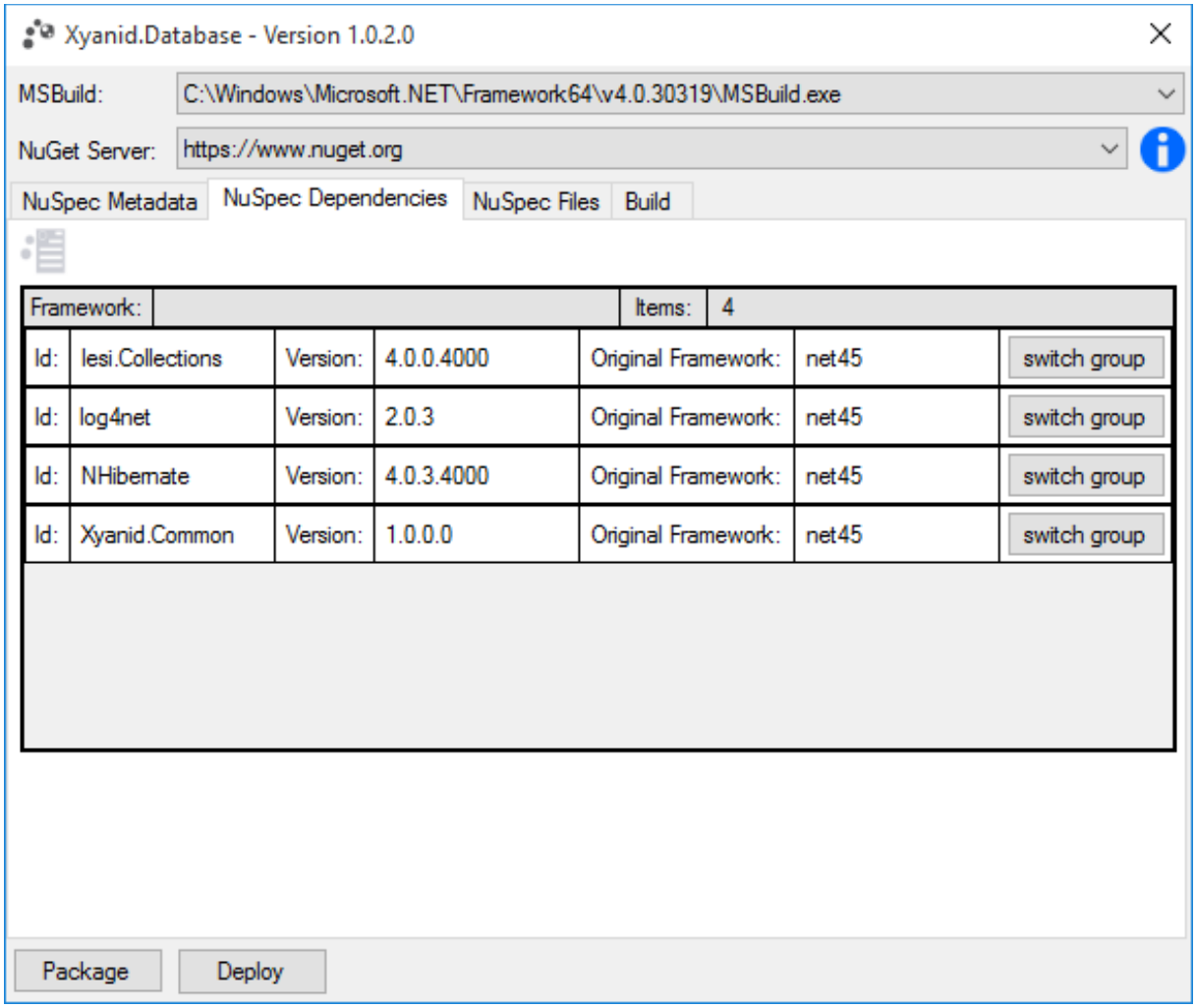

Illustration 2.5: NuSpec Dependencies of the project

This page show you what other NuGet dependencies the project is based on. If the icon on the top left is disabled, then the dependencies are from the package.config of the project. Otherwise the dependencies are from an existing nuspec file, if there is one if not they will still be retrieved from the project and stored in the nuspec file afterwards.

You can also switch the group in which the dependency is currently is. Doing so will either move the dependency into its original framework group (which will be created if there is none), or in the fallback group.

It is recommended that all dependencies are in the same group, otherwise when you install the package the corresponding group is used which might leave some of the dependencies unresolved. E.g. dependency A is in the fallback group and dependency B is in its framework group ***net45***. If the package gets installed into a net45 project then only the net45 group is used, not the fallback group. This might cause problems if dependencies are dependent on each other and must be in the same group.

2.1.3 NuSpec Files

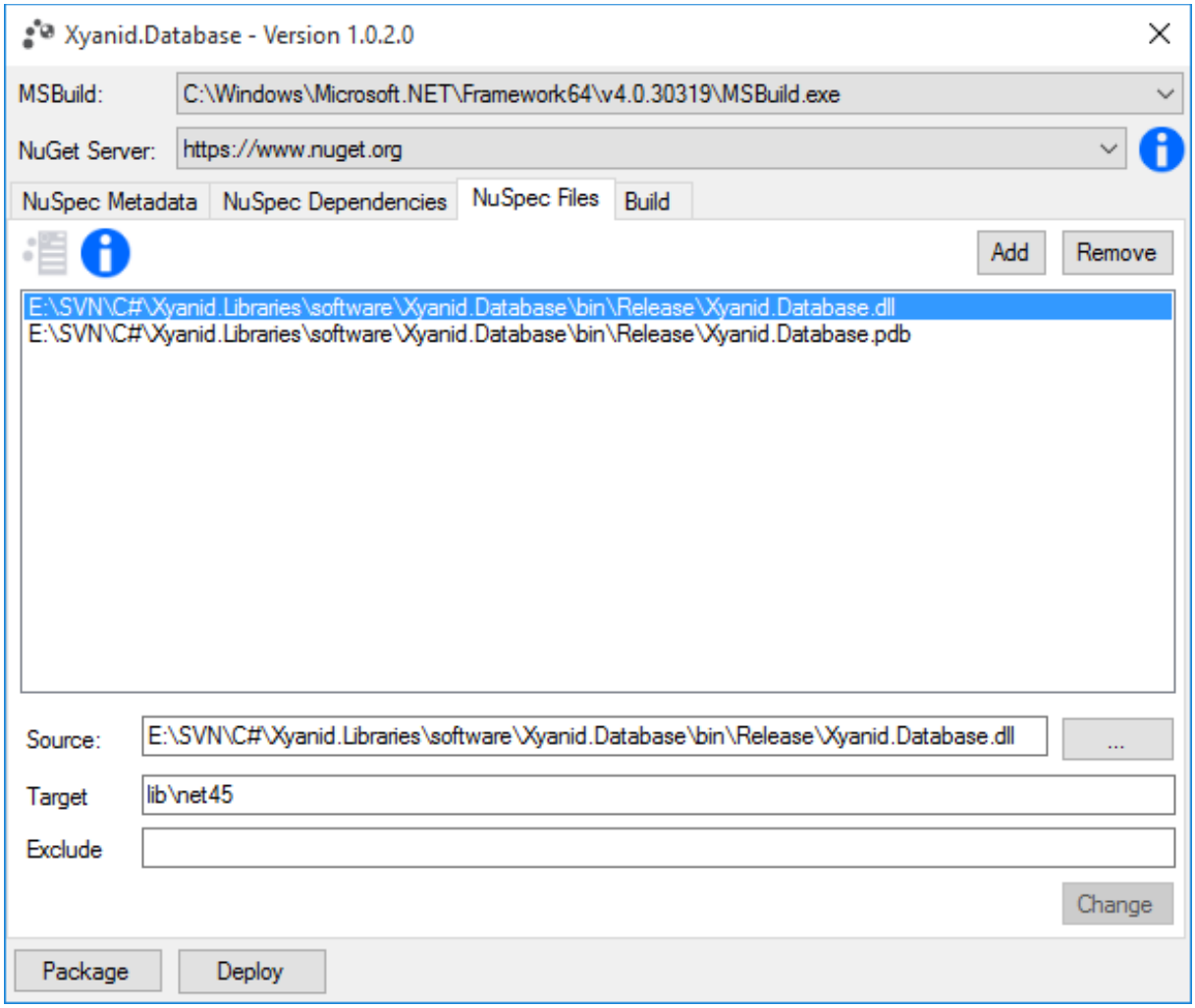

Illustration 2.6: NuGet deploy... dialogue, showing the files contained in the package

In this page you will see the Files, that will be in the package.

The files follow the rules of the NuSpec files element, as such you can also declare a file that excludes certain items or include whole groups of items via a wildcard. For further information about this element, please visit official NuGet website.

The information icon will provide a tool tip about the current file usage configuration as well as the set up file include options (if there area any).

If the project also contains a documentation file, then it will automatically be included as well.

Some files may have been automatically been added because of the NuSpec Files Options.

2.1.4 The Build Options

This page contains the available build options. Again the information icon will provide a tool tip about

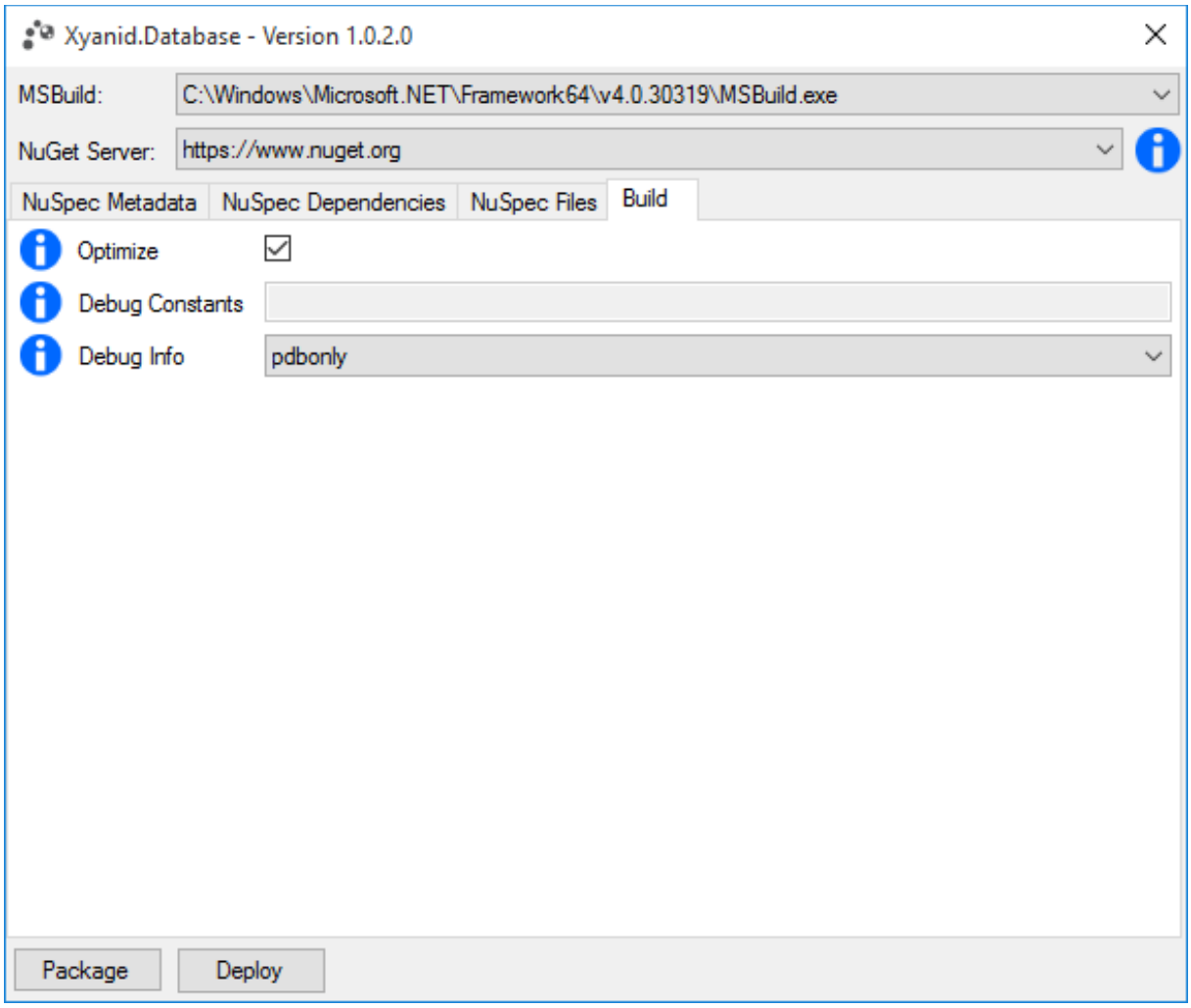

Illustration 2.7: NuGet deploy... dialogue, showing the build configuration

the current build configurations. Depending on you configuration, a build option can already have a value. E.g. if you want to use the projects debug constants, then those constants will be shown.

Note that a particular build option can only be used in the build process if it has not been set to ***do not use*** in the configuration.

Also .pdb files from the project will be added to the NuGet package under the following circumstances:

- you have unchecked the ***use from existing NuSpec*** in the Project – NuGet – NuSpec - Files (thus files will be automatically generated from the project)
- and you have set the debug info option to either ***pdbonly*** or ***full***

2.2 Project Config...

The project configuration option just the normal configuration which is available under Tools → Options → NuGet Deploy, with the difference being that only project related configurations will be shown in the dialogue.

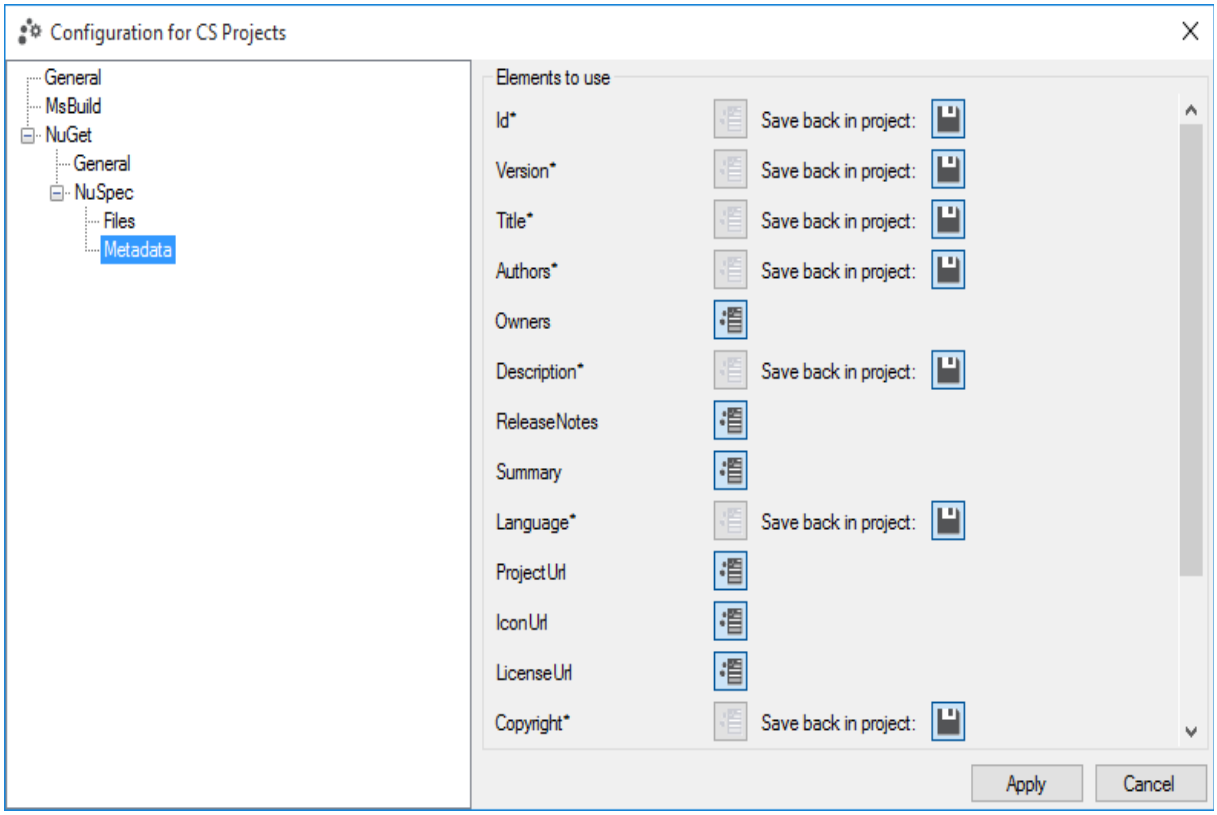

Illustration 2.8: Project Config... dialogue, showing the available project configurations

Hence this option is not available if the project storage option is set to User based for the specific project type. Note that the configurations in this page always refer to the corresponding project, like in the screenshot below, the configurations are for c# projects.

For the available configuration pages see Configuration.

3 Deployment Process

After all information are to you liking, you can deploy the library, which will show the following dialogue.

The deployment process consist of the following steps:

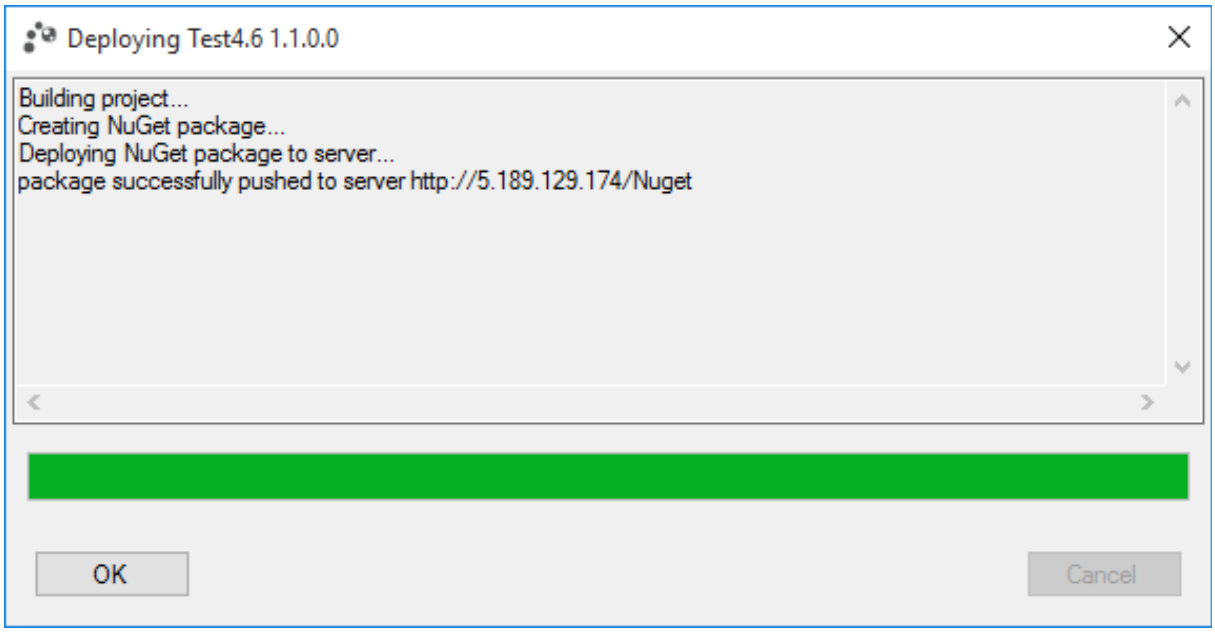

Illustration 3.1: deployment dialogue

1. the build process for the project will be done with the msbuild.exe, that was selected in the Nuget Deploy....
2. the nuget.exe will be used to create a .nupkg file form a .nuspec file which has been created using the provided information from the Nuget Deploy....
3. The final step uses the created .nupkg file to push it to the selected NuGet server.

If any of those steps fail, then the specific error message will be shown in the text box above. The deployment is done in a separate thread, so you can deploy different projects at once (you can however only deploy the same project once, since it is blocked via a mutex during deployment.) or also cancel the deployment any time.

Once deployment is finished or failed, the OK button will be unlocked.

4 Configuration

There are several configurations for the plugin, which are not stored in a configuration file or in th registry, but in a settings.xml file which is located in users document folder (e.g. C:\Users\CurrentUser\Documents\NugetDeploy).

The file will be created when using the tool for the first time or accessing its configurations and there is no settings file yet. So its not a problem if the settings file is depleted, although you will lose you settings (e.g. the NuGet servers you have set up).

As of version 0.12.0.0, old settings files will be converted, in case the format has changed in the new version. So your configuration is not lost.

4.1 Logging

The tool will also create log files in order to keep track of problems that might occur. Since this is done using log4net, you will be able to configure the logging mechanism by providing a file named **log4net.config** which needs to be located inside the users document folder on your system (so it will be next to the **settings.xml** file).

If no such file exists, then the tool will use a basic log configuration which logs all entries inside a logfile.log file, which again is located inside the users document folder on your system.

4.2 Configuration categories

The configurations are divided into sub categories for better overview. So if you open the configurations in visual studio (Tools → Options... → NuGet Deploy), you will find those sub categories.

The sub categories are explained down below.

4.3 General - MsBuild Paths

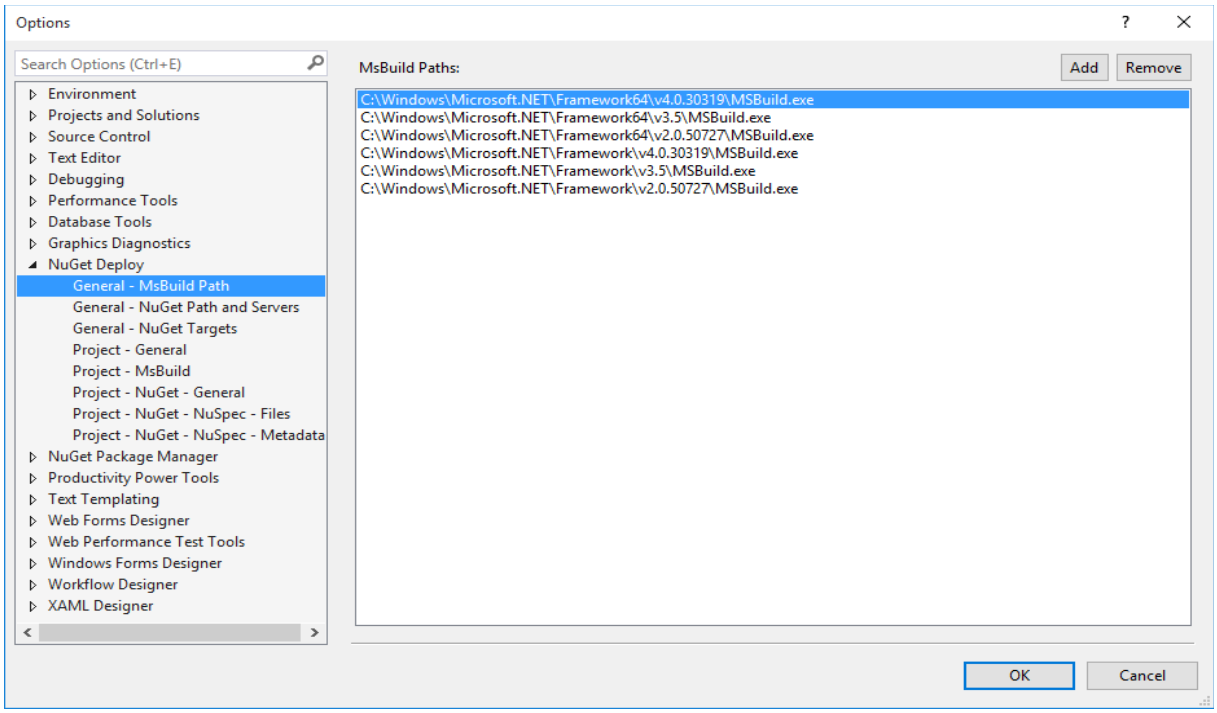

Illustration 4.1: General - MsBuild Paths page

Pretty much self-explanatory, in this dialogue you can add paths to any msbuild.exe you want. If there is no settings file yet or the settings file does not contain any msbuild path, then the tool will automatically look for any know msbuild.exe on your system.

Since the msbuild.exe is part of the .net framework, they are most likely located at the following destination:

- **<Systemroot>\Microsoft.NET\Framework<64>\<.net Version>\msbuild.exe**

Furthermore if you have set up the msbuild.exe to be included in the path environment variable, then this will also be checked.

4.4 General - NuGetPath and Servers

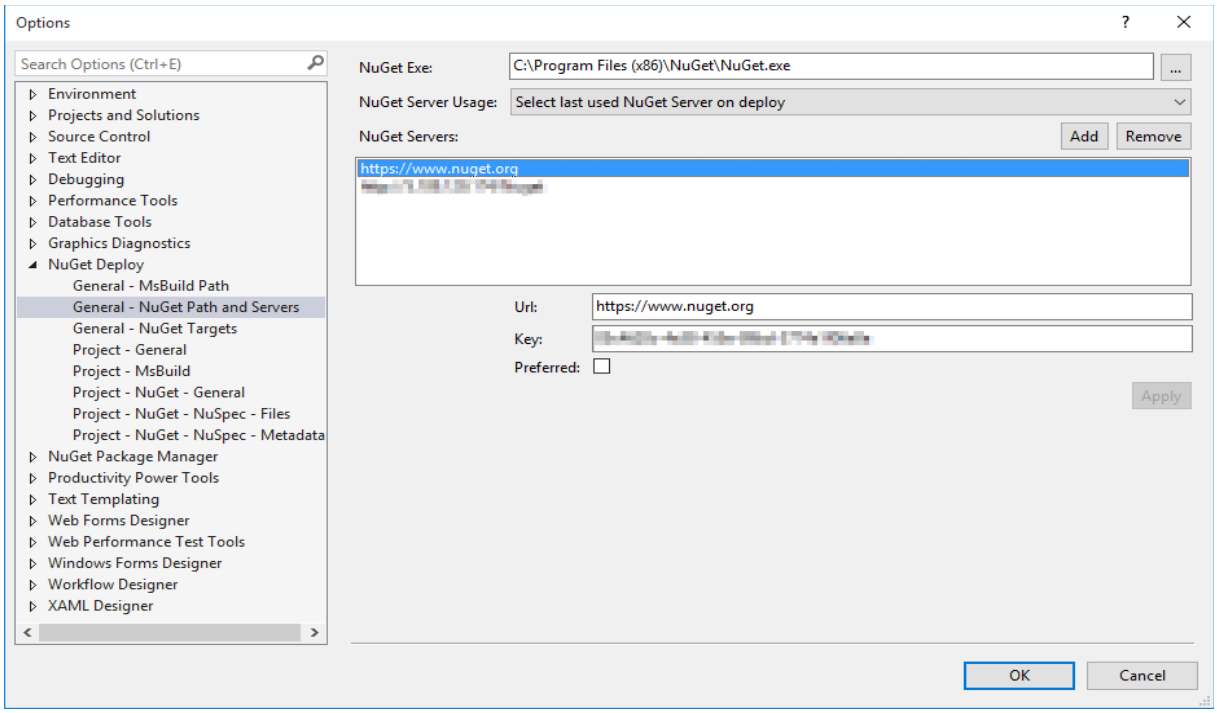

Illustration 4.2: General - NuGet Path and Servers page

Here you can set up the path where the nuget.exe is located. If the nuget.exe is already included in the path environment variable, then the tool will be able to find the path on its own. So its advised to do so.

It also allows to set the server which will be automatically selected when the deploy dialog is shown.

There are three option available:

- ***none*** → meaning the first server in the list is used
- ***last used*** → meaning the last server that was used is selected
- ***preferred*** → meaning the preferred server is used if there is one, otherwise the first server is still used

Additionally you can add the all NuGet servers on which you want to deploy your packages.

For each NuGet server the following option can be configured:

- ***Url*** → determines the Url of the server
- ***Key*** → determines the key of the server, the key is encrypted when saved but for readability reasons is being displayed unencrypted
- ***Preferred*** → determines if this server is the preferred server to deploy to. So if the NuGet Server Usage has been set to preferred, then this server is always selected when the deploy dialog is shown. Note that there can only be on preferred server, so if you set this option the old preferred server is no longer the preferred one.

4.5 General - NuGet Targets

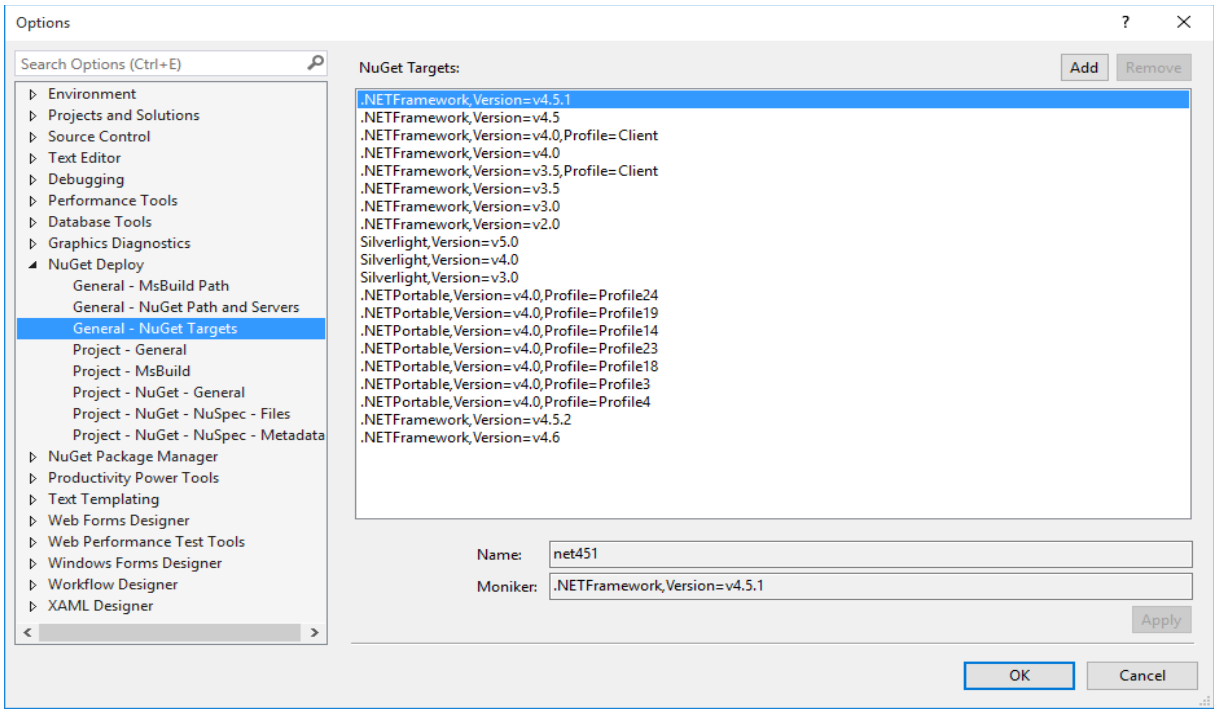

Illustration 4.3: General - NuGet Targets page

Here all the NuGet Targets are displayed and how they relate to the visual studio moniker. A moniker is basically the projects target framework (like .NET, Silverlight etc), the target framework version (like v4.5, v4.0 etc) and the traget framework profile if any (like Client, Full etc.)
currently the following targets are kown by default and can not be changed:

- **net451 -> .NETFramework,Version=v4.5.1 (as of Version 0.10.0.0)**
- **net45 -> .NETFramework,Version=v4.5**
- **net40-client -> .NETFramework, Version=v4.0, Profile=Client**
- **net40 -> .NETFramework, Version=v4.0**
- **net35-client -> .NETFramework, Version=v3.5, Profile=Client**
- **net35 -> .NETFramework, Version=v3.5**
- **net30 -> .NETFramework, Version=v3.0**
- **net20 -> .NETFramework, Version=v2.0**
- **sl5 -> Silverlight, Version=v5.0**
- **sl4 -> Silverlight, Version=v4.0**
- **sl3 -> Silverlight, Version=v3.0**
- **portable-net45+sl5 -> .NETPortable, Version=v4.0, Profile=Profile24**
- **portable-net403+sl5 -> .NETPortable, Version=v4.0, Profile=Profile19**
- **portable-net40+sl5 -> .NETPortable, Version=v4.0, Profile=Profile14**
- **portable-net45+sl4 -> .NETPortable, Version=v4.0, Profile=Profile23**
- **portable-net403+sl4 -> .NETPortable, Version=v4.0, Profile=Profile18**
- **portable-net40+sl4 -> .NETPortable, Version=v4.0, Profile=Profile3**

4.6 Project – General

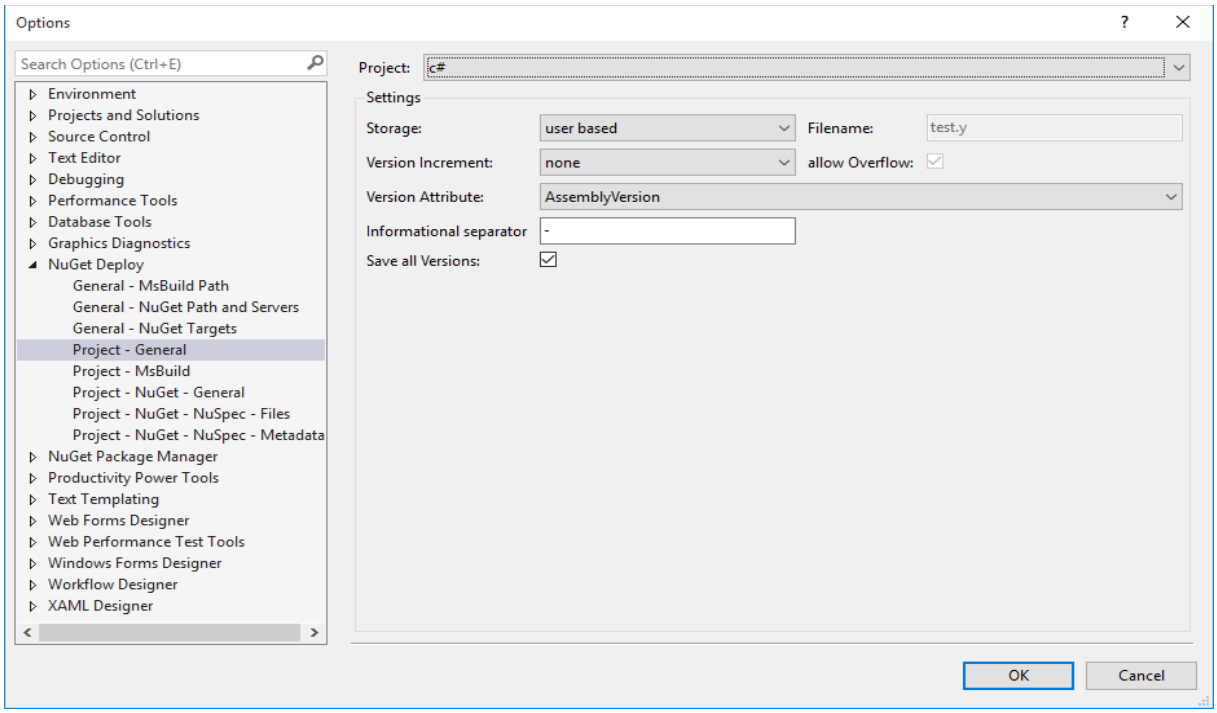

Illustration 4.4: Project – General Page

In this page you can change general options, which refer to a certain type of project.

- ***Storage*** → this options lets you change the way the configuration file is stored. By default the configuration file is saved as explained in the point Configuration which is referred to as ***user based***. However it is also possible to save the configuration file next to the project and have the project include the configuration file, which is referred to as ***project based***.
- **Filename** → lets you determine the filename of the configuration file when the storage is set to ***project based***. As such the option is only available if the storage is set to ***project based***. Note:
  - if the file name is empty, the default filename will be used, which is ***nugetdeploy.config***
  - If the filename does not provide an extension, then ***.config*** will be added by default
- ***Increment*** → allows you to auto increment the version each time a deployment is done. So if this option is set to anything other then ***None,*** it will increase the defined position in the version (Major, Minor, Revision or Build). If this value activated, then the save back option of the metadata version (see Project – NuGet – NuSpec – Metadata) should also be activated, so that the new version will be saved back to the project.
- ***allow Overflow*** → allows to increment the next position of a version, if the current position is at its maximum. (E.g. if the current Version would be 1.65535.0.0 and Minor should be increased, then the new version will be 2.0.0.0). Keep in mind that, should an overflow occur and this option is not set, then an error message will be shown up on deployment.
- ***Version Attribute*** → determines from which assembly version the version of the library is determined
- ***Informational separator*** → determines the informational separator in the version. This value can not be null and will be **–** by default
- ***Save all Version*** → determines that the version will be saved back in all assembly version attributes. E.g. if there is a AssemblyFileVersion and AssemblyVersion in the AssemblyInfo, then the version will be saved in both attributes if the option is enabled.

4.7 Project - MsBuild

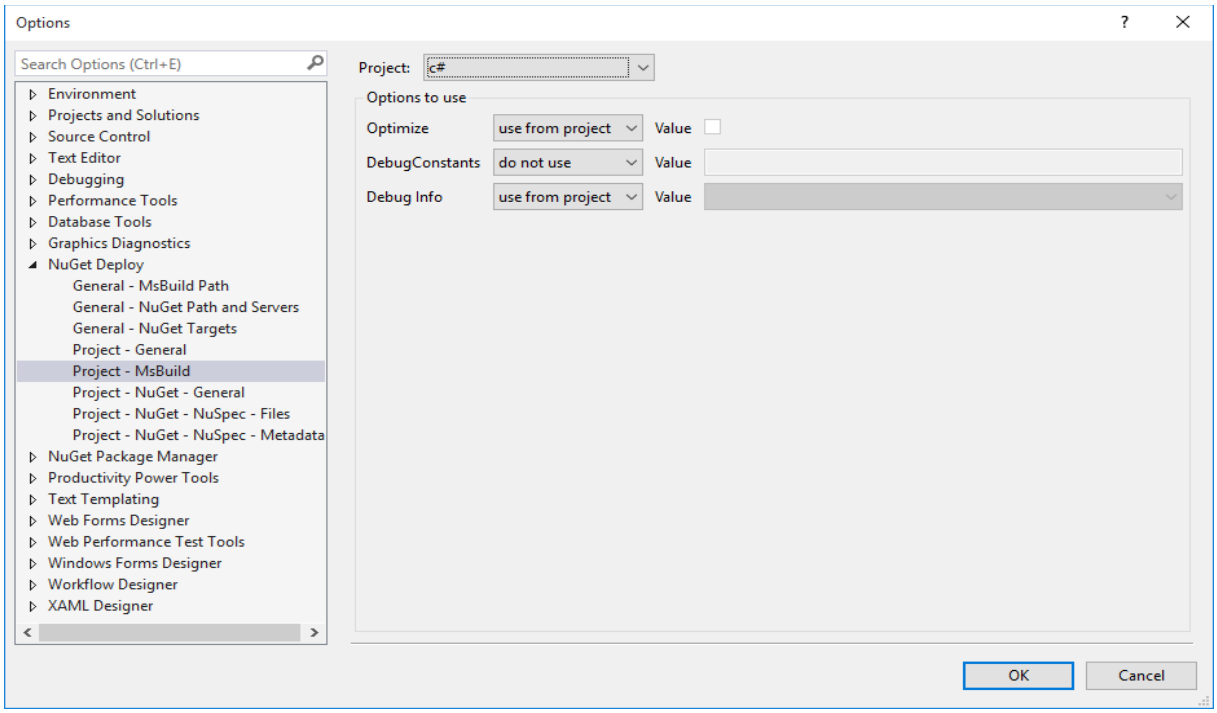

Illustration 4.5: Project – MsBuild page

Here you can set what build options will be used. Basically each of the options can be set the the following uses:

- ***do not use*** = which means this option is not used in the build process
- ***use*** = which mean this option will be used in the build process, this also enabled you the set the value that will be used (e.g. for Debug info you can set if none, pdbonly or full will be used)
- ***use from project*** = which means this option is used from the projects build configuration. this does not work for c++ since the build properties are not provided in c++ projects. So you should only set them to use and provide the values for c++ projects.

Also if the debug info has been set to **use + pdbonly,full** or to **use from project** (and the projects provides either **pdbonly** or **full**), then the projects .pdb file will also be added. This however only works if the NuSpec File Option is disabled because if the option is enabled the files for the package will be taken from the nuspec file of the project.

4.8 Project – NuGet – General

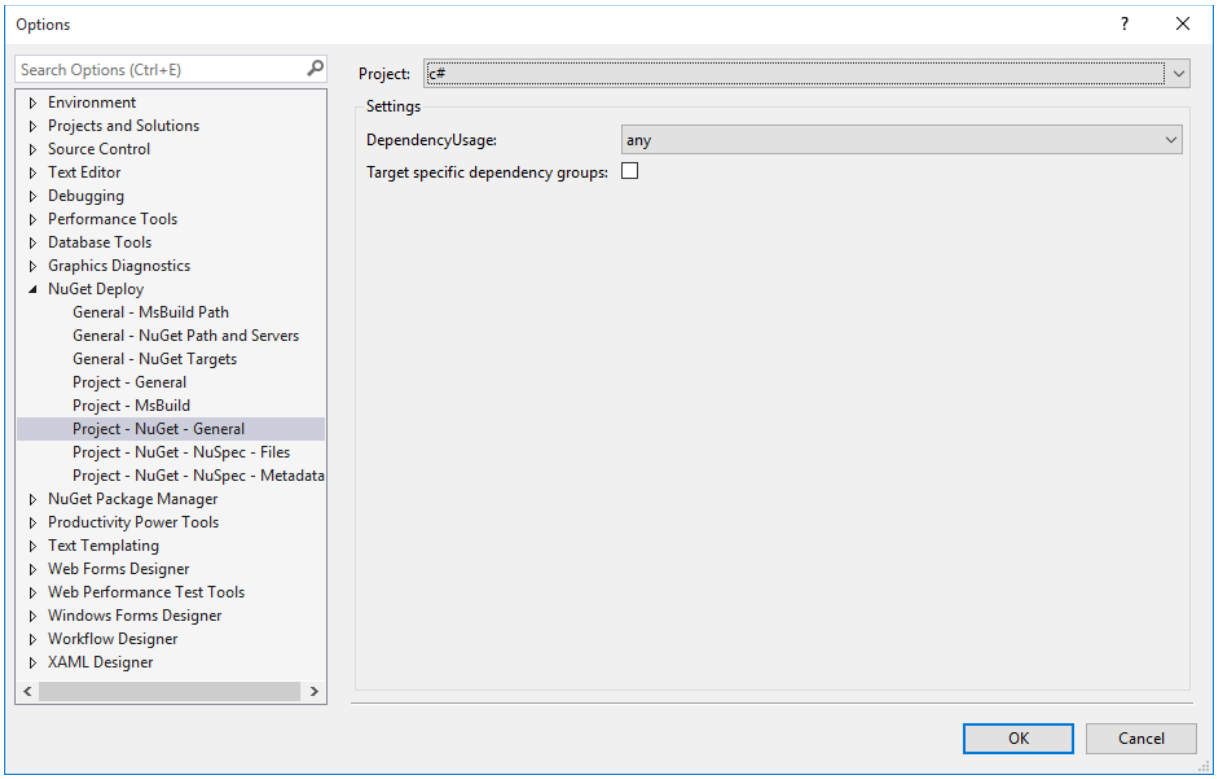

Illustration 4.6: Project - NuGet - General

In this page you can specify genreal project related nuget options.

The following options are available:

- ***Dependency Usage*** → determines how other nuget dependencies are used
  - ***do not use*** → dependencies are not used at all
  - ***any*** → any other nuget dependency is used
  - ***non development only*** → meaning that development dependencies are not used, only those that are not non development dependencies
- ***Target specific dependency groups*** → determines if dependencies will be group in their target specific group or not. If disabled, all dependencies will be placed in the fallback group

4.9 Project – NuGet – NuSpec - Files

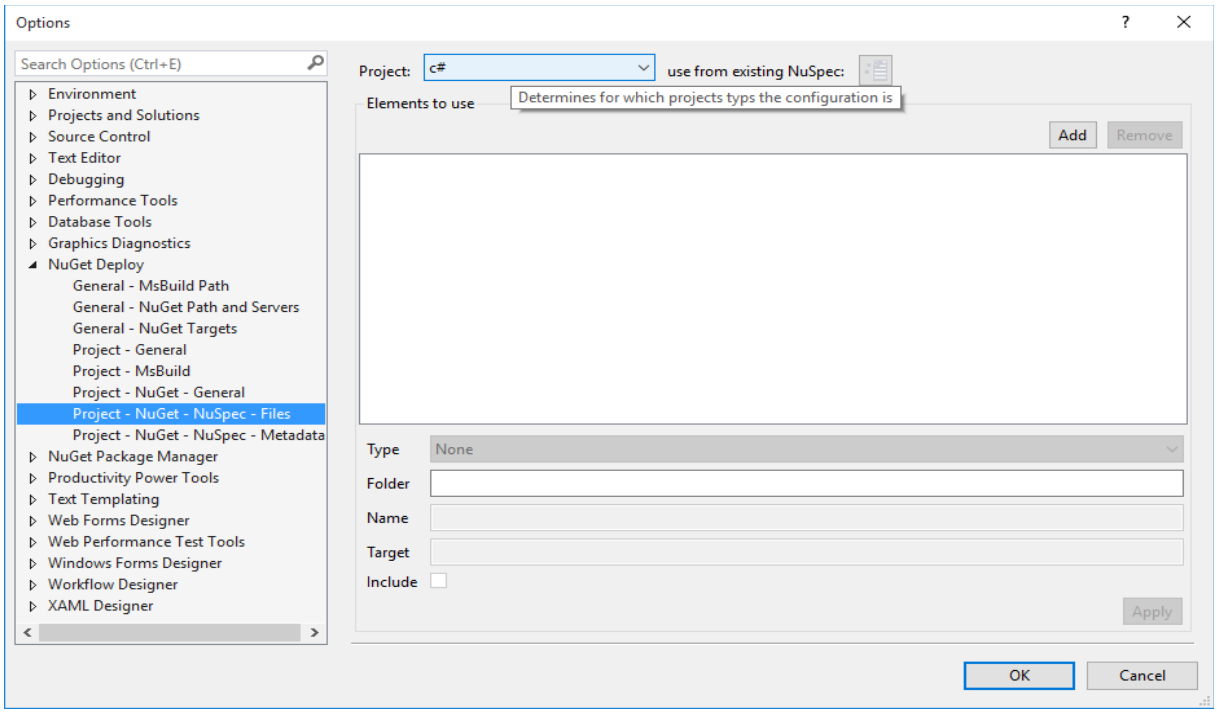

Illustration 4.7: Project – NuGet - NuSpec - Files

This option allows you to set up which files from the certain type of project will be included in the package and under which target they will appear. Currently this option is only available for C# and Visual Basic projects.

If the ***use from existing NuSpec*** option is checked, then the ***Elements to use*** will be blocked. Because in this case files are used from the existing .NuSpec file. However if the NuSpec file does not contain any file, then the files of the project are still used this once.

Since this is a rather complex option, the elements explained more in detail below

4.9.1 Type

Determines the Build Action the item must have in order to be used, these are the same as the ones available in visual studio.

For example if you have setup content, complie, then only items which this build action will be used.

4.9.2 Folder

Determines in which folder the item must appear. If it is empty, then any item matching the configured name will be used. This option also supports wildcard.

Some examples:

- ***A/**** = means all items that are in a subfolder of a folder called ***A*** will be used. E.g. ***A/B/b.txt*** and ***A/C/c.txt***, both would be added.
- ***A*B*** = means all items that are in a folder that starts with ***A*** and ends with ***B***, will be added. E.g. ***AaB/a.txt*** and ***AbB/a.txt***, both would be added.
- ***A/*/B*** = means all items that are in a subfolder ***B***, which is in any subfolder of ***A*** will be used. E.g. ***A/C/B/a.txt*** and ***A/D/B/a.txt***, both would be used, but ***A/C/D.a.txt*** would not be used.

4.9.3 Name

Determines which name the item must have. If it is empty, then any item matching the configured folder will be used. This option also supports wildcards.

Some examples:

- ***.txt*** = means all items that end with ***.txt*** will be used. E.g. a.txt or b.txt, both would be used.
- ****test*.txt*** = means all items that have ***test*** in their name will be used. E.g. ***atest.txt*** and ***btesta.txt***, both would be used.

4.9.4 Target

Determines under which target the file will appear in the package.

4.9.5 Include

Determines that items which match the configured folder and name will be used, so this option has to be checked in order to use the configured rule.

4.9.6 Folder + Name

The folder and name options are used together, so basically they are combined via a ***/***. E.g. ***A/*B*** and ****test*.txt*** would combine to ***A/*/B/*test*.txt***, which would mean all items that are in a subfolder ***B***, which is in any subfolder of ***A*** and have ***test*** in their name will be used.

4.9.7 Overlapping Rules

If two or more rules overlap for certain items (so the item match more then one rule), then the first rule will be used (the one that is the top most of the overlapping rules). This is only problematic if those rules have different targets though.

4.10 Project – NuGet – NuSpec – Metadata

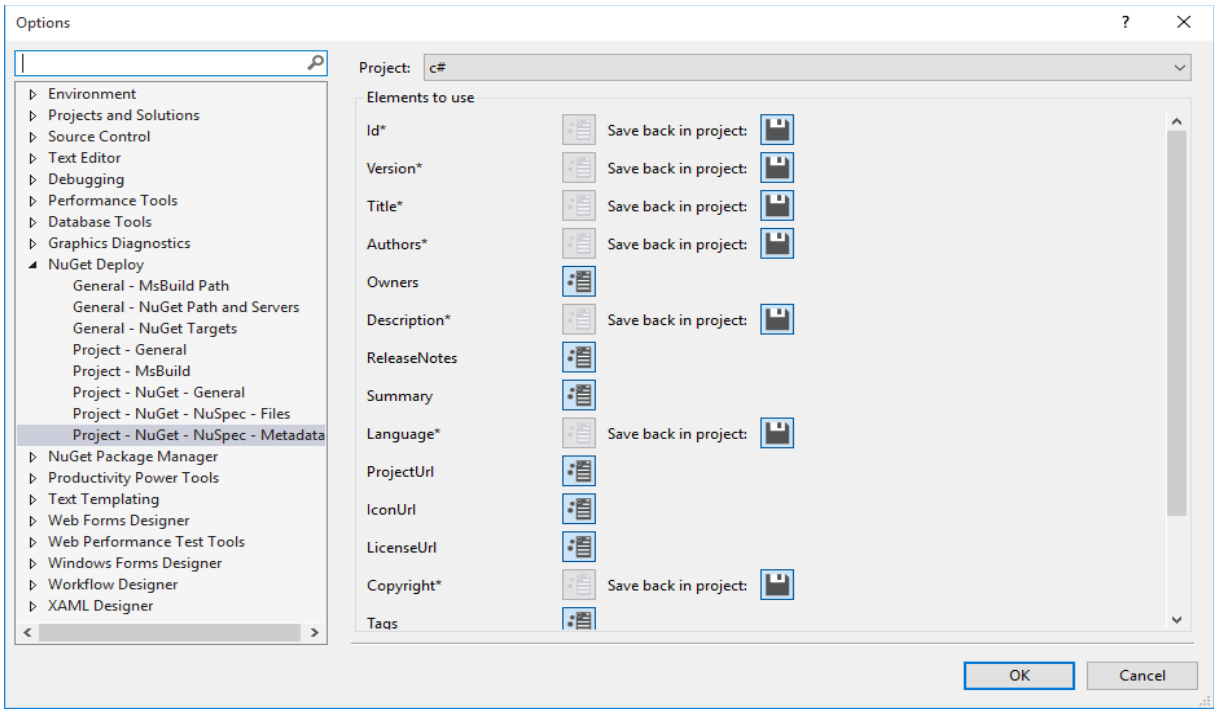

Illustration 4.8: Project – NuGet – NuSpec – Metadata

This option enabled you to automatically add the created .NuSpec file to your project or use an existing one in the project. If you already have a .NuSpec file in you project file, then the tool will be able to find it, even if its in a sub folder and update/reuse it. In order to do so, one of the NuSpec option must be checked, otherwise a .NuSpec file will not be added to your project.

All Elements which are checked will be saved in the NuSpec file and later be used when you deploy the library again. Elements marked with a * can be created from the project. So they do not actually need to be retrieved from the NuSpec file (e.g. version, which you will most likely always increase in the assembly info)
Most of the elements will be created using the information from the assembly info of your project. The following list shows you which NuSpec element will be used and which properties from the project they use:

- ***id*** = will be created from the projects assemblyName (prior to version 0.10.0.0 the default/rootnamespace was used)
- ***version*** = will be created from the projects version, located in the assembly info
- ***title*** = will be created from the projects title, located in the assembly info
- ***authors*** = since there is actually no such element in the assembly info, the company will be used instead. Since this element is mandatory but will not necessarily be set, the value *unkown author* will be used in such cases.
- ***description*** = will be created from the projects description, located in the assemblyinfo. Since this element is mandatory but will not necessarily be set, the value *unkown description* will be used in such cases.
- ***language*** = will be created from the culture, located in the assembly info.
- ***copyright*** = will be created from the copyright, located in the assembly info.
- ***dependencies*** = will be created from an existing package.config file, this way all NuGet packages which are already added will also be known and be added when you install this package

Additionally alls those element except ***dependencies*** can also be saved back to the project. This means that, should the value have been changed during deployment (see Nuget Deploy...) it will be saved back to the project, thus changing it.

5 Known Issues

This is the list of known issues I currently know of:

1. Silverlight libraries need to be build using the x86 Version of the msbuild.exe, otherwise the msbuild.exe will do nothing, not even return an error hint