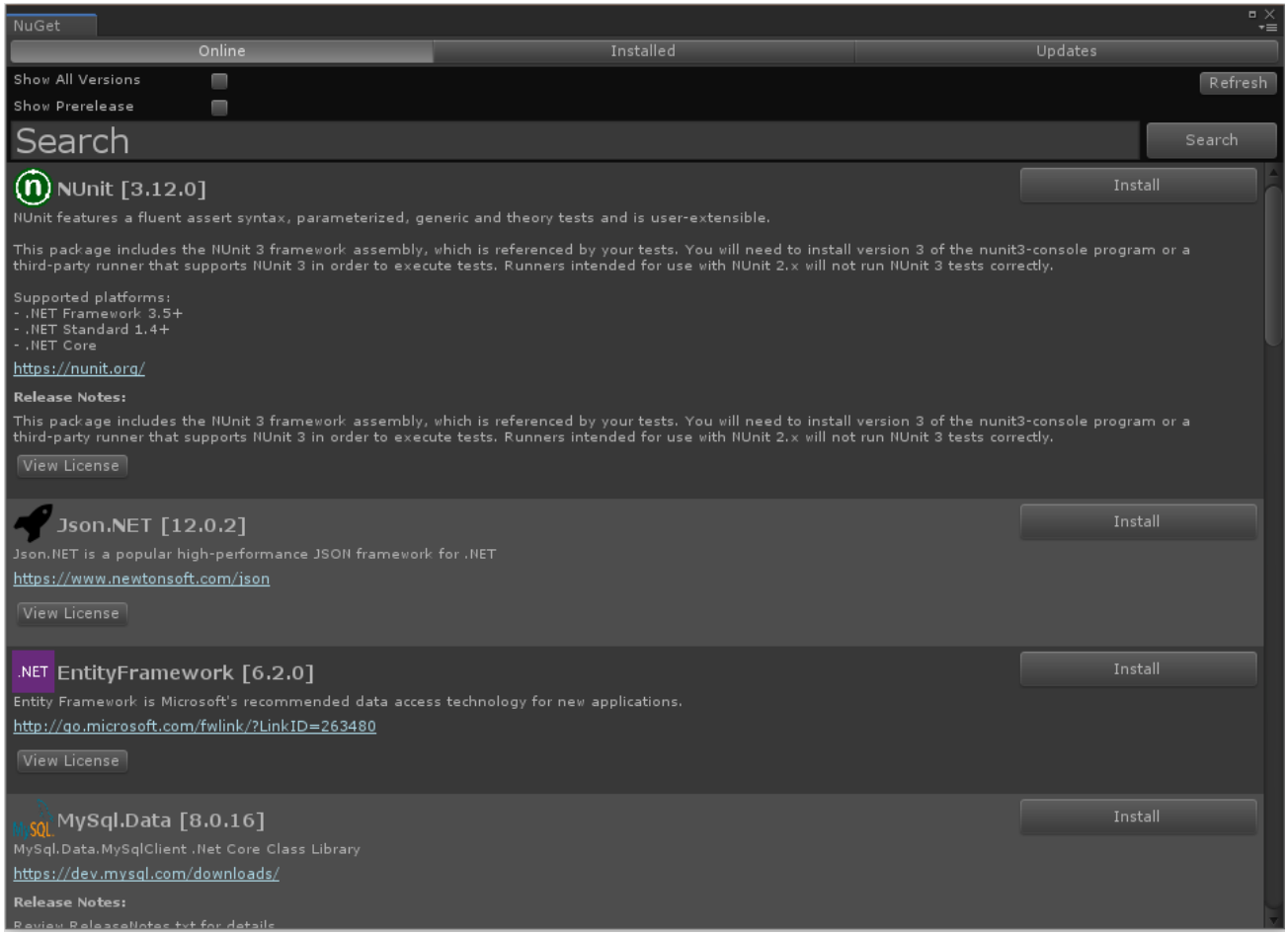
What is NuGetForUnity?

NuGetForUnity is a NuGet client built from scratch to run inside the Unity Editor. NuGet is a package management system which makes it easy to create packages that are distributed on a server and consumed by users. NuGet supports sematic versioning for packages as well as dependencies on other packages.

You can learn more about NuGet here: nuget.org

NuGetForUnity provides a visual editor window to see available packages on the server, see installed packages, and see available package updates. A visual interface is also provided to create and edit *.nuspec* files in order to define and publish your own NuGet packages from within Unity.

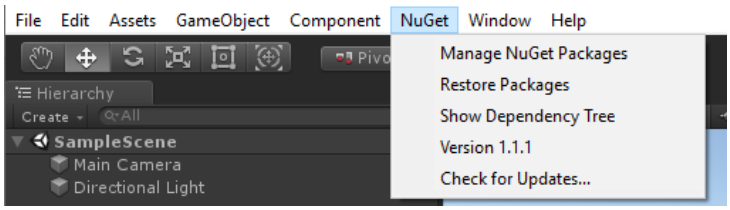


How do I install NuGetForUnity?

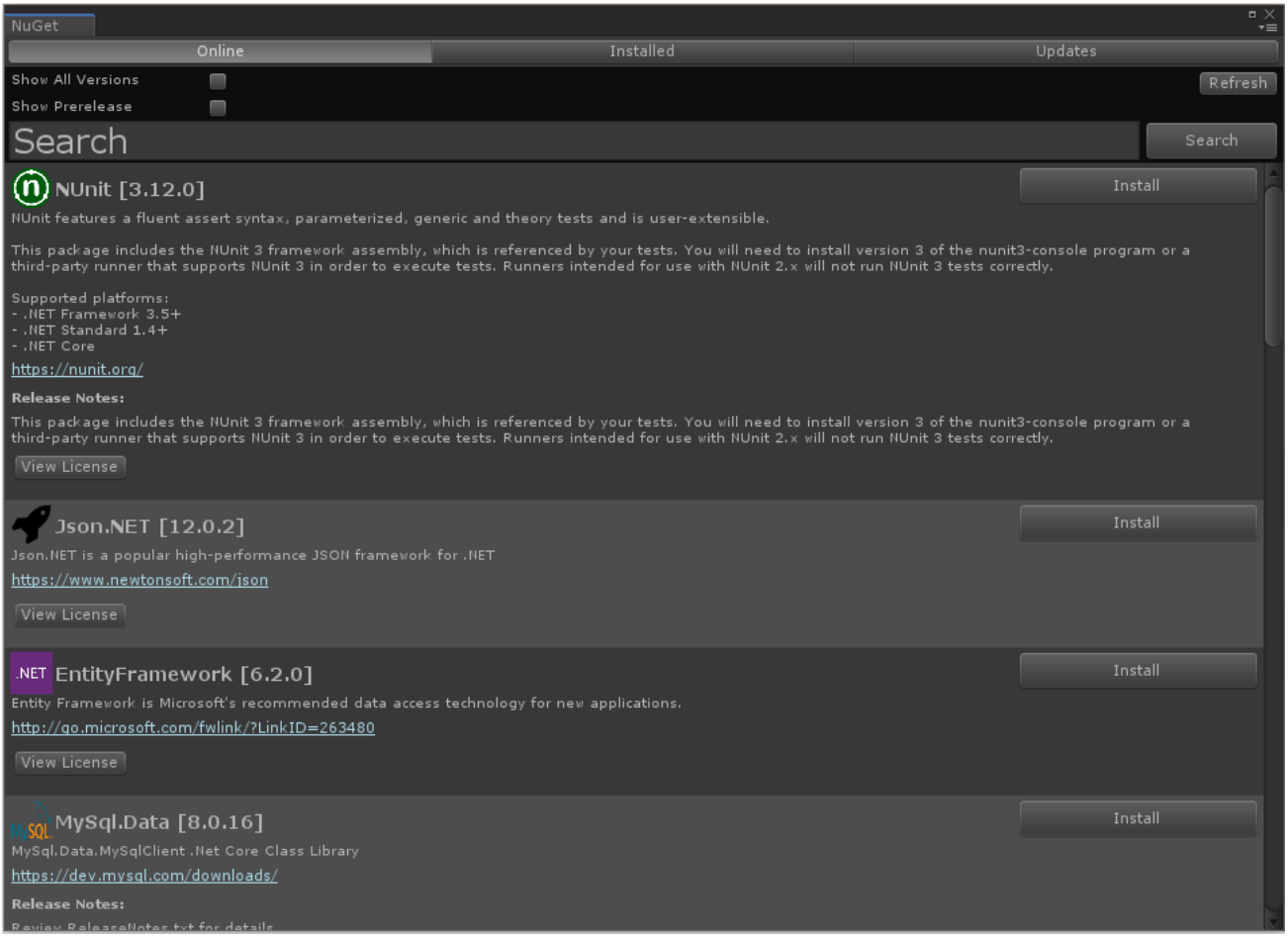
Install the latest Unity package into your Unity project. Located here.

How do I use NuGetForUnity?

To launch, select **NuGet → Manage NuGet Packages**



After several seconds (it can take some time to query the server for packages), you should see a window like this:



The **Online** tab shows the packages available on the NuGet server.

Enable **Show All Versions** to list all old versions of a package (doesn't work with nuget.org).
Disable **Show All Versions** to only show the latest version of a package.

Enable **Show Prelease** to list prerelease versions of packages (alpha, beta, release candidate, etc).
Disable **Show Prerelease** to only show stable releases.

Type a search term in the **Search** box to filter what is displayed.

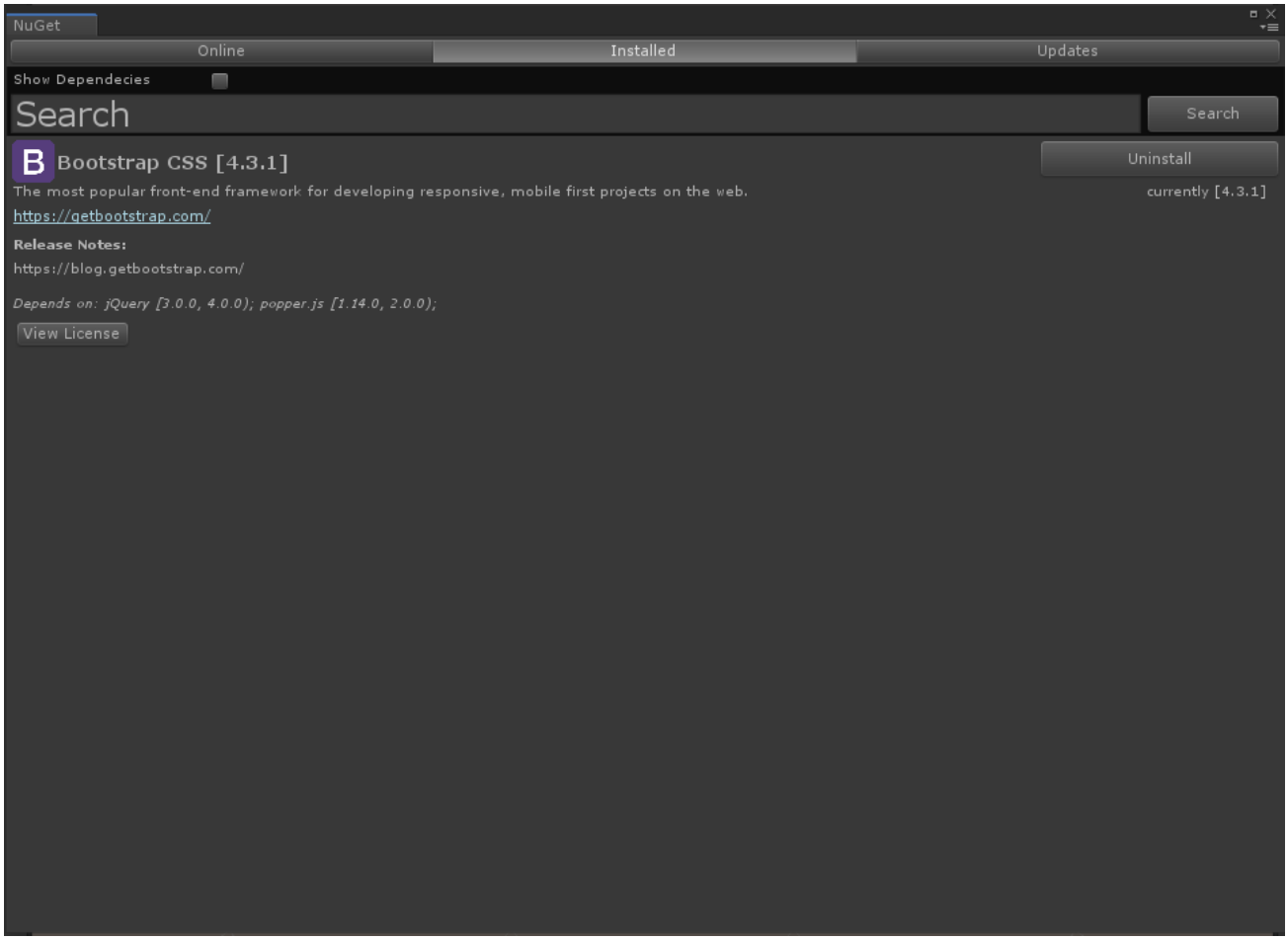
Press the **Refresh** button to refresh the window with the latest query settings. (Useful after pushing a new package to the server and wanting to see it without closing and reopening the window.)

The name of the package, the version of the package (in square brakets), and a description are displayed.

Click the **View License** to open the license in a web browser.

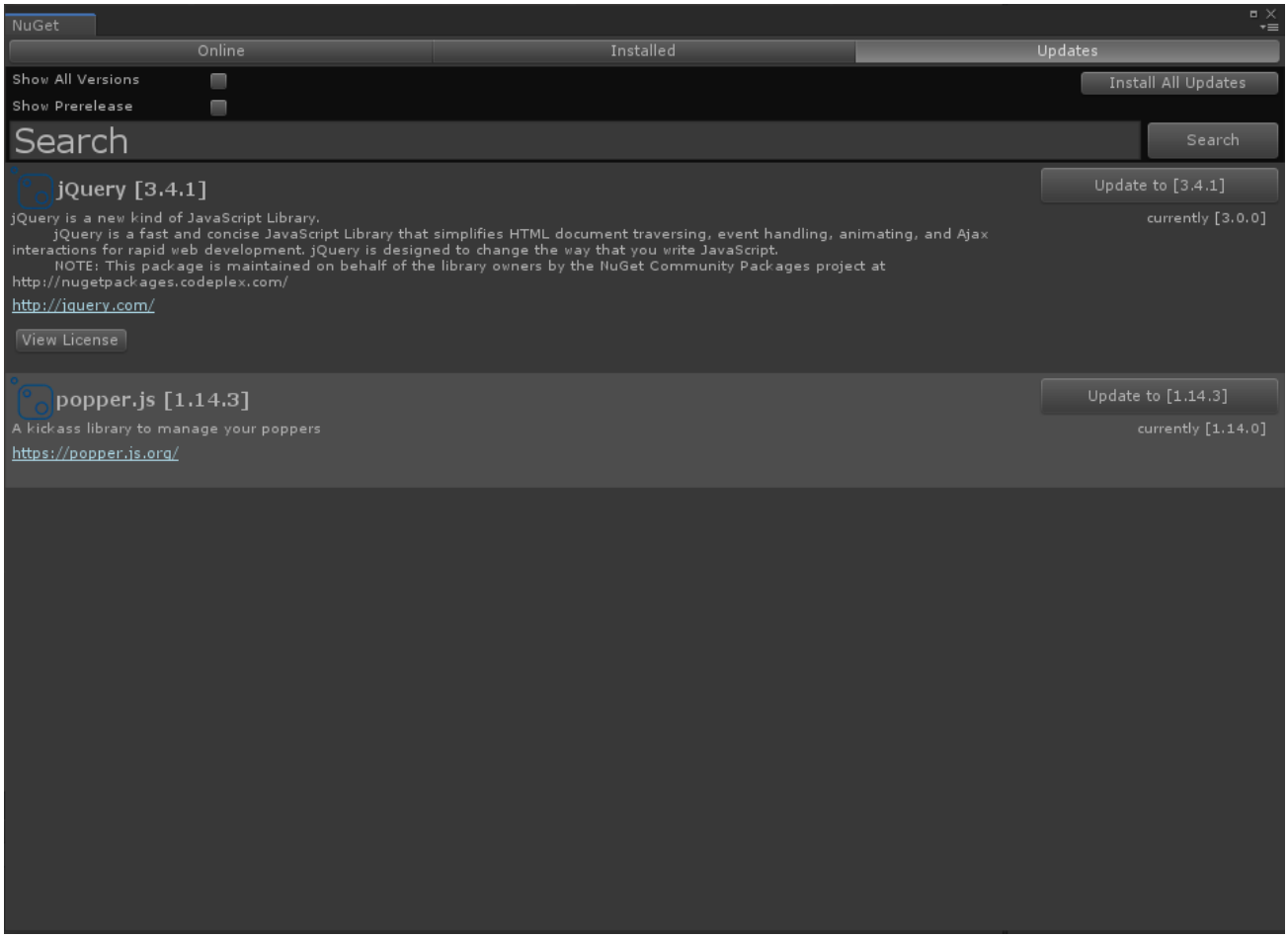
Click the **Install** to install the package. Note: If the package is already installed an **Uninstall** button will be displayed which lets you uninstall the package.

The **Installed** tabs shows the packages already installed in the current Unity project. By default it only displayed the packages that were manually installed and not the packages that were brough it as dependencies. If you want to see all the packages check the **Show Dependencies** checkbox.



Click the **Uninstall** button to uninstall the package.

The **Updates** tab shows the packages currently installed that have updates available on the server.



The version in brackets on the left is the new version number. The version in brackets in the **Update** button is the currently installed version.

Click the **Update** button to uninstall the current package and install the new package.

How does NuGetForUnity work?

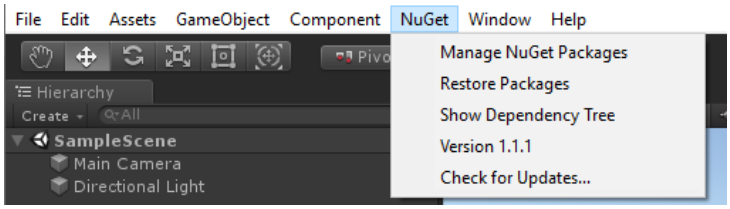
NuGetForUnity loads the *NuGet.config* file in the Unity project (automatically created if there isn't already one) in order to determine the server it should pull packages down from and push packages up to. By default, this server is set to the nuget.org package source.

The default NuGet.config file:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <packageSources>
    <add key="NuGet" value="http://www.nuget.org/api/v2/" />
  </packageSources>
  <activePackageSource>
    <add key="NuGet" value="http://www.nuget.org/api/v2/" />
  </activePackageSource>
  <config>
    <add key="repositoryPath" value="./Packages" />
    <add key="DefaultPushSource" value="http://www.nuget.org/api/v2/" />
  </config>
</configuration>
```

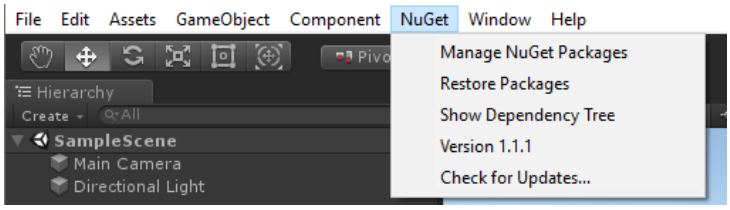
You can change this to any other NuGet server (such as NuGet.Server or ProGet - see below). The **NuGet → Reload NuGet.config** menu item is useful if you are editing the *NuGet.config* file.

See more information about *NuGet.config* files here: https://docs.nuget.org/consume/nuget-config-settings



NuGetForUnity installs packages into the local repository path defined in the *NuGet.config* file (`repositoryPath`). By default, this is set to the `Assets/Packages` folder. In the *NuGet.config* file, this can either be a full path, or it can be a relative path based on the project's Assets folder. Note: You'll probably want your Packages folder to be ignored by your version control software to prevent NuGet packages from being versioned in your repository.

When a package is installed, the *packages.config* file in the project is automatically updated with the specific package information, as well as all of the dependencies that are also installed. This allows for the packages to be restored from scratch at any point. The `Restore` operation is automatically run every time the project is opened or the code is recompiled in the project. It can be run manually by selecting the **NuGet → Restore Packages** menu item.



Note: Depending on the size and number of packages you need to isntalled, the `Restore` operation could take a *long* time, so please be patient. If it appears the Unity isn't launching or responding, wait a few more minutes before attempting to kill the process.

If you are interested in the process NuGetForUnity follows or you are trying to debug an issue, you can force NuGetForUnity to use verbose logging to output an increased amount of data to the Unity console. Add the line `<add key="verbose" value="true" />` to the `<config>` element in the *NuGet.config* file. You can disable verbose logging by either setting the value to false or completely deleting the line.

The *.nupkg* files downloaded from the NuGet server are cached locally in the current user's Application Data folder. (`C:\Users\[username]\AppData\Local\NuGet\Cache`). Packages previously installed are installed via the cache folder instead of downloading it from the server again.

Package default initialization

If package has an `Init.template` file in its root it will be used to inject default package initialization code into the project. The file should look like this:

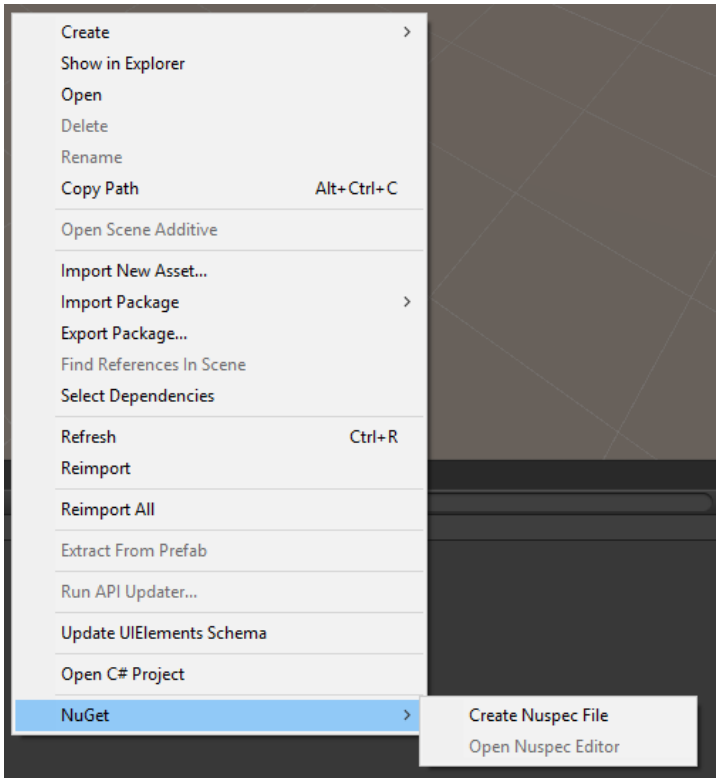
```
InitDependencies: a, b, c <in case the init code depends on packages the package itself doesn't depend on>
Uses: use1, use2 <using clauses that need to exist at the top of the file>
CustomExceptionLogging: <code to insert into CustomExceptionLogging if package wants to handle init exceptions>
{
        ...
}
InitCode:|SceneInitCode: <this is the only required section>
{
        ...
}
```

Nuget for Unity will look for `AppInitializer.cs` and `AppInitializer.Generated.cs` under `Assets/Scripts/Initialization/` . If they don't exist they will be created with the default content. The Generated file contains Unity initialization logic and calls to specific packages initialization code. The calls are automatically injected here and users should not modify this file. The actual initialization methods are injected in `AppInitializer.cs` and that is where users can modify how initialization is done and supply the required parameters.

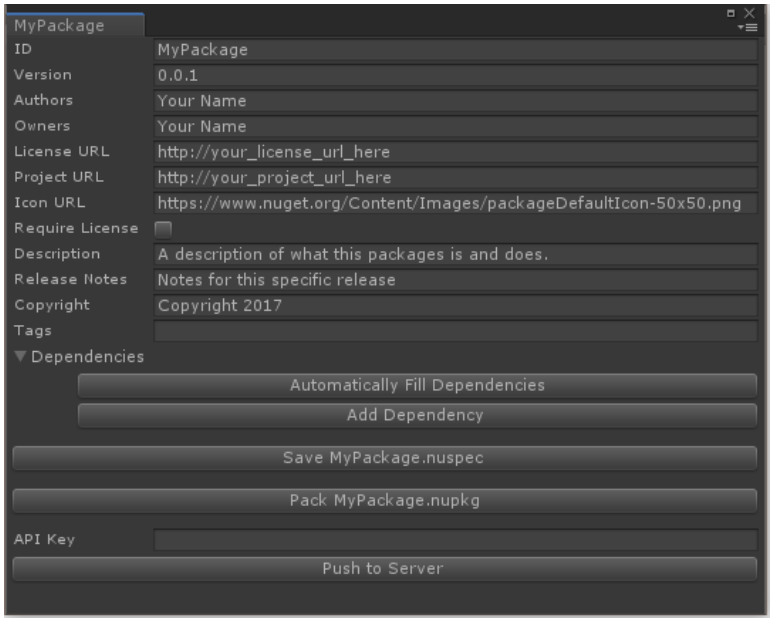
There are two ways to specify init code in `Init.template` file: `InitCode` and `SceneInitCode` . Most of the packages should use `InitCode` . `SceneInitCode` should be used only if your package needs to create Unity GameObjects on the scene in its initialization. Note that in that case it will also be initialized a bit later than packages that use `InitCode` .

How do I create my own NuGet packages from within Unity?

First, you'll need to create a *.nuspec* file that defines your package. In your Project window, right click where you want the *.nuspec* file to go and select **NuGet → Create Nuspec File**.



Select the new *.nuspec* file and you should see something like this:



Input the appropriate information for your package (ID, Version, Author, Description, etc). Be sure to include whatever dependencies are required by your package.

Press the **Pack** button to pack your package into a *.nupkg* file that is saved in the `C:\Users\[username]\AppData\Local\NuGet\Cache` folder.

Press the **Push** button to push your package up to the server. Be sure to set the correct API Key that give you permission to push to the server (if you server is configured to use one).

How do I create my own NuGet server to host NuGet packages?

You can use NuGet.Server, NuGet Gallery, ProGet, etc to create your own NuGet server.

Alternatively, you can use a "local feed" which is just a folder on your hard-drive or a network share.

Be sure to set the proper URL/path in the *NuGet.config* file and you should be good to go!

Read more information here: http://docs.nuget.org/create/hosting-your-own-nuget-feeds

How is this fork different from the original

- It is setup to use Unity 2018.3.11f1 for creating unitypackage.
- The CreateDLL solution is updated to .NETFramework 4.6.1
- DotNetZip.dll is deleted and replaced with `System.IO.Compression`
- Loading packages' icons is done asynchronously so that package window opens a bit faster.
- When uninstalling the package it will also delete all its dependencies that are not manually installed and that no other package depends on.
- It only shows the manually installed packages by default in installed tab but there is an option to show them all.
- It supports packages with Init.template which they can use to specify default initialization code that should be injected in the project on installation.

#How to build There is a provided build.ps1 powershell script you should run to rebuild the unitypackage. For it to work you need to make sure you have `UnitySetup` and `VSSetup` powershell modules installed. You can check what modules you have by running `Get-Module -ListAvailable`. Custom modules are listed at the top. If you don't have this modules installed you need to run these commands to install them:

```
Install-Module VSSetup -Scope CurrentUser -RequiredVersion 2.0.1.32208
Install-Module UnitySetup -Scope CurrentUser
```

Note that you need the exact version of VSSetup module so if you have some other version installed you should probably uninstall it first using `Uninstall-Module VSSetup`.

In case UnitySetup can't find your installation of Unity 2018.3.11f1 you may need to modify the installed module. The `Get-Module -ListAvailable` command will also write in which folder is it installed. There you should edit `function Get-UnitySetupInstance` in UnitySetup.psm1 and change the values of `$BasePath` array.

In case you only have a newer version of Unity and want to upgrade this project to use it you just need to manually open the root folder and also the Packager folder in newer Unity so it updates the ProjectVersion file.