A BEGINNER'S GUIDE TO MODDING SUBNAUTICA AND SUBNAUTICA BELOW ZERO

mroshaw

2021

TABLE OF CONTENTS

INTRODUCTION

I love writing code. I've written code since I was a kid, and that was quite a long (long, long) time ago. I don't code in my job, so it's a challenge to find time to mess around with the new tools and languages that are so prevalent. In the past, I've used Visual Studio and C# to write some tools to help me with my day job, so I'm aware of the language and vaguely aware of how it all works. Like many, I'm "self-taught" and rely on forums, websites and communities to help me with my hobby.

I also love games. I've played games since I was a kid, and that… you get the idea. My first "gaming" machine was a ZX81, then a ZX Spectrum, moving onto a Commodore Amiga. I bought, and still own, an original PlayStation. Zip through (a lot of) time to now, and I play PS5 and Nintendo Switch with my kids. I've never been a PC gamer, and never really saw the attraction. Then I decided to build myself a "work computer". I set a budget of £500. I spent hours on pcpartpicker.com and ended up spending nearly £2K on a beautiful build with an RTX2070 GPU. So what could I do? Couldn't let that go to waste! I installed Steam, and here I am, playing Subnautica and Below Zero in glorious resolution and in VR.

It was during my foray into Subnautica VR that I came across a splendid bunch of people in a Discord server that was dedicated to modding Subnautica games. I put two and two together, dusted off a fresh copy of Visual Studio, and set to. Modding is fun! It's challenging, it's open, it's creative and it's supported by brilliant communities. And so, I've written this guide, hopefully to help others who want to get involved, have fun, and themselves contribute to the growing community.

I hope you like it!

CREDITS

Though I've written this guide, the hard work has been done by a community of amazing modders. I've named a few below, as help contributors to the scene, the tools and my efforts to build mods. It will be incomplete, but thank you to everyone for your help and support.

Thanks to, but not limited to:

- PrimeSonic
- Metious
- tobeyStraitjacket
- DaWrecka
- Mikjaw
- DarkOne
- MrPurple6411

And to everyone else on the Subnautica modding Discord.

GETTING STARTED

TOOLS YOU'LL NEED

You'll want to setup your development machine with the following tools:

- **Visual Studio** – this is where we'll write our mod code.
- **dnSpy** – this essential tool helps you to explore the code in the game and find useful methods and fields to target and modify.
- **QMod Manager** – this will help integrate your mod into the game, as well as provide useful functions to simplify your code.
- **SML Helper** – not mandatory, but this mod provides loads of useful functions so that you don't have to write them yourself.
- **BepInExPublicizer** – not mandatory, but this plugin "unlocks" protected methods and fields in game objects that makes modding much more powerful.
- **Vortex** – not mandatory, but useful for installing and updated base mods in your games.
- **GitHub Desktop** – not mandatory, but good to manage your source code in a Git repository, which will allow you to share and collaborate with others.
- **Subnautica / Below Zero** – you'll need to have games themselves installed, of course!

Here's a simple checklist of the tools, the versions used at the time of writing, and where to get them. You can check these off once you've set them all up, too.

Tool Name	Version Used	Where to download	Credit	Got it!	Installed!
Visual Studio	2019	Microsoft.com		☐	☐
dnSpy	6.1.8	Github.com		☐	☐
QModManager	4.1.4	Nexusmods.com	PrimeSonic	☐	☐
SML Helper	2.9.7	Nexusmods.com	PrimeSonic	☐	☐
QModManager (Below Zero)	4.1.4	Nexusmods.com	PrimeSonic	☐	☐
SML Helper (Below Zero)	2.9.7	Nexusmods.com	PrimeSonic	☐	☐
BepInEx-Publicizer	1.0.0	Github.com	MrPurple6411	☐	☐
Vortex	1.4.12	Nexusmods.com	DarkOne	☐	☐
GitHub Desktop	2.8.3	Github.com		☐	☐

Don't forget to endorse the mods that you download and use from Nexusmods.

Once you've got everything downloaded, let's get it all setup and ready to go.

> **NOTE:** Don't worry too much about version numbers. Generally, the latest is fine, though look out for .NET Framework and compatibility between mods.

SETTING UP

When I set up a Dev environment, I like to keep everything together. I have a dedicated 1TB SSD on my machine that I have mounted as D: and I put everything development related into D:\Dev. You can do whatever you want, go with defaults, or put stuff where it's accessible for you.

VISUAL STUDIO

Run the Visual Studio installer and pick the ".NET desktop development" Workload. You'll also want to ensure you pick the ".NET 4.7.2 SDK". That's it, really, the setup process is pretty straightforward these days, and VS installs a much smaller footprint than it used to.

DNSPY

Simple download dnSpy and unzip it to a folder location. Simple as that!

QMODMANAGER

Make sure you download and install the correct version from Nexusmods. Once you've done so, simply run the installer and let it do it's thing. You'll need to have the Subnautica game installed at this point, obviously, though I think it's same to assume this one since you're reading this guide!

VORTEX

This is entirely optional for mod development, but I recommend it anyway for playing with mods and supporting the Nexusmods community. Simply download this from Nexusmods and run the installer. Within Vortex, get yourself logged in and add Subnautica and / or Below Zero to your managed games. This will set everything up to download mods direct from the Nexusmods site.

SML HELPER

Optional but highly recommended! If you've installed Vortex, simply go to the Nexusmods page for SML Helper and click "MOD Manager Download". Within Vortex, find the mod and change the status to "Enabled". If you skipped Vortex, download the mod manually and unzip to the QMods folder in your game installation location.

BEPEXIN-PUBLICIZER

I really recommend downloading this tool and going through the process documented below. It will enhance your modding experience no-end and unlock a whole load of potential that you'd have to do without.

Unzip the file that you download into <game>\BepInEx\plugins\Bepinex-Publicizer. You should have folder that looks like this:
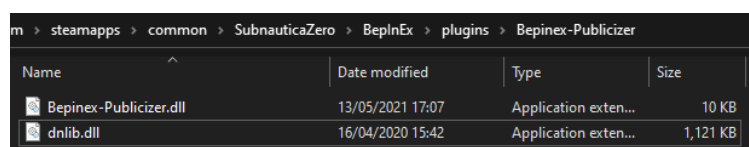
Run the game then exit. Return to Windows Explorer and go to this folder:

<game>\SubnauticaZero_Data\Managed\publicized_assemblies

You'll see some new files that we'll include in our project.

GitHub Desktop

Again, totally optional. In fact, Visual Studio will give you everything you need to manage your code. I just find the desktop tool gives me a warm fuzzy feeling that my code is safe, and nothing is ever lost! Install using the installer, following the prompts. Simples!

BEFORE YOU START

Coding and writing mods is fun! It's more fun with the support of a community of brilliant, helpful, fellow modders. This guide wouldn't have been possible without the help and support of this community. Supporting such a community is as much about giving and sharing as it is about receiving help and support.

Therefore, I would suggest you do two things before you begin:

- **Get yourself onto the Subnautica Modding Discord** - lurk, get involved, learn, ask questions, it's all good!
- **Set yourself up a repository on GitHub** – sharing and collaborating on code is one of the best ways to learn and improve. It's easy to do, just get yourself onto GitHub.com and register. We'll cover creating your first repository in the next section.

A couple more pointers before we start:

> **NOTE:** The examples below will use Subnautica: Below Zero. The principals, process and concepts are identical for the original Subnautica game, but do note that the tutorial is specifically focused on BZ.

> **NOTE:** Where I refer to <game>, this is the folder in which your Subnautica game is installed. For example, for me, it's:
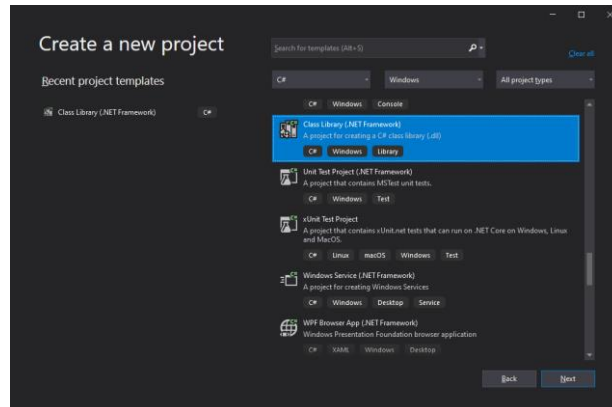> *C:\Games\Steam\steamapps\common\SubnauticaZero*

Right, let's make a mod!

YOUR FIRST SUBNAUTICA MOD

CREATING A VISUAL STUDIO C# PROJECT

Hop into Visual Studio and create a new project. You can use the "Class Library (.NET Framework)" template for this:

> **NOTE:** The "Class Library" template is not the one you want as it targets .NET Standard or .NET Core. You specifically want to select the template for .NET Framework

Let's call it "MyFirstSubnauicaMod_BZ", set a location and select ".NET Framework 4.7.2". Click the Create button, and you're done! Who said this was difficult, right?!
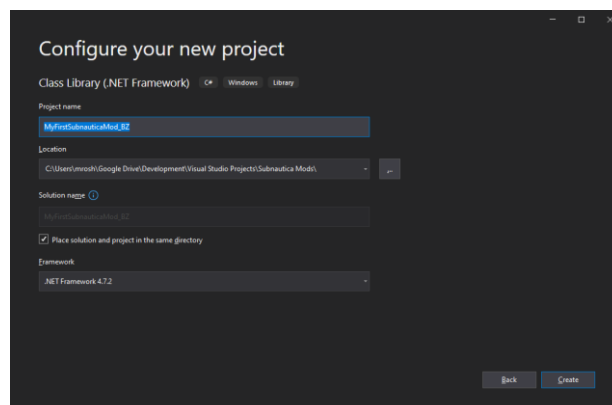

> **NOTE:** You must select ".NET Framework 4.7.2". If you don't see it, go back to the Visual Studio installer and select it. Alternatively, you can download it manually from Microsoft.com

Okay, there's more to it than that before we start.

In the Solution Explorer window, right click the "Solution" and change the name to something like "My Subnautica Mods". This will allow us to create multiple mod projects within the same solution, allow code reuse and a single Git repository keeping everything together.

You should have something like this:
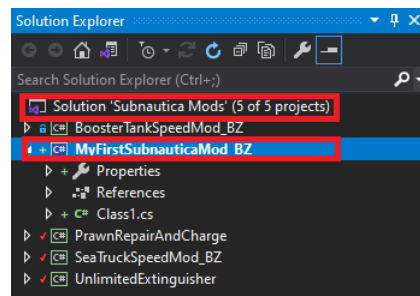
You can ignore the other projects that you can see there.

ADDING REFERENCES

Now we need to add references into our new project, so that we can reference our tools and Subnautica files.

Right click "References" and select "Add Reference". Click "Browse" and locate and add the following files:

Filename	Default / sample location	Got it!
UnityEngine.dll	<game>\SubnauticaBelowZero_Data\Managed	☐
UnityEngine.CoreModule.dll	<game>\SubnauticaBelowZero_Data\Managed	☐
UnityEngine.UI.dll	<game>\SubnauticaBelowZero_Data\Managed	☐
0Harmony.dll	<game>\ BepInEx\Core	☐
QModInstaller.dll	<game>\BepInEx\plugins\QModManager	☐
SMLHelper.dll	<game>\BepInEx\Core	☐

If you've run Publicizer, as recommended, you should add these references:

Filename	Default / sample location	Got it!
Assembly-CSharp_publicized.dll	<game>\SubnauticaBelowZero_Data\Managed\publicized_assemblies	☐
Assembly-CSharp-firstpass_publicized.dll	<game>\SubnauticaBelowZero_Data\Managed\publicized_assemblies	☐

If you've not run Publicizer, add these:
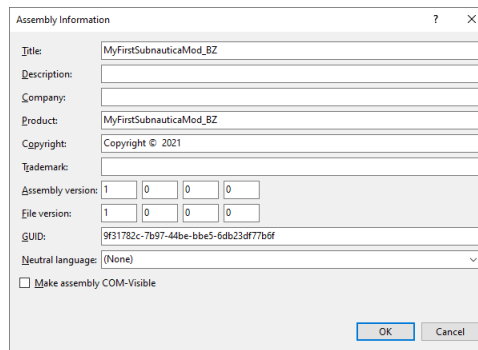
Filename	Default / sample location	Got it!
Assembly-CSharp.dll	game>\SubnauticaBelowZero_Data\Managed	☐
Assembly-CSharp-firstpass.dll	game>\SubnauticaBelowZero_Data\Managed	☐

CONFIGURING YOUR ASSEMBLY

You'll want to change some of the settings of your compiled DLL.

Right click the project in the Solution Explorer and select "Properties".

You'll see an Assembly Name and Default Namespace. If you've done a good job in picking a project name, you can leave these as is. Click the "Assembly" button and you'll see something like this:

Again, you can leave Title as it is, or change it, and you can set some of the data items here with your name and copyright info. Your initial version will be 1.0.0.0, so leave this as is too. Click OK to close that dialog.

Still in Properties, click the Build item. Here, we want to "Allow unsafe code". This prevents any issues when we try to use protected methods and fields that have been "unlocked" for us via the Publicizer. If you've chosen not to use the Publicizer, you can leave this as is.

One more thing we can do here is to setup our project to automatically deploy our mod after each successful build. We do this via the "Build Events" options. In Post-build event command line, you can add something like this:

```
mkdir "C:\Games\Steam\steamapps\common\SubnauticaZero\QMods\$(TargetName)"
copy /Y "$(TargetPath)"
"C:\Games\Steam\steamapps\common\SubnauticaZero\QMods\$(TargetName)"
copy /Y "$(ProjectDir)\mod.json"
"C:\Games\Steam\steamapps\common\SubnauticaZero\QMods\$(TargetName)\mod.json"
```

What this does is:

1. Creates a folder for our mod in the QMod folder, if one doesn't already exist
2. Copies the compiled mod DLL into the folder
3. Copies the mod.json file into the folder

Right, I think we're about ready to write some code!

CODING YOUR MOD

In order to activate your mod, QModManager needs to know a bit about it. This is where the "mod.json" file comes in. Every mod has to have a "mod.json" file, so right click your project, select "Add new item…" and in the name, use "mod.json". If Visual Studio has picked a template, don't worry, just select all and delete the content that it's added.

Now paste in the details about your mod:

```json
{
  "Id": "MyFirstSubnauticaMod_BZ",
  "DisplayName": "My First Subnautica Mod",
  "Author": "Oli Ollerenshaw",
  "Version": "1.0.0",
  "Enable": true,
  "AssemblyName": "MyFirstSubnauticaMod_BZ.dll",
  "VersionDependencies": {
    "SMLHelper": "2.9.6"
  },
  "Game": "BelowZero"
}
```

The important items here are the Assembly name, dependencies and game.

- **Assembly name** – must match the name of the DLL that you are building
- **Version Dependencies** – must call out other dependent mods and the version supported
- **Game** – can be either BelowZero or Subnautica

> **NOTE:** It's a good idea to keep on top of this, as well as the details of your Assembly, as mods like VersionChecker and QModManager itself, can make good use these values.

Okay, we're looking good. Now we can write some C#.

What I do, and this is completely down to personal preference, is to have a single class to handle registering the mod and managing options. I always call this class "QMod", but you can call it whatever you want. We'll stick with this for now.

> **NOTE:** When I refer to "your project", don't confuse that with "your solution"! Your solution is "Subnautica Mods", your project is "MyFirstSubnauticaMod_BZ.

By default, the template will have created a class for you. Right click "Class1.cs" within your project and choose rename. Give the class a name such as "QMod". Visual Studio will helpfully rename the class for you, as well as the file.

You'll want to refer to those lovely references we added, so first up, add these statements. This will also set you up with a great feature of QMod, which is a log file that you can write to for debugging:

```csharp
using System.Reflection;
using HarmonyLib;
using QModManager.API.ModLoading;
using Logger = QModManager.Utility.Logger;
```

You'll want your class to be static, and you'll need Harmony directives to tell it how to patch in your code:

Update your class and add the directives and Patch method as shown in the code below:

```
namespace MyFirstSubnauticaMod_BZ
{
    [QModCore]
    public static class QMod
    {
        [QModPatch]
        public static void Patch()
        {
        }
    }
}
```

We'll need to tell Harmony to patch our code, so add this into the Patch() method:

```
public static void Patch()
{
    var assembly = Assembly.GetExecutingAssembly();
    var modName = ($"<someuniquevalue>_{assembly.GetName().Name}");
    Logger.Log(Logger.Level.Info, $"Patching {modName}");
    Harmony harmony = new Harmony(modName);
    harmony.PatchAll(assembly);
    Logger.Log(Logger.Level.Info, "Patched successfully!");
}
```

The "modName" generated above needs to be unique across mods – this prevents mods interfering or otherwise messing with each other. The <someuniquevalue> should be replaced with something, well, unique. I tend to use my Discord / Github username, but it's entirely up to you:

```
var modName = ($"mroshaw_{assembly.GetName().Name}");
```

> **NOTE:** You'll see some calls to the Logger here. These are super useful to help debug your code. Just remember to use Level.Info for high level logging only. For detailed logging, especially in code that gets called a lot, remember to use Level.Debug. The last thing you need is to fill players log files with millions of lines of debug text!

Okay, so we probably want to do something with the game code! Again, personal preference, but I like to create a new class for each class that I'm impacting. For the tutorial, we'll just go with something simple. Do that by right clicking on your project and selecting "Add item…". Select "Class" and give it a sensible name. For this tutorial, we'll go for "MyFirstMod.cs".

At this stage, your Visual Studio project should look a bit like this:
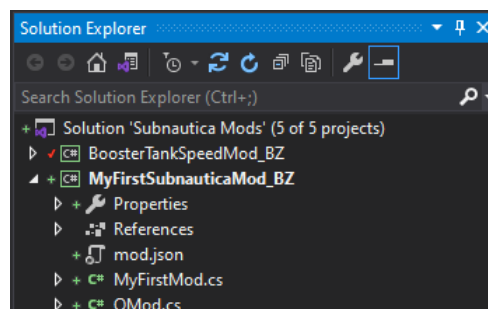
Open up new class file and add these using statements. This gives us the basics of what we need to tell Harmony what we're up to, and for us to reference game objects within our mod code:

```
using HarmonyLib;
```

We're going to patch the "Knife", and make it do mega damage. So, we'll tell Harmony to patch the "Awake" method of the "Knife" class:

```
using HarmonyLib;
using UnityEngine;

namespace MyFirstSubnauticaMod_BZ
{
    /// <summary>
    /// Class to mod the knife
    /// </summary>
    class MyFirstMod
    {
        [HarmonyPatch(typeof(Knife))]
        [HarmonyPatch("Awake")]
        internal class KnifeDamageMod
        {
        }
    }
}
```

The "Start" method is a really useful class method. If implemented in a class, it's something we can use that's always "run only once" when it comes to instances of that class. So, if we want to tweak something when a knife instance is created, this is the place to do it.

> **NOTE:** What you call your classes isn't important. It's the attributes that matter. You can find lots of information about these, and how Harmony works, in the [Harmony user guide.](#)

We're going to manipulate the properties of the Knife, after it's been "started". At this point, we're going to tweak the damage field to make it super powerful. This is what the code looks like:

```
class MyFirstMod
{
    [HarmonyPatch(typeof(Knife))]
    [HarmonyPatch("Start")]
    internal class KnifeDamageMod
    {
        [HarmonyPostfix]
        public static void Postfix(Knife __instance)
        {
            // Double the knife damage
            float knifeDamage = __instance.damage;
            float newKnifeDamage = knifeDamage * 2;
            __instance.damage = newKnifeDamage;
            Logger.Log(Logger.Level.Debug, $"Knife damage was: {knifeDamage}," +
                    $" is now: {newKnifeDamage}");
        }
    }
}
```

And we're done!

You can build this now by right clicking the project and selecting "Build". All going well, you'll see "Build succeeded". If not, go back through and check your code. You can also find the full source for this part of the tutorial on my GitHub.

TESTING YOU MOD

You should now be able to fire up the game and do some testing.

Once you've launched Subnautica, go into Options > Mods and check "Enable debug logs":
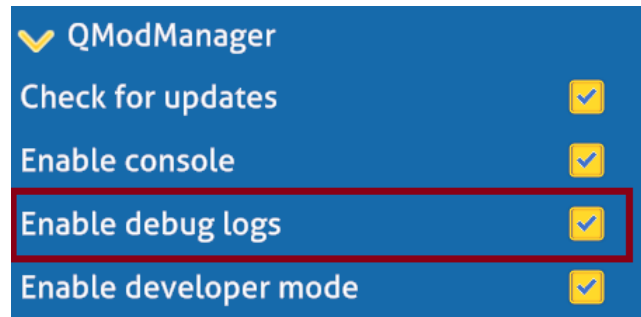
Exit the game and launch it once again. Load up a game and equip your knife – take a few swings while you're at it. You should now alt tab back to your desktop and get back into Windows Explorer. Navigate to your <game> folder, and you should see a file called: qmodmanager_log-SubnauticaZero.txt.
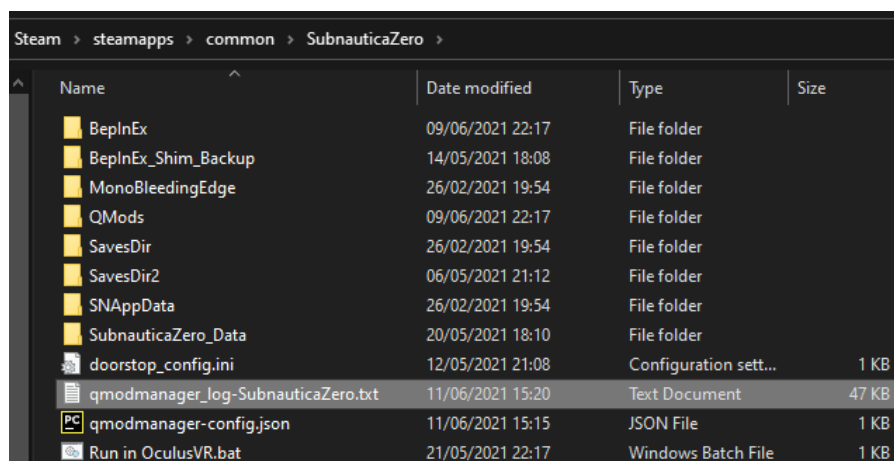
Open this in Notepad or your favourite text editor and have a look. You'll see lots of information in here, and you should also see entries prefixed with your mod name:

```
[SMLHelper/Debug] Attempting to handle console command: tools
[SMLHelper/Debug] No command listener registered for [tools].
[MyFirstSubnauticaMod_BZ:DEBUG] Knife damage was: 20, is now: 40
```

Now get out there and stab some sea life!

CHECKING IN TO GITHUB

Well, mod created. Good work! If you took my advice and registered for an account on GitHub – good for you! You won't regret it! Now you've tested your first build, you can check your code into GitHub. This is easy in Visual Studio, or you can use GitHub Desktop.

In Visual Studio, right click your "Solution", not your Project, and select "Git > Commit or Stash". You can now commit your code and start to share and collaborate.

Well done!

The Modding Process

Okay, so you can ignore this section if you like. I think it's useful to talk a bit about planning, designing and balancing your mod.

There are a load of reasons for modding a game and it can be a very personal thing. In fact, many mod authors build mods for themselves, to make the game more like the game they themselves wanted it to be. They share their mods because, well, because they're generally very nice people who just like to share the love of their favourite games with people who feel the same way.

That said, if you want to build mods to enhance the game, build on the universe and systems that the game presents, and you want it to enhance the game of others, it's worth having a plan. Think about what you're trying to achieve and how it will "play" for other players and how you might balance some pro's and con's of what you're bringing into the game world. While you're doing that, it's also worthwhile thinking about what objects, classes and methods you might need to work with to build your wonder mod.

Plan your mod

<TODO>

Balancing

<TODO>

<GIVE AND TAKE?>

Using dnSpy

<TODO>

<A BIT ABOUT SEARCHING FOR CLASSES, METHODS, FIELDS, CONSTANTS>

Allowing player configuration

<TODO>

<SML HELPER, VARIOUS CONTROL TYPES, ALLOWING DYNAMIC CHANGES>

Harmony(X)

<TODO>

<A BIT ABOUT HARMONY FRAMEWORK>

Manipulating the game

<TODO>

<HOW TO WORK WITH METHOD HOOKS, MANIPULATING INSTANCE FIELDS, GAMEOBJECTS AND UNITY>

Case Study: Booster Tank Speed Mod

The process in practice

Plan your mod

<TODO>

Balancing

<TODO>

Using DNSpy

<TODO>

Allow player configuration

<TODO>

Harmony(X)

<TODO>

Manipulating the game

<TODO>