Modding

My First Slime Rancher Mod (Modding Guide)



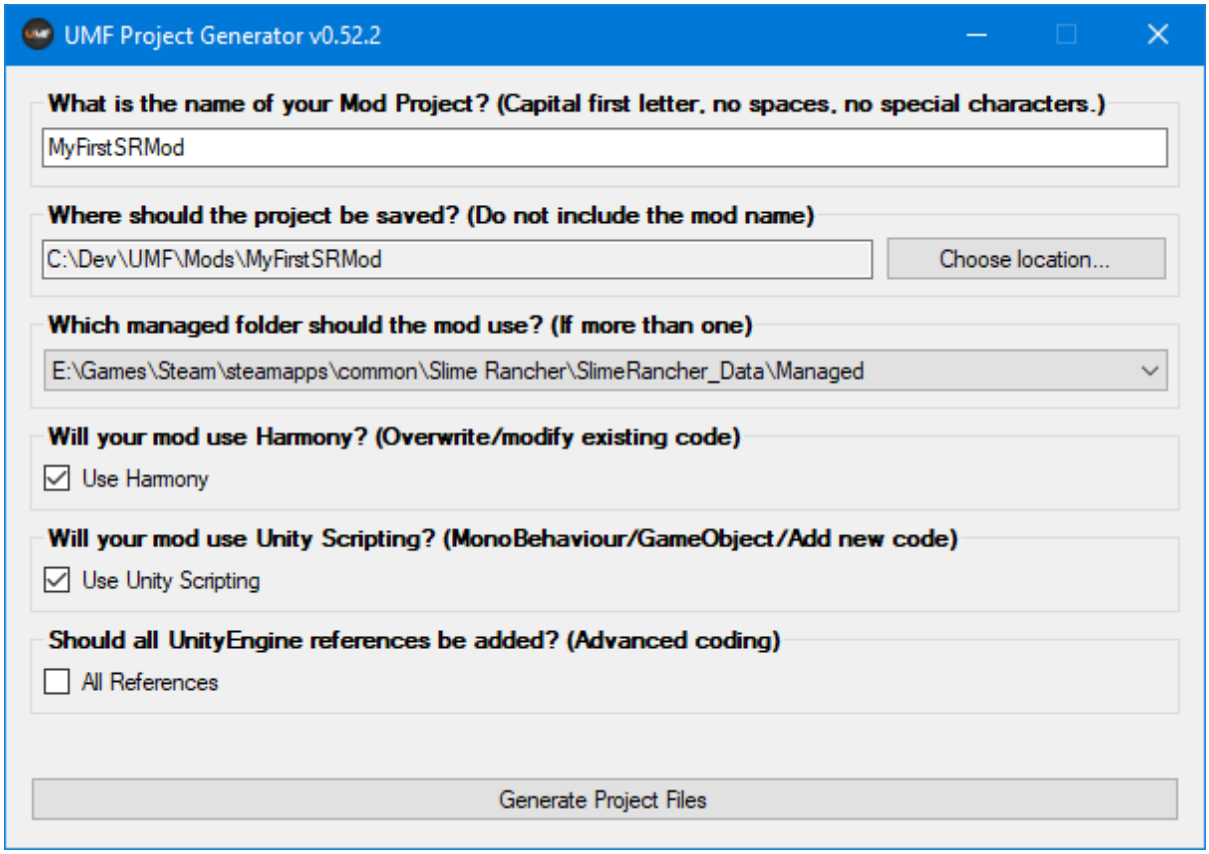
This guide will help you create a simple Slime Rancher Mod for UMF that lets you modify your max health.
It will guide you through all the different methods you could use to modify the players max health, along with the method best suited for this particular task.
It is up to you to determine which method is better suited for anything else you might attempt to mod.

Table of Contents

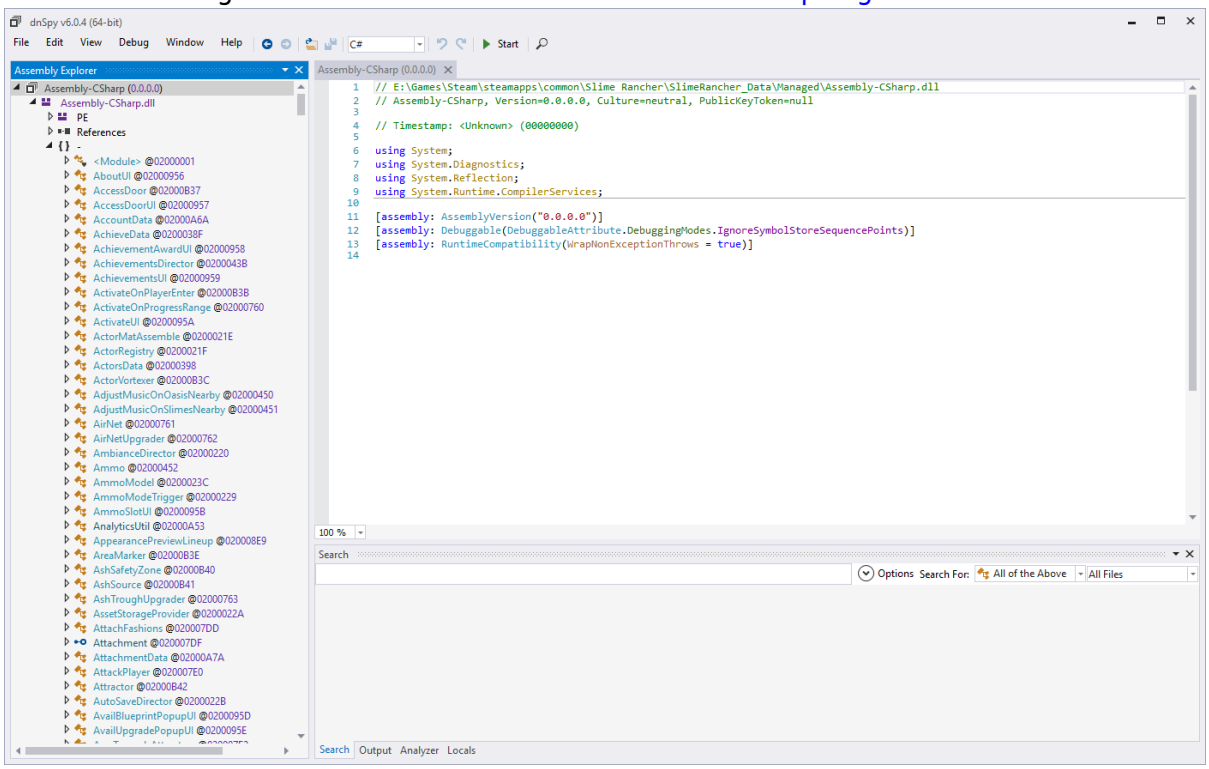
- Preliminary Setup
- Method 1 (Unity Scripting)
- Method 2 (Harmony Postfix)
- Method 3 (Harmony Traverse)
- Method 4 (Harmony Transpiler)
- Method 5 (UMF Patch)
- Method 6 (Optimal)
- Building
- Finalizing

Preliminary Setup

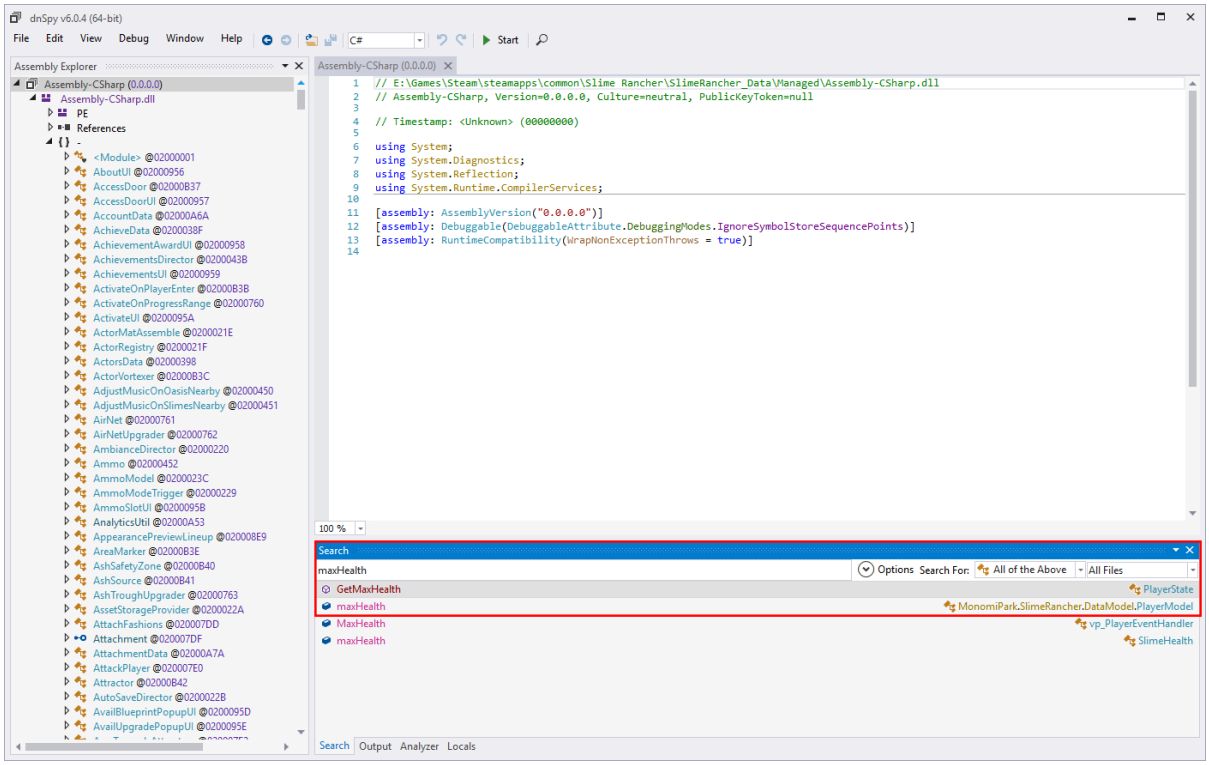
1. Start by creating the project files as seen in Mod Creation.
    - Use the parameters as seen in the image below.



2. Open the newly generated solution in Visual Studio.
3. Open and browse the game code as seen in Mod Creation: Decompiling Game Code.

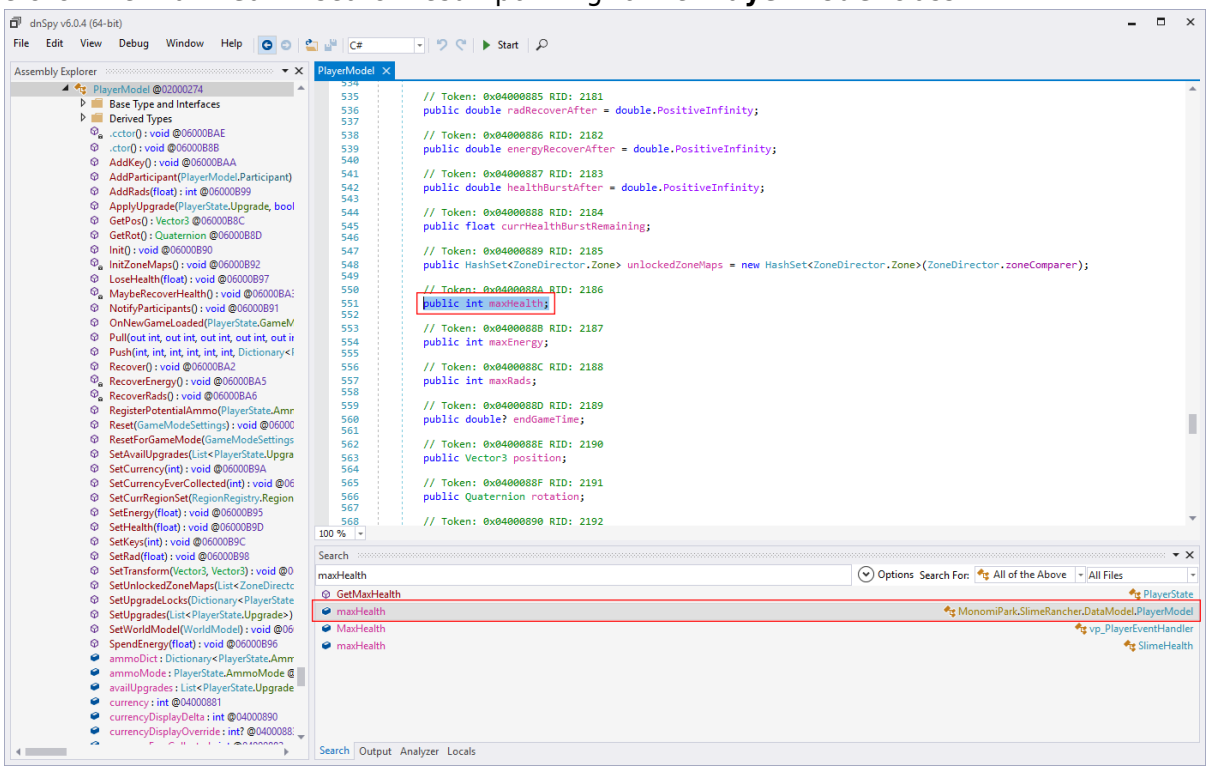


4. Use the search bar in dnSpy to search for `maxHealth`.



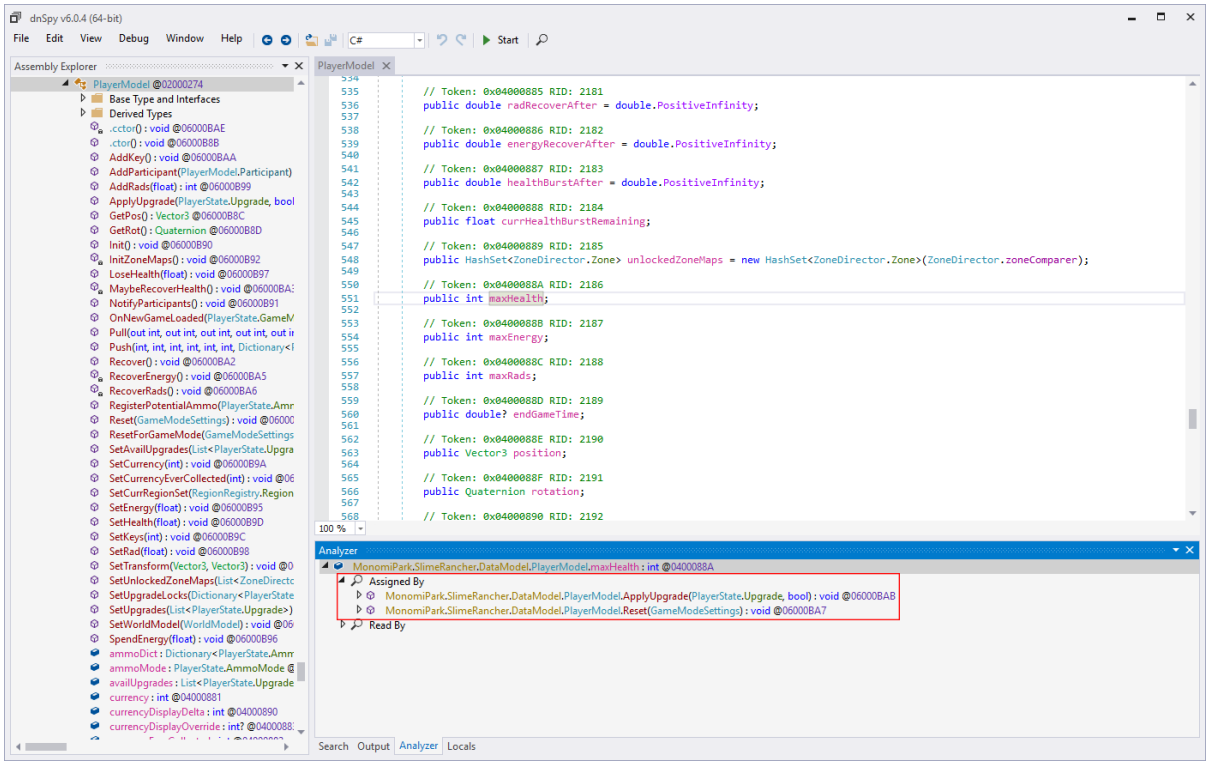
- Here we can see that the max health is retrieved by the **PlayerState** through the **GetMaxHealth** function for reading purposes.
- Double clicking **GetMaxHealth** will also reveal that it retrieves the real maxHealth value from the **PlayerModel**, just as seen in the second search result.

5. Double click the **maxHealth** search result pointing to the **PlayerModel** class.



- Here we can see that maxHealth is stored as a 32bit **int**.

6. Right click **maxHealth** and select **Analyze**.



- Here we see that **maxHealth** is assigned by **PlayerModel.ApplyUpgrade** and **PlayerModel.Reset**.
- Further analyzing of **ApplyUpgrade** and **Reset** will teach you that **ApplyUpgrade** is used by the in-game upgrade console to apply upgrades the player has purchased, including max health upgrades. The **Reset** function is used when a save is loaded or when a new game is started and also resets the max health.
- If we look at the **Reset** function we see that the default health without any upgrades is 100.
- If we look at the **ApplyUpgrade** function we see that **maxHealth** is 150, 200, 250, and 350 depending on the health upgrade the player has.
- We have now learned that **maxHealth** is of the **int** type and that **ApplyUpgrade** and **Reset** will modify the **maxHealth**.

7. Create a MaxHealth config setting in `MyFirstSRModConfig.cs` with the following code.
   - Add the following line below

```
//Add your config vars here.
```

```
public static int MaxHealth;
```

   - Add the following line below

```
//Add your settings here
```

```
MaxHealth = cfg.Read("MaxHealth", new UMFConfigInt(999, 1, 9999),
"This is the player's max health.");
```

   - Your config class should now look something like this:

```
using System;
using UModFramework.API;
```

```
namespace MyFirstSRMod
{
    public class MyFirstSRModConfig
    {
        private static readonly string configVersion = "1.0";

        //Add your config vars here.
        public static int MaxHealth;

        internal static void Load()
        {
            MyFirstSRMod.Log("Loading settings.");
            try
            {
                using (UMFConfig cfg = new UMFConfig())
                {
                    string cfgVer = cfg.Read("ConfigVersion", new
UMFConfigString());
                    if (cfgVer != string.Empty && cfgVer !=
configVersion)
                    {
                        cfg.DeleteConfig(false);
                        MyFirstSRMod.Log("The config file was
outdated and has been deleted. A new config will be generated.");
                    }

                    //cfg.Write("SupportsHotLoading", new
UMFConfigBool(false)); //Uncomment if your mod can't be loaded
once the game has started.
                    cfg.Read("LoadPriority", new
UMFConfigString("Normal"));
                    cfg.Write("MinVersion", new
UMFConfigString("0.52.1"));
                    //cfg.Write("MaxVersion", new
UMFConfigString("0.54.99999.99999")); //Uncomment if you think
your mod may break with the next major UMF release.
                    cfg.Write("UpdateURL", new
UMFConfigString(""));
                    cfg.Write("ConfigVersion", new
UMFConfigString(configVersion));

                    MyFirstSRMod.Log("Finished UMF Settings.");

                    //Add your settings here
                    MaxHealth = cfg.Read("MaxHealth", new
UMFConfigInt(999, 1, 9999), "This is the player's max health.");

                    MyFirstSRMod.Log("Finished loading
settings.");
                }
            }
```

```
            catch (Exception e)
            {
                MyFirstSRMod.Log("Error loading mod settings: " +
e.Message + "(" + e.InnerException?.Message + ")");
            }
        }
    }
}
```

- You have now added a config setting that can be adjusted from the UMF Menu in-game.

8. Proceed by testing out all the different methods you can mod the max health below.

Method 1 (Unity Scripting)

This method uses only Unity Scripting and the game's own code to modify the max health value.
You can find this method by looking through the game code which will take some experience and analyzing.
For this particular scenario this method is not the best to use.

In Slime Rancher the **PlayerModel** class instance can be accessed through the **GameModel** instance class which in turn can be retrieved from the **SceneContext** using the **SRSingleton** class.

We can set this value directly using the following line:

```
SRSingleton<SceneContext>.Instance.GameModel.GetPlayerModel().maxHealth =
MyFirstSRModConfig.MaxHealth;
```

This would set maxHealth to whatever the config value is set to, in the default case 999.
However we have to set this in a way that it is done after **ApplyUpgrade** and **Reset** which is why the other methods are normally better for this case. So to ensure the maxHealth is always set after those we can apply it using Unity Scripting's Update function, which require some extra checks to ensure we are in the game.

1. In your `MyFirstSRMod.cs` add the following at the top:

   ```
   using MonomiPark.SlimeRancher.DataModel;
   ```

2. Comment out the **[UMFHarmony(1)]** line.

   ```
   //[UMFHarmony(1)]
   ```

3. Add the following line to the class:

   ```
   private static PlayerModel playerModel;
   ```

4. Add the following to the bottom of the Awake method:

```
MyFirstSRModConfig.Load();
```

5. Add the following code to the **Update** function.

```
if (!Levels.isSpecial() &&
SRSingleton<SceneContext>.Instance?.GameModel != null) //Makes sure we
are in game and that the GameModel exists.
{
    playerModel =
SRSingleton<SceneContext>.Instance.GameModel.GetPlayerModel();
}
if (playerModel != null) //Make sure that the PlayerModel has been
retrieved.
{
    playerModel.maxHealth = MyFirstSRModConfig.MaxHealth; //Set the max
health to our config value.
}
```

6. Your `MyFirstSRMod.cs` should now look something like this:

```
using UnityEngine;
using UModFramework.API;
using System;
using System.Linq;
using System.Collections.Generic;
using MonomiPark.SlimeRancher.DataModel;

namespace MyFirstSRMod
{
    //[UMFHarmony(1)] //Set this to the number of harmony patches in
your mod.
    [UMFScript]
    class MyFirstSRMod : MonoBehaviour
    {
        private static PlayerModel playerModel;

        internal static void Log(string text, bool clean = false)
        {
            using (UMFLog log = new UMFLog()) log.Log(text, clean);
        }

        [UMFConfig]
        public static void LoadConfig()
        {
            MyFirstSRModConfig.Load();
        }

        void Awake()
        {
            Log("MyFirstSRMod v" + UMFMod.GetModVersion().ToString(),
```

```
true);
            UMFGUI.RegisterPauseHandler(Pause);
            MyFirstSRModConfig.Load();
        }

        public static void Pause(bool pause)
        {
            TimeDirector timeDirector = null;
            try
            {
                timeDirector =
SRSingleton<SceneContext>.Instance.TimeDirector;
            }
            catch { }
            if (!timeDirector) return;
            if (pause)
            {
                if (!timeDirector.HasPauser()) timeDirector.Pause();
            }
            else timeDirector.Unpause();
        }

        void Update()
        {
            if (!Levels.isSpecial() &&
SRSingleton<SceneContext>.Instance?.GameModel != null) //Makes sure we
are in game and that the GameModel exists.
            {
                playerModel =
SRSingleton<SceneContext>.Instance.GameModel.GetPlayerModel();
            }
            if (playerModel != null) //Make sure that the PlayerModel
has been retrieved.
            {
                playerModel.maxHealth = MyFirstSRModConfig.MaxHealth;
//Set the max health to our config value.
            }
        }
    }
}
```

    - This will make sure that your max health is always 999 every single frame regardless of which health upgrade the player has. This is obviously not normally recommended.
    - This code also ensures the config value takes effect immediately upon applying it in the UMF Menu.
    - For more information on the **Awake** function, see Unity Script Reference: Awake.
    - For more information on the **Update** function, see Unity Script Reference: Update.
7. See Building for the next steps.
    - WARNING: If this is the first method you try, you will also need to comment out `Patch_PURPOSEOFPATCH.cs` before building.

Method 2 (Harmony Postfix)

This method is the easiest way to do it, but it's still not entirely optimal.

1. If you did Method 1 before this method then perform the following steps:
    1. Uncomment the UMFHarmony attribute in `MyFirstSRMod.cs`.

        ```
        [UMFHarmony(1)]
        ```

    2. Comment out the extra config load you added to the awake function.

        ```
        //MyFirstSRModConfig.Load();
        ```

        - This is because the UMFHarmony will automatically trigger the UMFConfig attribute before applying the patches in the mod.
    3. Uncomment everything in `Patch_PURPOSEOFPATCH.cs`.
    4. Comment out any code you added with other methods so they do not conflict with what you are currently doing.
    5. Your `MyFirstSRMod.cs` should now look something like this:

        ```
        using UnityEngine;
        using UModFramework.API;
        using System;
        using System.Linq;
        using System.Collections.Generic;
        using MonomiPark.SlimeRancher.DataModel;

        namespace MyFirstSRMod
        {
            [UMFHarmony(1)] //Set this to the number of harmony patches in
        your mod.
            [UMFScript]
            class MyFirstSRMod : MonoBehaviour
            {
                internal static void Log(string text, bool clean = false)
                {
                    using (UMFLog log = new UMFLog()) log.Log(text,
        clean);
                }

                [UMFConfig]
                public static void LoadConfig()
                {
                    MyFirstSRModConfig.Load();
                }
        ```

```
        void Awake()
        {
            Log("MyFirstSRMod v" +
UMFMod.GetModVersion().ToString(), true);
            UMFGUI.RegisterPauseHandler(Pause);
        }

        public static void Pause(bool pause)
        {
            TimeDirector timeDirector = null;
            try
            {
                timeDirector =
SRSingleton<SceneContext>.Instance.TimeDirector;
            }
            catch { }
            if (!timeDirector) return;
            if (pause)
            {
                if (!timeDirector.HasPauser())
timeDirector.Pause();
            }
            else timeDirector.Unpause();
        }
    }
}
```

2. Rename `Patch_PURPOSEOFPATCH.cs` to `Patch_MaxHealth.cs` along with the class name for it.
3. Add the following to the top of `Patch_MaxHealth.cs`:

```
using MonomiPark.SlimeRancher.DataModel;
```

    - This lets us target the **PlayerModel** class which is inside the **MonomiPark.SlimeRancher.DataModel** namespace without having to type the full namespace everytime.
4. Set **typeof** in the first HarmonyPatch attribute to use the **PlayerModel** class we discovered with dnSpy.

```
[HarmonyPatch(typeof(PlayerModel))]
```

    - You could also use **MonomiPark.SlimeRancher.DataModel.PlayerModel** instead for the type.
5. Set the second **HarmonyPatch** attribute to the **ApplyUpgrade** function we discovered with dnSpy.

```
[HarmonyPatch("ApplyUpgrade")]
```

6. Inside the patch class create the following postfix function:

```
        public static void Postfix(PlayerModel __instance)
        {
```

```
        }
```

7. Add the following line to our new function:

```
__instance.maxHealth = MyFirstSRModConfig.MaxHealth;
```

   - At this point the max health will be set regardless of which health upgrades the player has due to this function being run everytime a save is loaded.
   - Since this function is only run on a save load or when the player buys a new upgrade, changes to the config will not take effect immediately like this.

8. Your `Patch_MaxHealth.cs` should now look something like this:

```
using UnityEngine;
using HarmonyLib;
using MonomiPark.SlimeRancher.DataModel;

namespace MyFirstSRMod.Patches
{
    [HarmonyPatch(typeof(PlayerModel))]
    [HarmonyPatch("ApplyUpgrade")]
    static class Patch_MaxHealth
    {
        public static void Postfix(PlayerModel __instance)
        {
            __instance.maxHealth = MyFirstSRModConfig.MaxHealth;
        }
    }
}
```

   - A Harmony **Postfix** patch will make the code in it execute at the end of the function when all other code in the original function has executed.
   - You could change it to a **Prefix** by simply renaming the function to **Prefix**. This would cause the code to be run before the other code in the original function. However that would not work for this scenario since the other code then overwrites our max health again.
   - It would also be a good idea to set the current health to be equal to your max health here, however we show you this in Method 6.
   - You can also alternatively apply this patch to the **Reset** function instead if you do not have any upgrades at all.

9. See Building for the next steps.

Method 3 (Harmony Traverse)

Sometimes a field/variable can be private or readonly and you may not be able to directly modify it

through the game code or a harmony postfix/prefix patch.
In these cases Harmony's Traverse comes in handy.
This Method assumes you have successfully completed Method 2 first.

1. Comment out the only line in the **Postfix** function of `Patch_MaxHealth.cs`.

```
//__instance.maxHealth = MyFirstSRModConfig.MaxHealth;
```

2. Add the following line to the **Postfix** function:

```
Traverse.Create(__instance).Field<int>("maxHealth").Value =
MyFirstSRModConfig.MaxHealth;
```

3. Your `Patch_MaxHealth.cs` should now look something like this:

```
using UnityEngine;
using HarmonyLib;
using MonomiPark.SlimeRancher.DataModel;

namespace MyFirstSRMod.Patches
{
    [HarmonyPatch(typeof(PlayerModel))]
    [HarmonyPatch("ApplyUpgrade")]
    static class Patch_MaxHealth
    {
        public static void Postfix(PlayerModel __instance)
        {
            //__instance.maxHealth = MyFirstSRModConfig.MaxHealth;
            Traverse.Create(__instance).Field<int>("maxHealth").Value =
MyFirstSRModConfig.MaxHealth;
        }
    }
}
```

    - The Traverse will bypass any private or readonly variable and let you read/write to it.
    - Traverse does not need to be used inside a Harmony patch, it can be used anywhere anytime in any code, however you have to make sure it doesn't try to access code that has not yet been loaded into the game.
    - This will function exactly like Method 2, and is wholly unnecessary since **maxHealth** is public and not readonly.
4. See Building for the next steps.

Method 4 (Harmony Transpiler)

This method will show you how you can use a Transpiler to overwrite code in memory rather than inject new code into an existing function.
This method is really solid for when you really need to modify something that can't be otherwise modified with Method 1 or 2.

1. If you have followed the methods in order like you should, then perform the following steps:
    1. Comment out the **Postfix** function in `Patch_MaxHealth.cs`.
2. Add the following variable to `MyFirstSRModConfig.cs`:

```
public static float MaxHealthFloat;
```

3. Below the **MaxHealth** config being loaded add the following line:

```
MaxHealthFloat = MaxHealth;
```

4. Your `MyFirstSRModConfig.cs` should now look something like this:

```
using System;
using UModFramework.API;

namespace MyFirstSRMod
{
    public class MyFirstSRModConfig
    {
        private static readonly string configVersion = "1.0";

        //Add your config vars here.
        public static int MaxHealth;

        public static float MaxHealthFloat;

        internal static void Load()
        {
            MyFirstSRMod.Log("Loading settings.");
            try
            {
                using (UMFConfig cfg = new UMFConfig())
                {
                    string cfgVer = cfg.Read("ConfigVersion", new
UMFConfigString());
                    if (cfgVer != string.Empty && cfgVer !=
configVersion)
                    {
                        cfg.DeleteConfig(false);
                        MyFirstSRMod.Log("The config file was outdated
and has been deleted. A new config will be generated.");
                    }
```

```
                    //cfg.Write("SupportsHotLoading", new
UMFConfigBool(false)); //Uncomment if your mod can't be loaded once the
game has started.
                    cfg.Read("LoadPriority", new
UMFConfigString("Normal"));
                    cfg.Write("MinVersion", new
UMFConfigString("0.52.1"));
                    //cfg.Write("MaxVersion", new
UMFConfigString("0.54.99999.99999")); //Uncomment if you think your mod
may break with the next major UMF release.
                    cfg.Write("UpdateURL", new UMFConfigString(""));
                    cfg.Write("ConfigVersion", new
UMFConfigString(configVersion));

                    MyFirstSRMod.Log("Finished UMF Settings.");

                    //Add your settings here
                    MaxHealth = cfg.Read("MaxHealth", new
UMFConfigInt(999, 1, 9999), "This is the player's max health.");
                    MaxHealthFloat = MaxHealth;

                    MyFirstSRMod.Log("Finished loading settings.");
                }
            }
            catch (Exception e)
            {
                MyFirstSRMod.Log("Error loading mod settings: " +
e.Message + "(" + e.InnerException?.Message + ")");
            }
        }
    }
}
```

- We need this extra float because we are patching a float in memory now, rather than modifying the maxHealth int variable.
- You could also change the **MaxHealth** from **int** to **float** instead if the int was no longer needed.

5. Add the following lines to the top of `Patch_MaxHealth.cs`:

```
using System.Reflection;
using System.Reflection.Emit;
using System.Collections.Generic;
```

6. Add the following function to the **Patch_MaxHealth** class:

```
        public static IEnumerable<CodeInstruction>
Transpiler(IEnumerable<CodeInstruction> instructions)
        {
            foreach (var instruction in instructions)
            {
                if (instruction.opcode.Equals(OpCodes.Ldc_R4) &&
instruction.operand.Equals(350f))
```

```
                {
                    //yield return new CodeInstruction(OpCodes.Ldc_R4,
MyFirstSRModConfig.MaxHealthFloat);
                    yield return new CodeInstruction(OpCodes.Ldsfld,
typeof(MyFirstSRModConfig).GetField(nameof(MyFirstSRModConfig.MaxHealth
Float), BindingFlags.Public | BindingFlags.Static));
                    continue;
                }
                yield return instruction;
            }
        }
```

7. Your `Patch_MaxHealth.cs` should now look something like this:

```
using UnityEngine;
using HarmonyLib;
using MonomiPark.SlimeRancher.DataModel;
using System.Reflection;
using System.Reflection.Emit;
using System.Collections.Generic;

namespace MyFirstSRMod.Patches
{
    [HarmonyPatch(typeof(PlayerModel))]
    [HarmonyPatch("ApplyUpgrade")]
    static class Patch_MaxHealth
    {
        public static IEnumerable<CodeInstruction>
Transpiler(IEnumerable<CodeInstruction> instructions)
        {
            foreach (var instruction in instructions)
            {
                if (instruction.opcode.Equals(OpCodes.Ldc_R4) &&
instruction.operand.Equals(350f))
                {
                    //yield return new CodeInstruction(OpCodes.Ldc_R4,
MyFirstSRModConfig.MaxHealthFloat);
                    yield return new CodeInstruction(OpCodes.Ldsfld,
typeof(MyFirstSRModConfig).GetField(nameof(MyFirstSRModConfig.MaxHealth
Float), BindingFlags.Public | BindingFlags.Static));
                    continue;
                }
                yield return instruction;
            }
        }
    }
}
```

   - This example assumes you have all health upgrades and only affects the final health upgrade of 350 max health.
   - If you wish to test this with the other health upgrades simply change 350 to the max

health upgrade you are on.
    - In this event we are using an if condition to check for the existence of 350 which in this scenario only happens once in the **ApplyUpgrade** function. If it happened more than once it would be prudent to use different conditions or even replace the instruction directly in it's expected position.
    - You could also add additional conditions for each health upgrade along with a separate config setting for each if you wanted.
    - Making Transpilers require you to be well acquainted with IL code. You can view the IL code of C# code by right clicking something within a method of dnSpy.
    - Transpilers have their specific use scenarios and are sometimes required in order to mod something, but should be avoided by inexperienced modders as they can just as easily break or mess up more code than intended if you are not careful.
    - Commented out but included is an alternative **CodeInstruction** way to patch in the config number or any number just at game start, however this method would require restarting the game each time the config changes.
8. See Building for the next steps.

Method 5 (UMF Patch)

This method overwrites code in in the dll file on disk and is not reversible.
There are some cases where code may be inlined or simply inaccessible without overwriting something in the dll file.
This method should only ever be used for these rare cases, and is certainly the worst one you could use for this max health scenario.



WARNING: This Method is currently still in development and still missing a lot of functionality, as well as has several bugs.

1. If you completed steps from any other method do the following steps:
    1. Comment out the UMFHarmony attribute in `MyFirstSRMod.cs`.

    ```
    //[UMFHarmony(1)]
    ```

    2. Comment out everything in `Patch_MaxHealth.cs` or `Patch_PURPOSEOFPATCH.cs`.
2. Make a backup of Assembly-CSharp.dll.
3. Add the following variable to `MyFirstSRMod.cs`:

```
        private const string UMFPatchMaxHealth =
            "FILE: Assembly-CSharp.dll\n" +
            "TYPE: MonomiPark.SlimeRancher.DataModel.PlayerModel\n" +
            "METHOD: ApplyUpgrade\n" +
```

```
            "FIND: ldc.r4 350\n" +
            "REPLACE: ldc.r4 350\n" +
            "WITH: ldc.r4 999"
        ;
```

4. Add the following lines to the bottom of the **Awake** function:

```
            MyFirstSRModConfig.Load();
            UMFPatch.ApplyPatch("MaxHealthPatch", UMFPatchMaxHealth);
```

5. Your `MyFirstSRMod.cs` should now look something like this:

```
using UnityEngine;
using UModFramework.API;
using System;
using System.Linq;
using System.Collections.Generic;
using MonomiPark.SlimeRancher.DataModel;

namespace MyFirstSRMod
{
    //[UMFHarmony(1)] //Set this to the number of harmony patches in
your mod.
    [UMFScript]
    class MyFirstSRMod : MonoBehaviour
    {
        private const string UMFPatchMaxHealth =
            "FILE: Assembly-CSharp.dll\n" +
            "TYPE: MonomiPark.SlimeRancher.DataModel.PlayerModel\n" +
            "METHOD: ApplyUpgrade\n" +
            "FIND: ldc.r4 350\n" +
            "REPLACE: ldc.r4 350\n" +
            "WITH: ldc.r4 999"
        ;

        internal static void Log(string text, bool clean = false)
        {
            using (UMFLog log = new UMFLog()) log.Log(text, clean);
        }

        [UMFConfig]
        public static void LoadConfig()
        {
            MyFirstSRModConfig.Load();
        }

        void Awake()
        {
            Log("MyFirstSRMod v" + UMFMod.GetModVersion().ToString(),
true);
            UMFGUI.RegisterPauseHandler(Pause);
            MyFirstSRModConfig.Load();
```

```
            UMFPatch.ApplyPatch("MaxHealthPatch", UMFPatchMaxHealth);
        }

        public static void Pause(bool pause)
        {
            TimeDirector timeDirector = null;
            try
            {
                timeDirector =
SRSingleton<SceneContext>.Instance.TimeDirector;
            }
            catch { }
            if (!timeDirector) return;
            if (pause)
            {
                if (!timeDirector.HasPauser()) timeDirector.Pause();
            }
            else timeDirector.Unpause();
        }
    }
}
```

- This same umf patch could be made by creating a file called MaxHealth.umfpatch and putting the contents of the **UMFPatchMaxHealth** variable in there instead. This patch would need to always be included/copied to the Release folder.
- See UMF Patch API to learn more about this system.

6. See Building for the next steps.
   - Don't forget to restore your Assembly-CSharp.dll backup after you are done with this test.

Method 6 (Optimal)

This shows you the most optimal way to mod Max Health in Slime Rancher.
While this is the best way to do it for this scenario, it is not necessarily that for modding other things.
You will have to determine your self which method is better for modding something.
Often times you will have to use more than one either in combination to patch one thing, or for several different things in a mod.

1. Comment out the **MaxHealth** variable in `MyFirstSRModConfig.cs`.

```
//public static int MaxHealth;
```

2. Comment out the loading of the **MaxHealth** variable as well.

```
//MaxHealth = cfg.Read("MaxHealth", new UMFConfigInt(999, 1, 9999),
```

```
"This is the player's max health.");
```

3. Add the following variables to `MyFirstSRModConfig.cs`:

```
        public static int MaxHealth0;
        public static int MaxHealth1;
        public static int MaxHealth2;
        public static int MaxHealth3;
        public static int MaxHealth4;
```

4. Add the follow lines below the **Add your settings here** line:

```
                    MaxHealth0 = cfg.Read("MaxHealth0", new
UMFConfigInt(999, 1, 9999), "The player's max health without any health
upgrades.");
                    MaxHealth1 = cfg.Read("MaxHealth1", new
UMFConfigInt(999, 1, 9999), "The player's max health with the first
health upgrade");
                    MaxHealth2 = cfg.Read("MaxHealth2", new
UMFConfigInt(999, 1, 9999), "The player's max health with the second
health upgrade");
                    MaxHealth3 = cfg.Read("MaxHealth3", new
UMFConfigInt(999, 1, 9999), "The player's max health with the third
health upgrade");
                    MaxHealth4 = cfg.Read("MaxHealth4", new
UMFConfigInt(999, 1, 9999), "The player's max health with the fourth
health upgrade");
```

5. Change the **configVersion** variable to “1.1”.
    - This is done so that the old MaxHealth config variable which is no longer used is also removed from existing older config files.
6. Your `MyFirstSRModConfig.cs` should now look something like this:

```
using System;
using UModFramework.API;

namespace MyFirstSRMod
{
    public class MyFirstSRModConfig
    {
        private static readonly string configVersion = "1.1";

        //Add your config vars here.
        //public static int MaxHealth;
        //public static float MaxHealthFloat;
        public static int MaxHealth0;
        public static int MaxHealth1;
        public static int MaxHealth2;
        public static int MaxHealth3;
        public static int MaxHealth4;

        internal static void Load()
```

```
        {
            MyFirstSRMod.Log("Loading settings.");
            try
            {
                using (UMFConfig cfg = new UMFConfig())
                {
                    string cfgVer = cfg.Read("ConfigVersion", new
UMFConfigString());
                    if (cfgVer != string.Empty && cfgVer !=
configVersion)
                    {
                        cfg.DeleteConfig(false);
                        MyFirstSRMod.Log("The config file was outdated
and has been deleted. A new config will be generated.");
                    }

                    //cfg.Write("SupportsHotLoading", new
UMFConfigBool(false)); //Uncomment if your mod can't be loaded once the
game has started.
                    cfg.Read("LoadPriority", new
UMFConfigString("Normal"));
                    cfg.Write("MinVersion", new
UMFConfigString("0.52.1"));
                    //cfg.Write("MaxVersion", new
UMFConfigString("0.54.99999.99999")); //Uncomment if you think your mod
may break with the next major UMF release.
                    cfg.Write("UpdateURL", new UMFConfigString(""));
                    cfg.Write("ConfigVersion", new
UMFConfigString(configVersion));

                    MyFirstSRMod.Log("Finished UMF Settings.");

                    //Add your settings here
                    //MaxHealth = cfg.Read("MaxHealth", new
UMFConfigInt(999, 1, 9999), "This is the player's max health.");
                    //MaxHealthFloat = MaxHealth;
                    MaxHealth0 = cfg.Read("MaxHealth0", new
UMFConfigInt(999, 1, 9999), "The player's max health without any health
upgrades.");
                    MaxHealth1 = cfg.Read("MaxHealth1", new
UMFConfigInt(999, 1, 9999), "The player's max health with the first
health upgrade");
                    MaxHealth2 = cfg.Read("MaxHealth2", new
UMFConfigInt(999, 1, 9999), "The player's max health with the second
health upgrade");
                    MaxHealth3 = cfg.Read("MaxHealth3", new
UMFConfigInt(999, 1, 9999), "The player's max health with the third
health upgrade");
                    MaxHealth4 = cfg.Read("MaxHealth4", new
UMFConfigInt(999, 1, 9999), "The player's max health with the fourth
health upgrade");
```

```
                    MyFirstSRMod.Log("Finished loading settings.");
                }
            }
            catch (Exception e)
            {
                MyFirstSRMod.Log("Error loading mod settings: " +
e.Message + "(" + e.InnerException?.Message + ")");
            }
        }
    }
}
```

- We have now created a setting for each health upgrade.

7. Add the following function to the **Patch_MaxHealth** class:

```
        public static void Postfix(PlayerModel __instance, ref
PlayerState.Upgrade upgrade)
        {
            switch (upgrade)
            {
                case PlayerState.Upgrade.HEALTH_1:
                    __instance.maxHealth =
MyFirstSRModConfig.MaxHealth1;
                    break;
                case PlayerState.Upgrade.HEALTH_2:
                    __instance.maxHealth =
MyFirstSRModConfig.MaxHealth2;
                    break;
                case PlayerState.Upgrade.HEALTH_3:
                    __instance.maxHealth =
MyFirstSRModConfig.MaxHealth3;
                    break;
                case PlayerState.Upgrade.HEALTH_4:
                    __instance.maxHealth =
MyFirstSRModConfig.MaxHealth4;
                    break;
            }
            __instance.currHealth = __instance.maxHealth;
        }
```

- This ensures we are patching all of the different health upgrades only when they are applied.
- We are still missing the health you have without any health upgrades though.

8. Add the following code to `Patch_MaxHealth.cs`:

```
    [HarmonyPatch(typeof(PlayerModel))]
    [HarmonyPatch("Reset")]
    static class Patch_MaxHealth2
    {
        public static void Postfix(PlayerModel __instance)
        {
            __instance.maxHealth = MyFirstSRModConfig.MaxHealth0;
```

```
            __instance.currHealth = __instance.maxHealth;
        }
    }
```

- Here we also patch in the health you have with no health upgrades as seen in the **Reset** function with dnSpy.

9. Your `Patch_MaxHealth.cs` should now look something like this:

```
using UnityEngine;
using HarmonyLib;
using MonomiPark.SlimeRancher.DataModel;

namespace MyFirstSRMod.Patches
{
    [HarmonyPatch(typeof(PlayerModel))]
    [HarmonyPatch("ApplyUpgrade")]
    static class Patch_MaxHealth
    {
        public static void Postfix(PlayerModel __instance, ref
PlayerState.Upgrade upgrade)
        {
            switch (upgrade)
            {
                case PlayerState.Upgrade.HEALTH_1:
                    __instance.maxHealth =
MyFirstSRModConfig.MaxHealth1;
                    break;
                case PlayerState.Upgrade.HEALTH_2:
                    __instance.maxHealth =
MyFirstSRModConfig.MaxHealth2;
                    break;
                case PlayerState.Upgrade.HEALTH_3:
                    __instance.maxHealth =
MyFirstSRModConfig.MaxHealth3;
                    break;
                case PlayerState.Upgrade.HEALTH_4:
                    __instance.maxHealth =
MyFirstSRModConfig.MaxHealth4;
                    break;
            }
            __instance.currHealth = __instance.maxHealth;
        }
    }

    [HarmonyPatch(typeof(PlayerModel))]
    [HarmonyPatch("Reset")]
    static class Patch_MaxHealth2
    {
        public static void Postfix(PlayerModel __instance)
        {
            __instance.maxHealth = MyFirstSRModConfig.MaxHealth0;
```

```
            __instance.currHealth = __instance.maxHealth;
        }
    }
}
```

- We have also made sure that the current player health is set to the new max health when applied, just like the normal game code does.

10. Change the **UMFHarmony** attribute to 2 in `MyFirstSRMod.cs`.

```
[UMFHarmony(2)]
```

- This tells UMF the number of patches that is expected to be successfully applied, and will show a warning to the user should any of them fail.
- Since you now have two patches, one targeting the **ApplyUpgrade** function, and one targeting the **Reset** function, this number should be 2.

11. Your `MyFirstSRMod.cs` should now look something like this:

```
using UnityEngine;
using UModFramework.API;

namespace MyFirstSRMod
{
    [UMFHarmony(2)] //Set this to the number of harmony patches in your
mod.
    [UMFScript]
    class MyFirstSRMod : MonoBehaviour
    {
        internal static void Log(string text, bool clean = false)
        {
            using (UMFLog log = new UMFLog()) log.Log(text, clean);
        }

        [UMFConfig]
        public static void LoadConfig()
        {
            MyFirstSRModConfig.Load();
        }

        void Awake()
        {
            Log("MyFirstSRMod v" + UMFMod.GetModVersion().ToString(),
true);
            UMFGUI.RegisterPauseHandler(Pause);
        }

        public static void Pause(bool pause)
        {
            TimeDirector timeDirector = null;
            try
            {
                timeDirector =
SRSingleton<SceneContext>.Instance.TimeDirector;
```

```
            }
            catch { }
            if (!timeDirector) return;
            if (pause)
            {
                if (!timeDirector.HasPauser()) timeDirector.Pause();
            }
            else timeDirector.Unpause();
        }
    }
}
```

12. See Building for the next steps.
    - You should now have a fully working Max Health Mod.

Building

1. In Visual Studio in the top bar menu click **Build > Rebuild solution**.
    - Your mod is automatically packed, placed into the game's `Mods` folder, and ready to play.
2. Start the game and test if your mod works.
    - You can verify that the mod works by seeing if your Max Health bar is higher than your current health.
3. Proceed to test out the other methods or see Finalizing.
    - Don't forget to comment out the code from each method so it doesn't affect your test results.

Finalizing

Since this is just an example mod you should not publicize it anywhere since anyone can easily do this.
However for the purpose of completion these steps should be taken before releasing any mod.

1. Edit `Properties\AssemblyInfo.cs` and fill in all the details of your mod.
2. Edit `ModInfo.txt` to whatever you want to show users who install or update the mod.
3. Edit `configVersion` in `MyFirstSRModConfig.cs` to match the version in `AssemblyInfo.cs`.
    - After your first release you should only need to edit this variable when you remove or

rename a config option.

4. Clean and remove or comment out any unused code.

From:
https://umodframework.com/wiki/ - **UMF Wiki**

Permanent link:
https://umodframework.com/wiki/guide_firstsrmod?rev=1562237561

Last update: **2019/07/04 10:52**