DLL Assembly Modding Handbook

July 10, 2023

Contents

1 Introduction 1

2 Major changes in naming 2

3 Overview 2

4 Setting up Visual Studio 3

5 Configurable Options 6

6 Uploading mods from a new machine 7

7 Useful methods 9

8 Mod version targetting 10

9 Adding commands for access via JSON events 11

10 List of cheats with their required arguments 12

1 Introduction

C-Sharp Assemblies allow nearly every part of Shadows to be modded, and are how custom Gods, Agents, challenges and other complex content can be added. The process involves taking the game's own DLL Assembly, which lays out which classes and methods are available, importing it into Visual Studio, building a new set of classes and functions and exporting those back, also in DLL Assembly form. Once read in, the game will run the code alongside its own.

The code contains a set of hooks, allowing easy access to a number of core game systems, letting a mod easily and smoothly inject AI decisions, alter the map during map generation, update itself after the end of a turn or take action in a range of other situations. A sufficiently motivated modder could of course ignore most of these hooks, they are only provided for convenience, not as a mandatory framework to employ.

Due to the nature of game development, and limited resources, not all code will be documented, nor can support be provided. If this handbook and provided modding framework is useful you are welcome to make use of it, and if I can I will answer questions on discord, but much of the resources are simply provided 'as is'. Certain parts of the code base are inelegantly constructed or named, due to changes during development, leading to a few quirks which modders will need to work around.

2 Major changes in naming

Various parts of the game are named differently internally than they are in-game.

1. **Modifiers/Properties** Modifiers are termed 'properties' internally. They all use the code prefix 'Pr_'.

2. **Rituals/Unique Challenges** Any challenge which is held by the unit, rather than associated with a location or property is termed a 'ritual', and uses the prefix 'Rti_'. This is mostly because the first unique abilities were all magical. It is also important to note that because the challenges all assume they have a location, the rituals also must be given the unit's own location, even if this is not overly applicable.

3. **Heroes are Agents**. Units follow a naming prefix system, whereby all agents/heroes/acolytes are 'agents'. They are prefixed UA, with agents being assigned class names starting in "UAE_" for "Unit Agent Evil". Heroes are "UAG" for "Unit Agent Good" and Acolytes are "UAA" for "Unit Agent Acolytes". Neural evil agents, such as non-controlled Deep Ones, are "UAEN" for "Unit Agent Evil Neutral". They all sub-class the associated class, to allow the game to identify them properly.

3 Overview

C-Sharp modding is intended to take place by overriding functions in subclasses. This is roughly how the game itself operates. Initially, the mod starts with the mod kernel, found in Assets.Code.Modding.ModKernel. If the DLL Assembly contains one, it will be loaded as a mod, and its various hooks called. These give you a point to inject your code into the various parts of the game.

You should create challenges, agents and other game concepts by subclassing the relevant super-class. These will then define a set of functions which you can employ, in a standard object-orientated way. The game will be able to recognise these, including saving them to disk and reloading the game and recognising that their logic depends on the modded DLL.

Using the save/load system is mostly handled automatically. Any variable marked as public will be saved. Critically, since these are being serialized, no reference to Unity concepts nor to large data structures such as images can exist. Sprites are to be referenced by a method, rather than saved directly on the object. The EventManager's mod image system is intended to be used for this purpose, for example "return EventManager.getImg("insect.iconDeadFish.png");" being used to provide an icon Sprite for a property. If the Sprite were loaded and saved directly as a variable on the property the save/load could not occur. **Storing a reference to 'map', the central object of the game's data, is common and recommended, but storing a reference to the 'world', which serves as a link between UI elements, or to any UI element, is not possible and will break the save/load system**.

4 Setting up Visual Studio

This guide assumes that Visual Studio will be used for the production of C-Sharp DLLs. It's possible that other IDEs support this process, but those are outside the scope of this work.

It might potentially, if you are using advanced features, also be necessary to download Unity, to extract the DLL assemblies relating to Sprites. Certain DLLs are automatically distributed in the compiled version, under **"ShadowsOfForbiddenGods_Data/Managed"**
. If others are needed, for example for UI changes, these will need to be obtained from your own Unity project under its own license. Legally, the DLLs can't simply be uploaded to the modding repository, sadly. **It is recommended to make copies, not reference the game files directly**

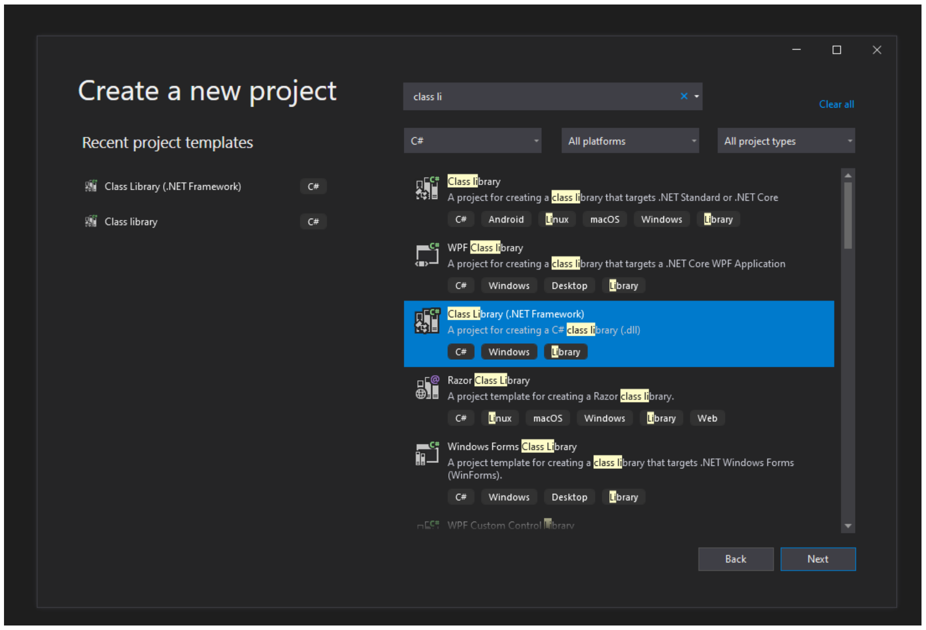
You will need, at the very minimum, the game's own code Assembly, which defines Shadows's code structure ("Assembly-CSharp.dll") and Unity's main engine DLL, "UnityEngine.CoreModule.dll", both of which are available at in the aforementioned 'Managed' folder, in the game files. Make a copy of these in an easily accessed location.

You will need the '.NET desktop development' addon for Visual Studio to give you the required template to make a DLL, it is available via the Visual Studio installer.

Start a new Visual Studio C-Sharp project. It needs to be following the 'class library' template (You may need to obtain new templates if it is not pre-installed). **IT MUST BE A 'A project for creating a C# class library (.dll)', NOT 'A project for creating a class library that targets .NET standard or .NET core'. Trust me, I spent an entire day trying to figure out what was wrong when I clicked the wrong one here, learn from my mistakes**.

With regards to targetted .NET Version, the test mod, the Insect God, uses .NET version 4.7.2 and it seems to work well.

After this, you need to reference the game's DLLs, so your project can make use of its classes. Right click on 'references', in the top right (see Figure 2), then



A project for creating a class library that targets .NET Standard or .NET Core

A project for creating a class library that targets a .NET Core WPF Application

A project for creating a C# class library (.dll)

A project template for creating a Razor class library.

A project template for creating a class library that targets .NET Windows Forms (WinForms).

Figure 1: Select a class library template (note it must be the .dll one, not the standard .NET one)

select ‘Add Reference’, then select ‘browse’ and find the DLLs (both Assembly-CSharp.dll and Unity Engine).

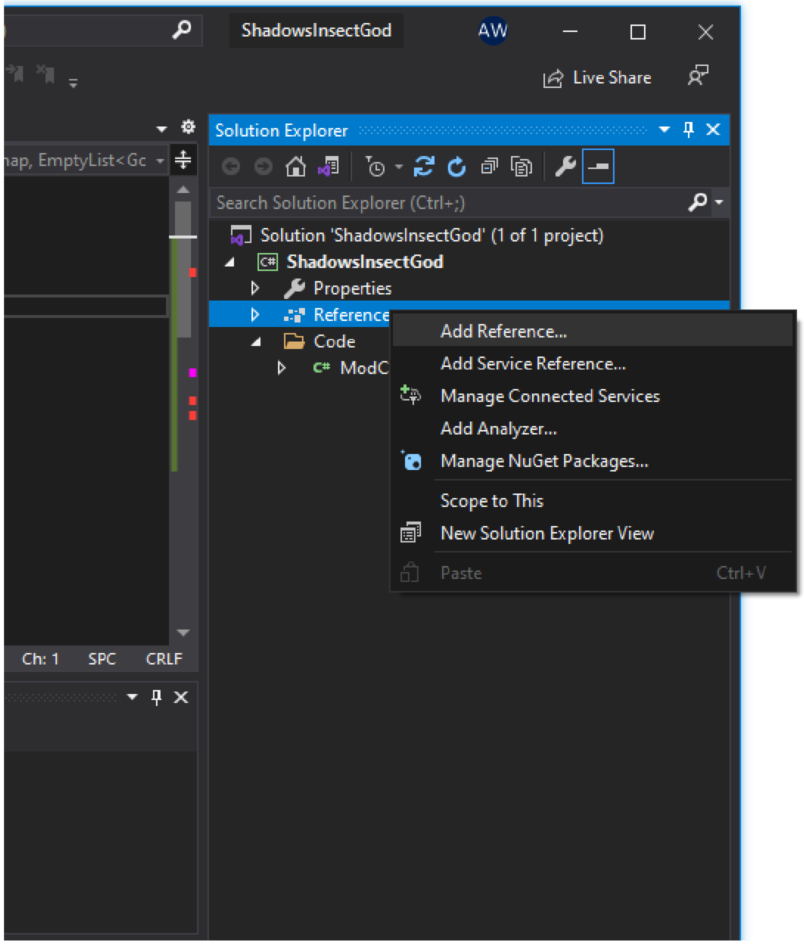


Figure 2: Add references to the DLLs

Once added, you should be able to add a “using Assets.Code” to the top of your files, and begin to access Shadows’ code structure.

For your mod to work, you must implement a class which subclasses “Modding.ModKernel”.

When your mod is ready for initial testing, go to “build” in the top of the Visual Studio window, then ”build foo”, where ‘foo’ is your project name. The DLL will then appear your project’s “bin/Debug” folder.

For a minimal worked example, see the “CSharp_MinimumWorkedExample” folder in the modding github repository. It contains both the source code, and an example of the DLL placed into a mod. This mod folder can be dropped into the game’s “data/optional_data” folder, and it will run.

5 Configurable Options

To allow your mod to have options, you can specify a set of integers and booleans in a JSON file which will be configurable by the user, and saved into the save game folder. This JSON must go along with your other files (inside any version-specific folder if you are using them), and be called “mod_config.json”. Options have a name, a description, a min/max (if it’s an integer), a default value and an indication of which type of variable it is.

An example of a JSON, with one of both types of variable, is available below:

```
{
    "options":[
    {
        "name":"option_integer",
        "description":"tests an integer, with a range between 0 and 10",
        "isInteger":"true",
        "minValue":0,
        "maxValue":10,
        "defaultValue":5,
        "defaultBoolValue":"true"

    },
    {
        "name":"option_boolean",
        "description":"tests true/false value settings"
    }
    ]
}
```

Copying this text into a JSON file and storing it alongside your mod should make the 'configure' button appear on the mod list accessed from the main menu. Your mod will receive the values via the 'receiveModConfigOpts_int' and 'receiveModConfigOpts_bool' methods in the modKernel.

6 Uploading mods from a new machine

If you move computers, the game won't be able to recognise that you are the mod's author, so won't let you update it, and instead try to get you to re-publish your own mod. The game records which mods you've uploaded in 'publishedMods.json' which is in the normal saved game folder. Copying this file to your new machine's saved game folder will allow you to update mods again, or you can rebuild the file.

You will need the workshop ID for the mod. This can be found in the workshop item's URL. For example 'https://steamcommunity.com/sharedfiles/filedetails/?id=3002216317' indicates that this item has an ID of 3002216317.

The publishedMod.json file is structured as a single line object called 'items' which is a list of all published mods. It is structured in the following format:

```
{"items":[{"path":"modUploadFolder\VALUE1\mod.json","fileId":{"m_PublishedFileId":VALUE2}}]}
```

Where VALUE1 is the name of the folder you're placing in the mod upload folder, and VALUE2 is the mod ID number. For example this would be:

```
{"items":[{"path":"modUploadFolder\\astrid\\mod.json","fileId":{"m\_PublishedFileId":3002216317}}]}
```

Note the number matching the URL ID.

If you want to update this file for multiple items, you need to put a comma after the SECOND TO LAST curly bracket (before the closing square bracket) and add in a new element with a new path and fileId field.

For example, for a set of three mods:

```
{"items":[
{"path":"modUploadFolder\bonus_insanity\test.json","fileId":{"m_PublishedFileId":2983329175}},
{"path":"modUploadFolder\bonus_insanity\mod.json","fileId":{"m_PublishedFileId":2983330232}},
{"path":"modUploadFolder\astrid\mod.json","fileId":{"m_PublishedFileId":3002216317}}
]}
```

Except all on one line (linebreaks added just for formatting purposes)

7 Useful methods

While not everything can be documented, some useful method are presented below to get you started:

1. **Save/Loading** To save variables, note they need to be marked as public
2. **Eleven** This poorly named class holds the static reference to the random number generator, and a set of other utility functions. So named because the codebase was originally the eleventh attempt at getting a Unity project working, its RNG should be used if you want to exploit the map seed system (Although seeding your own RNG would also be viable)
3. **UNIFIED MESSAGE** These are the main message form presented to the player in the middle of the screen. They are the ones which have one or two people, units, locations or societies, and have a go-to button under each, and a 'dismiss all of this type' button. These are accessible from the map object (which will also store and handle the player-chosen type dismissal). Example usage is: "map.addUnifiedMessage(personA, personB, "Infection", "A person has infected another with the Ophiocord fungal infection", "PERSON INFECTS OTHER");"
   The two arguments are the objects referenced. They can be one of: PERSON, UNIT, LOCATION, SOCIETY. The second one can be null. The third argument is the title of the message, the fourth is the message itself. The fifth, last argument in this example, is the code used to dismiss messages of a given type. By convention it is all upper-case. If the player dismisses this message using the "dismiss all of this type" button, all messages with this code will be silenced. This is automatic, you do not need to perform any checks of your own to see if a given message type is dismissed.
   Note there is an optional boolean argument. This is used purely for debugging purposes, and indicates whether the message will still appear in automatic mode. It can be ignored in most cases.
4. **Custom Cheats** Pressing Left_Shift + Backspace will bring up a tiny text prompt in the top right. This allows you to enter cheat commands for testing your mods. You can override the "onCheatEntered" method in the modKernel to intercept your custom cheats, and enter debug commands you need.
5. **Custom Tags** You can add your own concepts which the AI can gain or lose preference for by calling two static methods on the Tags class. Do this before the game is generated if you want pre-existing characters to have preferences when the world starts. Adding an enemy (generic term, could be a friendly race if you want) is easily done, by calling Tags.addTagEnemy("name"). This will return an integer, which you can include in your tags for different units/items/people to get the AI preferences to recognise them. Adding broader personality concepts, like the existing 'ambition' for example, requires you to also include adjectives to describe the different facets. For example, you could add "technophobe", by calling Tags.addTagConcept("technology", new string[]"luddite","technophobe","techenthusiast","transhumanist");. These four descriptions then reflect hate/dislike/like/love of the term.

8 Mod version targetting

Should you need to have two separate versions of your mod for two separate versions of the game, you can put all files **other than mod_desc.json** into a subfolder. Names directly relate to the version, and are of the form ‘v’ + major number + minor number. 1.0 is therefore “v1.0”, and 0.13 is “v0.13”. Folders named this way will be read preferentially when loading the mod. If no folder exists for a given name, the root of the mod folder will be read.

9 Adding commands for access via JSON events

10 List of cheats with their required arguments

Cheat commands can be entered by pressing left_shift + backspace. This will open a tiny window in the top right, where a command, or a command + a value (such as “skip 25” to skip 25 turns) can be entered. These are undocumented, and may crash the game if used incorrectly. They are provided as-is, to be used only when you have no unsaved changes which may be lost. Often the issue will be that they required something to be selected and either nothing was, or the wrong type of thing was (for example ‘shadow’ requires a unit or human settlement to be selected). They are provided to potentially assist with modding if you can find ones which help you.

1. Simple Command: power
2. Simple Command: testsave
3. Simple Command: testload
4. Simple Command: endless
5. Simple Command: automatic
6. Simple Command: testCast
7. Simple Command: ghast
8. Simple Command: ravenous
9. Simple Command: silence
10. Simple Command: shadow
11. Simple Command: halfshadow
12. Simple Command: 99shadow
13. Simple Command: nationShadow
14. Simple Command: testMsg
15. Simple Command: plague
16. Simple Command: music
17. Simple Command: playback
18. Simple Command: keys
19. Simple Command: fishCurse
20. Simple Command: throughTheirEyes
21. Simple Command: wastingSouls
22. Simple Command: controlFont
23. Command with numeric value: xp
24. Simple Command: 100
25. Command with numeric value: political
26. Simple Command: hateOrc
27. Simple Command: insanity
28. Simple Command: geomancer
29. Simple Command: destroyAll
30. Simple Command: necromancer

31. Simple Command: orctype1
32. Simple Command: orctype2
33. Simple Command: orctype3
34. Simple Command: orctype4
35. Simple Command: orctype5
36. Simple Command: testCustomChallenge
37. Simple Command: windowedMode
38. Simple Command: holy
39. Simple Command: blood
40. Simple Command: madout
41. Simple Command: massInsanity
42. Simple Command: hateLocalLord
43. Simple Command: massDisdain
44. Simple Command: internationalChaos
45. Simple Command: corrupt
46. Simple Command: murderer
47. Simple Command: heroesHateMe
48. Simple Command: removeFaith
49. Simple Command: rotateFaith
50. Simple Command: holyInf
51. Simple Command: daughter
52. Simple Command: firstDaughter
53. Simple Command: iasturArmy
54. Simple Command: maxLevel
55. Simple Command: armsRace
56. Command with numeric value: gainKnowledge
57. Simple Command: testTrade
58. Simple Command: opposition
59. Simple Command: vampire
60. Simple Command: toHero
61. Simple Command: fixate
62. Simple Command: testSpread
63. Simple Command: aggro
64. Simple Command: blockme
65. Simple Command: addmenace
66. Simple Command: add100menace
67. Simple Command: addprofile

68. Simple Command: add1profile
69. Simple Command: hateMe
70. Simple Command: mammonTrade
71. Simple Command: dark empire
72. Simple Command: halfall
73. Requires a unit to be selected. Takes the form “testEventsFrom FOO” where ‘FOO’ is your mod prefix. Opens all events with the selected unit as context
74. Simple Command: alliance
75. Simple Command: darkCrusader
76. Simple Command: enshadowNation
77. Simple Command: path
78. Simple Command: ruined healpot
79. Simple Command: mammoneat
80. Simple Command: mammoneatplus
81. Simple Command: mammongrow
82. Simple Command: mammonult
83. Simple Command: aware
84. Simple Command: testTextSelect
85. Command with numeric value: loseXP
86. Command with numeric value: itempool1
87. Command with numeric value: itempool2
88. Command with numeric value: itempool3
89. Command with numeric value: skip
90. Command with numeric value: addDanger
91. Command with numeric value: landgrab
92. Command with numeric value: testmad
93. Simple Command: ruin
94. Simple Command: hot
95. Simple Command: cold
96. Simple Command: minion
97. Simple Command: minion2
98. Simple Command: minion3
99. Simple Command: minionf
100. Simple Command: miniono
101. Simple Command: allowAllAgents
102. Simple Command: complete
103. Simple Command: internationalView
104. Simple Command: cavTest

105. Simple Command: snow
106. Simple Command: min sanity
107. Simple Command: sgMenace
108. Simple Command: dark coronation
109. Simple Command: die
110. Simple Command: addAncientRuins
111. Simple Command: addAbyssalCity
112. Simple Command: refresh
113. Simple Command: castMember
114. Simple Command: gridlock
115. Simple Command: inflame
116. Simple Command: volcanic
117. Simple Command: civilWar
118. Simple Command: rosebud
119. Simple Command: midas
120. Simple Command: wantgold
121. Simple Command: makeReligious
122. Simple Command: giveTome
123. Simple Command: hateShadow
124. Simple Command: bigsnake
125. Simple Command: FundOrder
126. Command with numeric value: unrest
127. Command with numeric value: famine
128. Command with numeric value: death
129. Command with numeric value: ward
130. Command with numeric value: devastation
131. Command with numeric value: orcPlunder
132. Command with numeric value: magicDuel
133. Command with numeric value: dropPersonal
134. Command with numeric value: vingift
135. Command with numeric value: vinMan
136. Command with numeric value: opha
137. Command with numeric value: ophaNation
138. Command with numeric value: ophanimite
139. Command with numeric value: ophaDoubt
140. Command with numeric value: ophaFesteringDoubt
141. Command with numeric value: ophaTemple

142. Simple Command: submenace
143. Simple Command: mmm
144. Simple Command: international hate
145. Simple Command: warVictim
146. Simple Command: warlike
147. Simple Command: peaceful
148. Simple Command: ambitious
149. Simple Command: awaken
150. Simple Command: deepFreeze
151. Simple Command: madness
152. Simple Command: krorc
153. Command with numeric value: takeDmg
154. Simple Command: meteors
155. Simple Command: victory
156. Simple Command: defeat
157. Simple Command: battleroyale
158. Simple Command: heal
159. Simple Command: antagonist
160. Simple Command: antagonist2
161. Simple Command: combatbuff
162. Simple Command: deepOne
163. Simple Command: deepOneF
164. Command with numeric value: deepCult
165. Simple Command: hunger
166. Simple Command: hungerF
167. Simple Command: panic
168. Simple Command: deepOneAll
169. Simple Command: megabuff
170. Simple Command: infiltrate
171. Simple Command: possess