

TOL mod usage/management guide

Written by Creator Cyfire, last revision: May 14th, 2024

Introduction

After reading this guide, you're expected to know the following stuff:

- How to make local backups for mods acquired from Steam Workshop.
- How to import local mods and/or any mod downloaded from Nexus or anywhere.
- How to "create" in-game usable mods from source folder (e.g. 'ModProject')
- General-purpose knowledge about which/where you can find basic game saves and mod data.

As a reading help, this document includes a table of content which you "could" use to skip certain parts, although I would not necessarily recommend doing so.

The 1st section describes the four top-most (computer) File System top-layer folders; How/why they're named the way they do. The 2nd section describes how to distinguish what folder can be imported as Tale of Immortal mod (and when it's successful/unsuccessful). The 3rd section contains by far the most important information because it's the easiest to overlook or brushed off.

Table of Contents

| | |
|--|---|
| Introduction | 1 |
| 1. Mod folder locations | 3 |
| 1.1 Steam Workshop downloaded mod-folder | 3 |
| 1.2 Local Imported mod-folder | 3 |
| 1.3 Mod creation project folder | 4 |
| 1.4 Local in-game achievements & Main/DLC-game saves directory | 5 |
| 2. How to local import mods | 5 |
| 3 Turning mod source into a playable mod | 6 |
| 3.1 Outdated 'debug/mod' folders | 6 |
| 3.2 How to local-export an import'able mod folder | 6 |
| 4. Problems importing a mod | 7 |
| 4.1 DLLs cannot be overwritten while already-loaded! | 7 |
| 4.2 The in-game import button DOES NOT delete files!! | 7 |
| 5. Advanced info | 8 |
| 5.1 VS project source shows dependency errors | 8 |
| 5.2 How to reverse engineer // source-reconstruct other mods? | 9 |

1. Mod folder locations

Every single mod in this game is contained within a **folder**, and this folder itself can be named anything. Within your computer's storage drives, there are 2 locations from which the game can read **mod folders**. At the Local Mods interface, you can distinguish if a mod is acquired from the Steam Workshop or local imported.

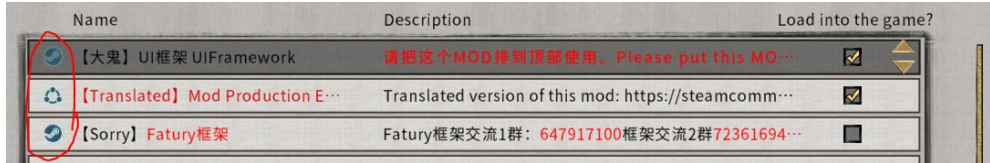


Figure 1: The Steam icon means Steam Workshop, the other icon means 'local import'.

1.1 Steam Workshop downloaded mod-folder

- Windows: <Steam path>\Steam\steamapps\workshop\content\1468810\
- Unix-like: <Steam path>/Steam/steamapps/workshop/content/1468810/

Being common computer-knowledge, the <Steam path> pertains to the location within your computer, in which your Steam games' folder is saved. By default on Windows it is in "C:\Program Files (x86)\\" and if you installed a certain game elsewhere, then you must also know where that would be. The number 1468810 pertains to the Tale of Immortal's game ID that it's known as (by Steam).

Within the Steam Workshop folder, every mod folder is named as an ID, which corresponds to its Workshop page URL-ID, as seen in the screenshot:

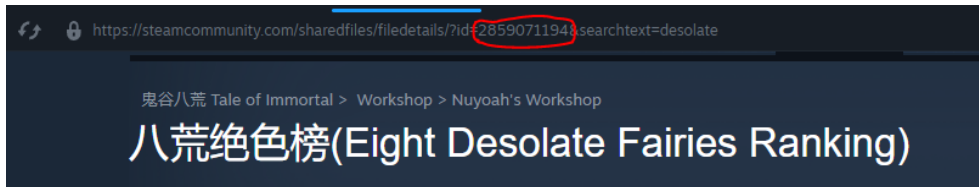


Figure 2: Finding a Steam Workshop mod's folder name on the Steam Workshop page.

If the mod is open source, then inside the number-folder you'll find a "debug" folder from which the game executes the mod itself, and a "ModProject" folder which contains the project's source(s).

1.2 Local Imported mod-folder

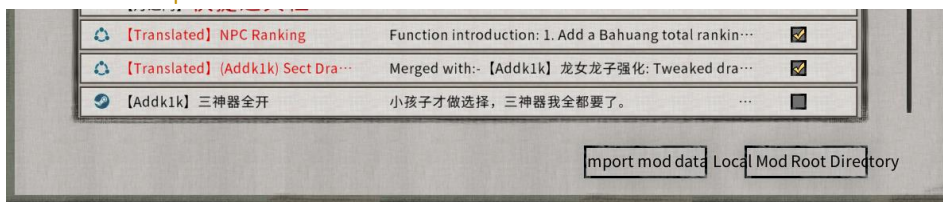


Figure 3: The two buttons do exactly what the English text says they do.

The "Import mod data" button opens up a Windows folder selection dialogue, which lets you select any mod folder that isn't in the game's defined local mods folder. Under the hood, it basically copy/pastes the folder (including all of its content) into:

- Windows: <Game installation path>\ModExportData\
- Linux: <Game installation path>/ModExportData/

The <Game installation path> is indeed the folder within your computer that contains all of your game files, for example: “C:\Program Files (x86)\Steam\steamapps\common\鬼谷八荒\”

The 鬼谷八荒 is what the game’s folder (by default) is named as.

(Windows users don’t need to worry about this.) Linux users might have to rename the folder with English characters unless they’ve configured all of their relevant game-emulation tools to support whatever-encoding-Chinese-symbols-may-use.

1.3 Mod creation project folder

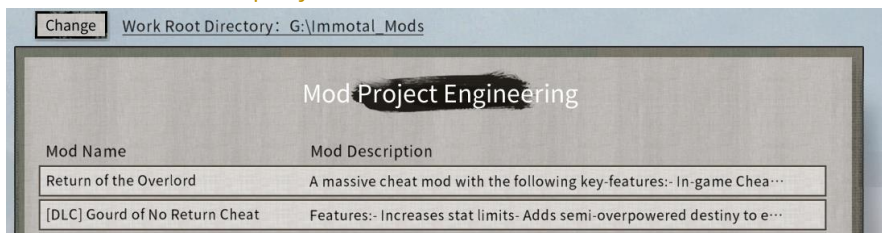


Figure 4: The mod project interface’s appearance for those that have never even clicked it before.

As is seen from the screenshot, there’s a big “change” button at the top which lets you choose a directory of your liking. After that, pressing “New Project” at the bottom of the same interface (*not shown in the above-screenshot*), a new folder will be generated within your chosen Work Root Directory.

By default upon “New project” generation, the game will generate a random NID (Namespace ID) for your project. Based on this, the project’s folder will be named “ModProject_NID” by default, see:

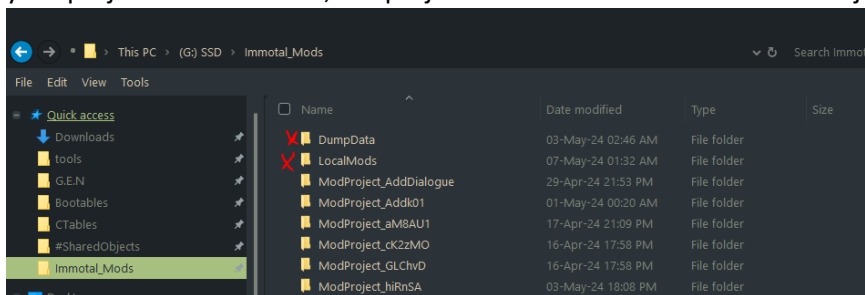


Figure 5: Example on how such a Work Root Directory would look like. The top-2 folders with red crosses are NOT project folders, but instead just folders that could contain totally non-Tale of Immortal related files.

The project folders inside the Work Root Directory can be renamed into anything and shouldn’t break the containing mod from working. **Within each project folder** you’ll find the following items:

- “ModProject” – A must-have folder, for the parent-folder to be classified project directory by the game. *This folder contains all of the source files from the respective mod project.*
- “debug” – Optional folder which’ll only be generated by the game, if you’ve ever entered Debug Mode within the respective mod project.

Other than the above mentioned 1 or 2 folder(s), you’re allowed to store folder with any other name and/or any type of files within the parent-folder “ModProject_NID”. And if you do, then uploading your

mod to the Steam Workshop will include every and any content inside “ModProject_**NID**”. (This could let you include extra stuff that you’d like to share, such as guide/tutorial documents, etc..)

1.4 Local in-game achievements & Main/DLC-game saves directory

In Windows, they’re stored at:

1. <User folder>\AppData\LocalLow\guigugame\guigubahuang\Steam

In Linux I don’t know where it’s located

Beyond this single piece of information, I don’t really know a lot more about the underlying folder structure and naming conventions yet, because...

In-game on the main menu’s righthand-side you’ll find the buttons “Export Saves” and “Import Saves”, which handles what their text describes they’d do. So you don’t always necessarily have to navigate to the storage directory itself.

2. How to local import mods

In [section 1.2](#) we’ve seen a screenshot of how the “local import” button looks like. Now we’ll learn what kind of folder you can import to the in-game mod list:

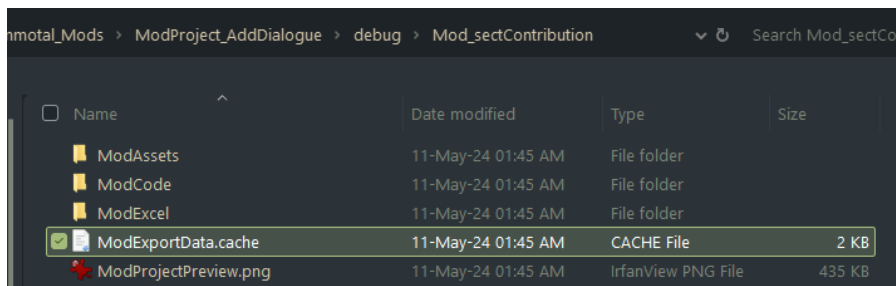


Figure 6: The file ‘ModExportData.cache’ indicates that its parent-folder is a mod-folder.

As is seen in the above-screenshot, the selected/highlighted file name indicates that the respective folder (*in this case* “**Mod_sectContribution**”) is a mod folder that you can select to import. It is also legitimate to import the “**debug**” folder which is one layer above “**Mod_sectContribution**”, it will work just as fine.

On a successful import you’ll always see the following notice, otherwise importing has failed!



Very important!

At this point you already know how to distinguish import’able mod folders from non-import’able. However unwilling you ‘may’ be to read how to create import’able mods from source, it might just be what makes or breaks your gaming experience! How/why? Find out by continuing to read...

3 Turning mod source into a playable mod

In [section 1.3](#) we've learnt about "choosing your own Work Root Directory" as well as how the mod project folders are structured. Now we'll learn how to create import'able mod folders from source folders, and (*most importantly*) why *not knowing* this information might break your gaming experience.

3.1 Outdated 'debug/mod' folders

Since you've got your hands on this guide, chances are that you've discovered my Google Drive folder: <https://drive.google.com/drive/folders/1PKv34AG5c5GXGXHe8mwuG92oLmW8-jr4>

From there you can download many mods that I'll probably never release on Steam, as .zip file that contains mod project folders. Some of these mods 'may' also contain a "debug" folder, which you'd probably/lazily want to import directly.

These "debug" folders of mine could however contain outdated (or broken) versions of the mod!

(From here you can either skip to section 3.2, or continue to read how/why the above-written thing could be a possibility.)

The reason why such a thing is possible, is because the "debug" folder is generated when you enter Debug Mode within a respective mod project. (Yes) If you've never used Debug Mode while working on a mod, the debug folder will not exist.

The debug Folder is essentially "just a local export", e.g. the conversion from "non-import'able" mod source folder into "import'able" mod folder, which Debug Mode conveniently loads/executes with.

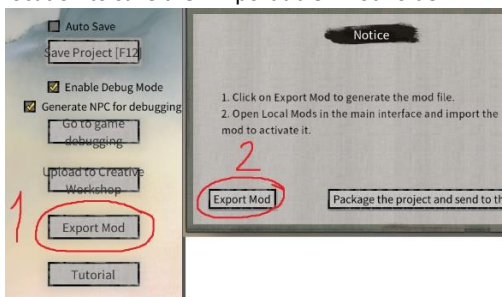
That said: *If* I had 'remembered' to enter Debug Mode on a Mod right before I zip-compress & upload it on Google Drive, the result will be a convenient "debug" folder which everyone could lazily import.

However if I forgot to do the above-written thing, and there happens to be **an old-old** debug folder present inside the mod project: Then anyone that's lazy, is essentially importing an incomplete/broken version of the mod, without even knowing it!

3.2 How to local-export an import'able mod folder

This following instruction works for **any** mod project source folder:

1. Create a new folder inside your Work Root Directory, see [section 1.3](#) if you had skipped reading it earlier and don't know what I'm talking about right now, then place the folder "ModProject" inside the newly created folder.
2. Open the "Create Mod" interface and you'll already see the mod project there. Select it and "Edit"
3. Via classic English reading comprehension, press the buttons as shown in the following screenshot, then select a location to save the "import'able" mod folder.



4. From the local import dialogue interface you can select the folder that you just exported, and that's it!

4. Problems importing a mod

There are several issues/imperfections related to the in-game button shown in [section 1.2](#), which we must look at:

4.1 DLLs cannot be overwritten while already-loaded!

When a DLL-mod's been loaded (e.g. checked in your mod list), you cannot use the "Local Import" button to replace it. This'll result in the "Mod imported!" notification (shown in [section 2](#)) not appearing on your screen.

There are two ways to fix this issue:

1. Uncheck the relevant mod on your mod-list, restart the game and it'll work.
2. Shutdown the game and overwrite the respective mod-folder manually, see [section 1.2](#) to find the directory.

4.2 The in-game import button DOES NOT delete files!!

This behavior is most likely to cause unexpected behaviors with mods that rely on **ModExcel** e.g. json files to function. Assume the following example:

1. You're experimenting with data-files, and as things go wrong you naturally delete the "experimental" json/xlsx files from your Mod Project folder that causes your issues.
2. You then create a new Local Export which you then Local Import to run; **Or** you directly (re-)enter Debug Mode, after having previously experienced the issue in Debug Mode.

The result is: Your game will continue to run into the same behavior that caused you to delete those "experimental" json/xlsx files, because the game does not delete files from mod folders nor a mod project's debug folder!

How to fix this issue:

- Manually delete the entire respective mod-folder from "<Game installation path>\ModExportData\" or press remove from within the game's Local Mods interface, before installing the latest version.
- Manually delete the entire "debug" folder from your ModProject, before going into your next Debug Mode session.

Neither this issue nor the 4.1 issue should arise with Steam Workshop-downloaded mods. Because in that case, it is your Steam Client that handles the file management (and not the game.)

5. Advanced info

If you don't plan on creating mods, then you can skip reading this section.

5.1 VS project source shows dependency errors

This happens because your Tale of Immortal has been installed in a different directory than the mod's creator has. The "ModCode" folder is auto-generated by the in-game Mod Creator when you press the "Write code" button. Within this auto-generation process, the folders...

3. "<Game installation path>\MelonLoader"
4. "<Game installation path>\MelonLoader\Managed"

...Are targeted as dependencies of the Visual Studio project.

Hence as example: If your game's installed on your C:\ drive while the mod creator's game has been installed in G:\ instead: The project indeed fails to capture the dependencies on your machine.

To fix the issue:

1. Open "ModMain.csproj" in your favorite plain-text editor.
2. Ctrl + F to search/highlight the mod creator's game path, for example mine is 'G:\Steam\steamapps\common\鬼谷八荒\'
3. Simply "Replace All" with the path towards your own game's installation directory and save the file, that's it!!

Now the Visual Studio project should capture all the DLLs fine.

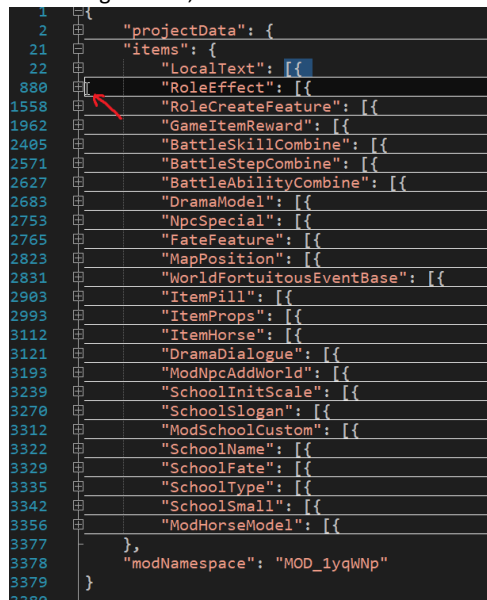
5.2 How to reverse engineer // source-reconstruct other mods?

You'll need a few tools:

- **Mod decryption tool** (read the Steam page desc for the instructions:)
<https://steamcommunity.com/sharedfiles/filedetails/?id=3225372871>
- **iISpy** (has less features compared to dnSpy, but is overall better at interpreting type casting:)
<https://github.com/icsharpcode/iISpy/releases>
- **dnSpyEx** (could directly decompile/edit source code but not recommended, better copy/paste it into a new VS project:)
<https://github.com/dnSpyEx/dnSpy/releases>

Pure-Mod Creator mods are a pain to reconstruct back into “ModData.cache” and “ModProject.cache”.
You could however...

1. (of course first) Use my Mod decryption tool against “ModExportData.cache” to decrypt it.
2. Open “ModExportData.cache” in your favorite text editor, Json-format the text and fold all tags. I recommend Notepad++ with JSTool plugin over VSCode for this task, because default-vs-default it's folding works better with Json.
3. Unfold tag “Items”, which'll show a list of all Json file names (see screenshot:)



```

1  {
2    "projectData": {
21   "items": {
22     "LocalText": [{"
880   "RoleEffect": [{"
1558  "RoleCreateFeature": [{"
1962  "GameItemReward": [{"
2405  "BattleSkillCombine": [{"
2571  "BattleStepCombine": [{"
2627  "BattleAbilityCombine": [{"
2683  "DramaModel": [{"
2753  "NPCSpecial": [{"
2765  "FateFeature": [{"
2823  "MapPosition": [{"
2831  "WorldFortuitousEventBase": [{"
2903  "ItemPill": [{"
2993  "ItemProps": [{"
3112  "ItemHorse": [{"
3121  "DramaDialogue": [{"
3193  "ModNPCAddWorld": [{"
3239  "SchoolInitScale": [{"
3270  "SchoolSlogan": [{"
3312  "ModSchoolCustom": [{"
3322  "SchoolName": [{"
3329  "SchoolFate": [{"
3335  "SchoolType": [{"
3342  "SchoolSmall": [{"
3356  "ModHorseModel": [{"
3377  },
3378  "modNamespace": "MOD_1yqWNP"
3379  }
3380  }

```

4. Create the respective Json file for every Items-key and sect the text starting from [{" until the next line. See the red arrow in the example-screenshot that points to where my cursor is, to know where the selection ends. Copy paste the selection into the Json file.
5. That's it!!