ModTool 2.0 Documentation

Overview 1
Features 1
Limitations 1
Contact 1

Setting up ModTool 2
Importing ModTool 2
Settings 2
Shared Assets and Packages 2

Setting up Restrictions 3
Adding A Restriction 4
Message 5
Target Type 5
Restriction Mode 6
Restriction 6

Creating an exporter 7
Creating the Mod Exporter package 7

Finding Mods 8
Search Directories 8
Mods 8

Loading a Mod 9
Loading and Unloading 9
Loaded Mods 9

Creating a Mod 10
Mod Project 10
Exporting a Mod 10
Restrictions 11

Overview

ModTool makes it easy to add mod support to your game. It enables modders to use Unity to create scenes, prefabs and code and export them as mods for your game.

To make sure the modder's scripts or Assemblies will be compatible with the game, ModTool has a simple but powerful code validator that lets you configure any number of restrictions and requirements.

ModTool creates a custom unitypackage for every game, which includes everything to create mods.

Features

- Let modders use the Unity editor to create scenes, prefabs and code for your game
- Scripts and assemblies are fully supported
- Code validation
- Supports Windows, OS X, Linux and Android
- Mod conflict detection
- Automatic Mod discovery
- Asynchronous discovery and loading of mods.

Limitations

- ModTool relies on AssetBundles, which means there could be some issues if mods are created with the wrong Unity version. The exporter will check if the same version is used and inform the user if that's not the case.
- Unity can't deserialize fields of [Serializable] types that have been loaded at runtime. This means that a Mod can't use fields of its own serializable Types in the inspector.
- Mods have to rely on the game's project settings. E.g. Mods can not define their own new tags, layers and input axes. The created Mod exporter will include the game's project settings.

Contact

If you have any questions, suggestions, feedback or comments, please do one of the following:

- Send me an email: tim@hellomeow.net
- Make a post in the ModTool thread on the Unity forums
- Submit a new issue on GitHub

Setting up ModTool

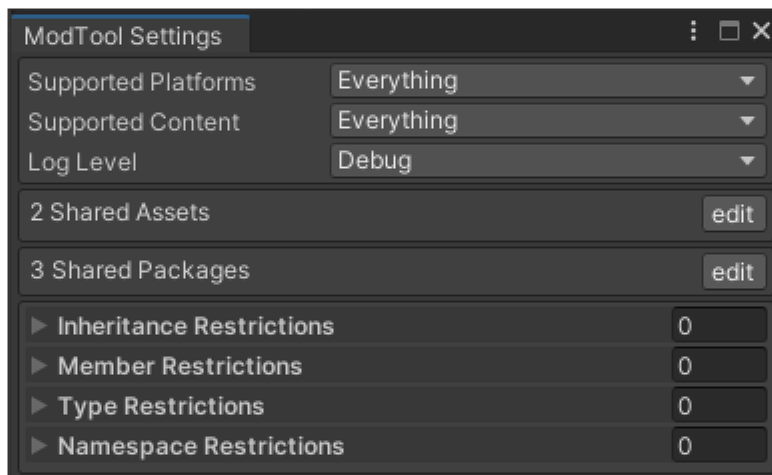
ModTool consists of two parts; the part that can export mods and the part that can find and load mods. The exporter is created automatically, on a per game basis. ModTool creates a Unity package containing everything that's needed to create mods for your game.

Importing ModTool

To begin using ModTool, just import the Unity Package. After that, you can start configuring it and generate a Mod exporter for your game. Use ModManager to find and eventually load mods. See the included example for a quick overview of the package.

Settings

In the settings window you can configure the things ModTool needs to know about the game.

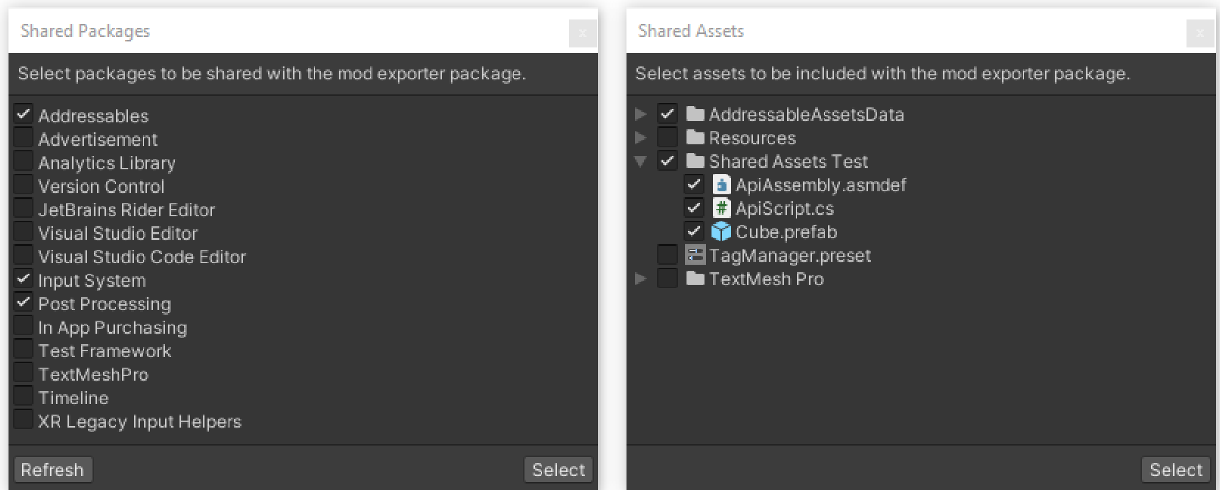


Here you can set up restrictions (see Setting up Restrictions), the game's shared assets you want to include, supported platforms and the types of content you allow to be included.

Shared Assets and Packages

With the Shared Assets and Shared Packages windows you can configure which assets and packages will be included with the mod exporter package. Here you can select scripts, prefabs and other assets that will be included.

When including scripts, it is recommended to use Assembly Definitions. By grouping shared scripts with an Assembly Definition, both the game and mod will reference the same code.



By including the Addressables package and the AddressableAssetsData folder, you can let mods use addressable assets from your game. When sharing addressable assets, mods can use prefabs and other assets from your game via their address, without creating duplicates. When you include regular, non-addressable assets, they will be included in mods as duplicates.

Setting up Restrictions

With Restrictions you can control the use of namespaces, Types, a Type's members, and inheritance in a Mod.

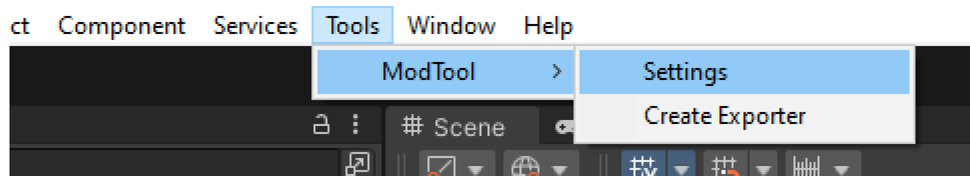
Restrictions can be applied to all Types, or only Types that derive from a specific Type. Restrictions can either require or prohibit something.

Code will be validated before exporting a Mod and before a Mod is loaded into the game, to make sure mods will be compatible with the game and don't cause issues.

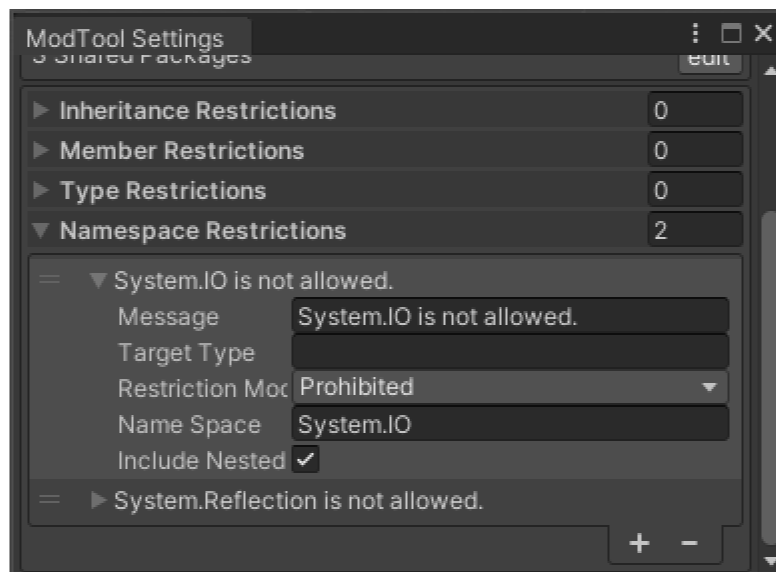
ModTool has a few default restrictions, which restrict the use of System.Reflection and System.IO. The default restrictions can be removed in the settings window if needed. It might be a good idea to keep these in place, unless you want mods to be able to access the user's file system or circumvent restrictions with reflection.

Adding A Restriction

To add a restriction, open ModTool's setting window by going to the ModTool menu.

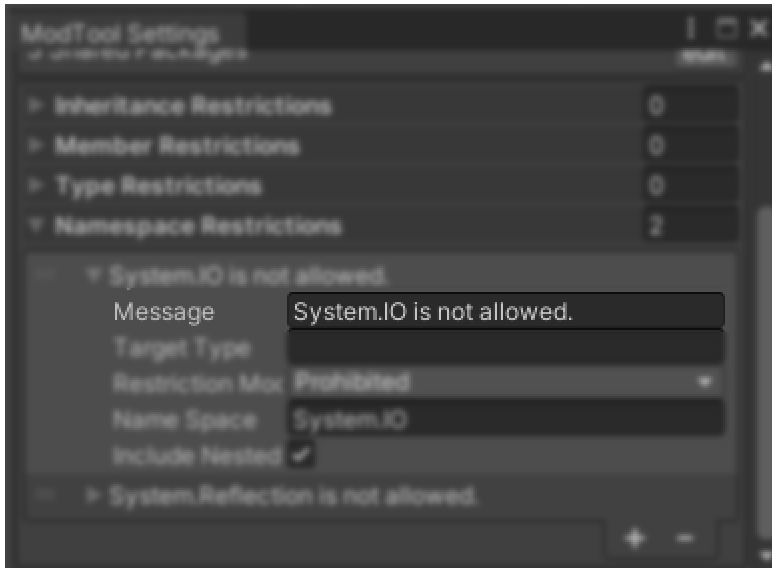


Here you can see various settings, including restrictions. Adding a Restriction works similarly to how you'd add a new axis in Unity's input manager. You can add a new Restriction by increasing the size of the collection.



Message

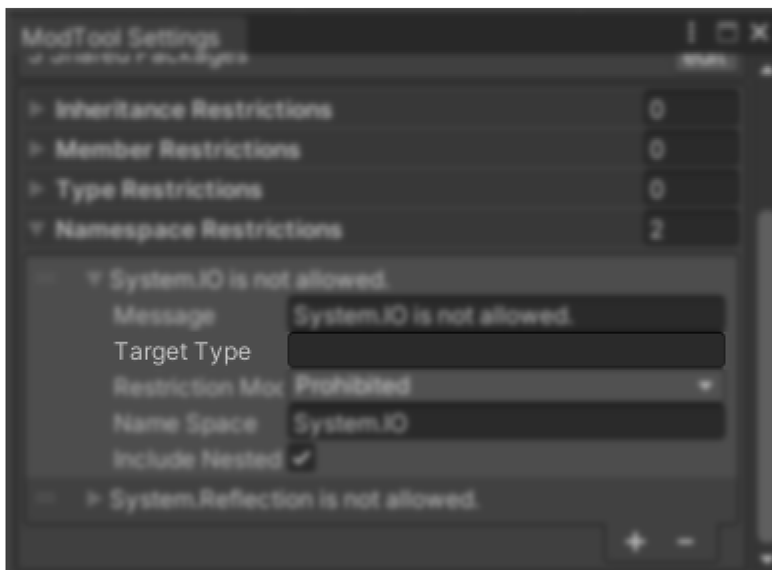
A Restriction's message is what will be displayed or logged when the loading or exporting of a Mod fails due to this Restriction.



Target Type

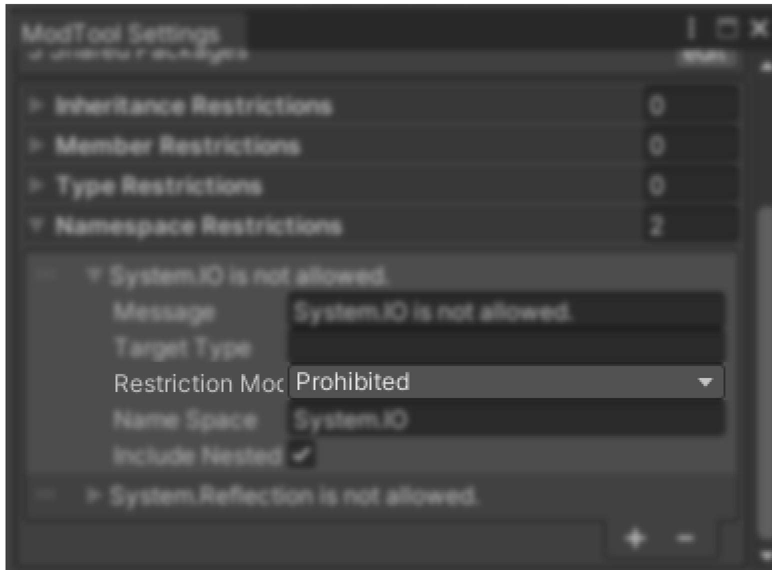
The Target Type of a restriction is the base type to which the Restriction will be applied. For example, if you want a restriction to only affect Types that derive from MonoBehaviour, you can configure it here.

When left empty, the Restriction will be applied to all Types.



Restriction Mode

Use the Restriction's RestrictionMode to determine whether the Restriction prohibits or requires the use of something.

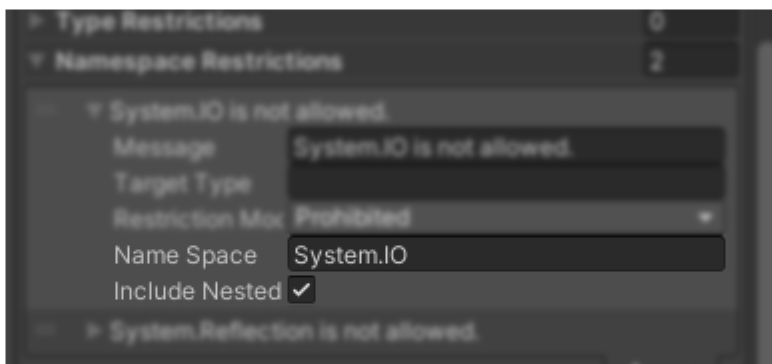


Restriction

For each Restriction you need to provide which member, Type, namespace or base class you want to prohibit or require. The restriction will look for the name including the type name and namespace where applicable.

When you want to restrict the use of a property, you need to provide the name of the generated getter and/or setter methods. Usually these are "get_" or "set_" followed by the property name. Constructors can be restricted by using ".ctor". This name includes the dot, for example: "UnityEngine.GameObject..ctor".

This restriction will prohibit the use of the System.IO namespace for all Types.



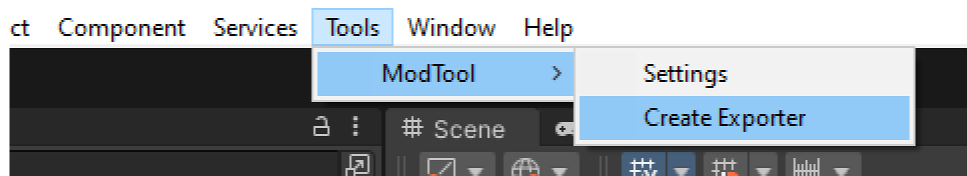
Creating an exporter

The Mod Exporter is the package modders will use to export their assets and code, so they can be used as Mods in your game.

The package includes everything that's needed to create mods for your game. These are the ModTool exporter dll and the shared assets and package names that have been configured in ModTool's settings.

Creating the Mod Exporter package

The package will be automatically generated and included in the game's directory when you build the game in Unity. It can also be created at any time by using the ModTool menu.



The package will be named **"<productName> Mod Tools.unitypackage"**.

ModTool relies on AssetBundles, which means mods that are exported with a different version of Unity can have some issues. The exporter will check if the correct version is used and inform the user if that's not the case.

Finding Mods

ModTool will look for mods in certain folders. It does this by monitoring one or more directories for any added, changed or removed mods.

ModTool's default search directory is the "Mods" folder inside the game's install directory. On Android, this is based on Application.persistentDataPath. It's possible to add or remove search directories.

ModTool keeps a collection of available Mods and provides events for whenever a Mod is added, changed or removed.

Search Directories

To add a search directory, use ModManager.AddSearchDirectory(String)

To remove a search directory, use ModManager.RemoveSearchDirectory(String)

The search directories can be refreshed with ModManager.Refresh(). This will look for any added, removed or changed mods.

When a new Mod is found, or when a Mod has been removed, ModTool will update the collection of available Mods.

Mods

ModManager.mods contains all Mods that are currently available. The ModManager. ModsChanged event will be triggered when this collection changes.

ModManager. ModFound will occur when a new Mod has been found. ModManager. ModRemoved will occur when an existing mod has been removed.

Mod.modInfo contains all of a Mod's info, like the type of content that is included in the Mod, the Mod's name and the Mod's version.

A Mod will become invalid when a change is detected in its folder. For example, when a Mod's files are removed. It will be replaced with a new, up to date Mod. Make sure that you don't try to use old Mod instances that are marked invalid. Invalid Mods won't load.

Loading a Mod

Available Mods can easily be loaded and unloaded. Loading a mod loads the Mod's Assemblies and assets so they can be used in the game.

Loading Mods is done asynchronously. Both the ModManager and Mods provide events for when mods have completed loading or unloading.

Loading and Unloading

Load a mod with Mod.Load(). When a Mod has finished loading, Mod.Loaded and ModManager.ModLoaded will occur.

To unload a mod, use Mod.Unload(). Because Mods can be loaded asynchronously, a Mod could still be loading when Unload is called. You can keep track of a Mod's load progress with Mod.progress and Mod.loadState. When a Mod has unloaded, Mod.Unloaded and ModManager.ModUnloaded will occur.

Mod.isEnabled can be used to keep track of which mods a user wants to enable or disable. This property does not affect what you can do with a Mod; a Mod that is not enabled can still be loaded. This property is saved in the Mod's modInfo. With Mod.ConflictingModsEnabled() you can check if any conflicting mods are enabled as well. A Mod can not be loaded when another conflicting Mod is loaded. Two Mods are in conflict when they share scenes or assemblies that have the same name. This should not happen very often, because the mod exporter gives these assets a name based on the Mod's name.

Any issues related to initializing or loading a Mod, like missing files or failed code validation are logged to the console / log file, and can also be accessed with Mod.errors.

Loaded Mods

Once a Mod has been loaded, you can use its resources. These are the prefabs, scenes and Types in any of the Mod's Assemblies.

Scenes are wrapped in the ModScene class, to make them easy to handle and load in a similar way as Mods.

Included prefabs are accessible through Mod.prefabs. If you wish to destroy any prefab instances when a Mod is unloaded, you can instantiate them with the ObjectManager. The ObjectManager keeps track of which objects belong to a certain Mod, so they can be cleaned

up properly if the mod is unloaded. This is also how ModTool keeps track of a mod's instantiated objects internally.

You can find non-MonoBehaviour implementations of interfaces with Mod.GetInstances(). This will find and create instances of types that implement a given interface or class.

To find MonoBehaviour implementations, you can use Mod.GetComponentsInScenes() and Mod.GetComponentsInPrefabs().

When a Mod is unloaded while any of its scenes are still loaded, the scenes will be unloaded. Instances of prefabs and components that are still in use will be destroyed.

Creating a Mod

Mods are created in Unity. You can create scenes, prefabs and scripts just like you would in any other Unity project.

Once you have finished creating the assets and scripts, you can export the project as a Mod. After that, they can be used in the game.

Mod Project

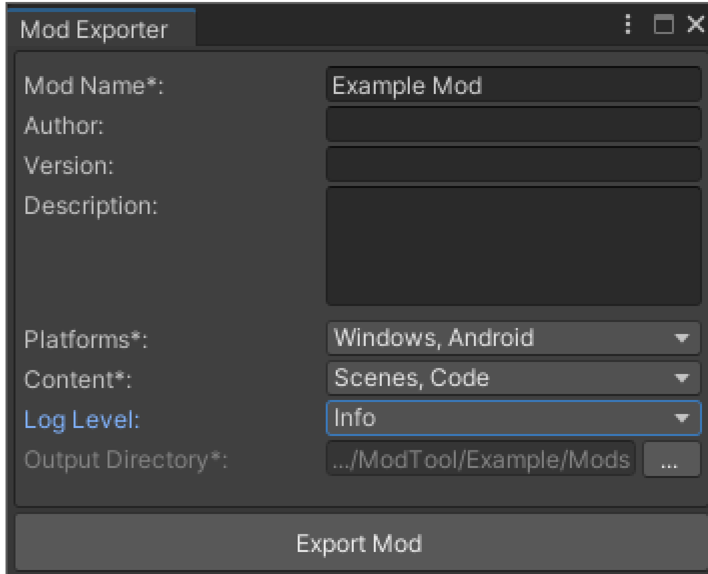
To begin creating a Mod, create a new Unity project. It is recommended to have a separate project for each Mod.

After creating a new project, you can import the Mod Exporter package into Unity to begin creating Mods.

Exporting a Mod

To export the project as a mod, go to the Tools menu and select ModTool > Export Mod.

Here you can give the Mod a name and other information. You can select which platforms the Mod will support and whether to include scenes and prefabs.



ModTool relies on AssetBundles, which means mods that are exported with a different version of Unity can have some issues. The exporter will check if the correct version is used and inform the user if that's not the case.

Restrictions

The game might have some restrictions imposed on what you can do with scripting. This is to make sure it will be compatible with the game and to prevent the Mod from breaking things as much as possible.

The mod's scripts will be verified before it's exported. You can also choose to verify via the ModTool menu at any time to see if your mod is compatible with the game. Any issues during validation will be logged to the console or the game's log file.