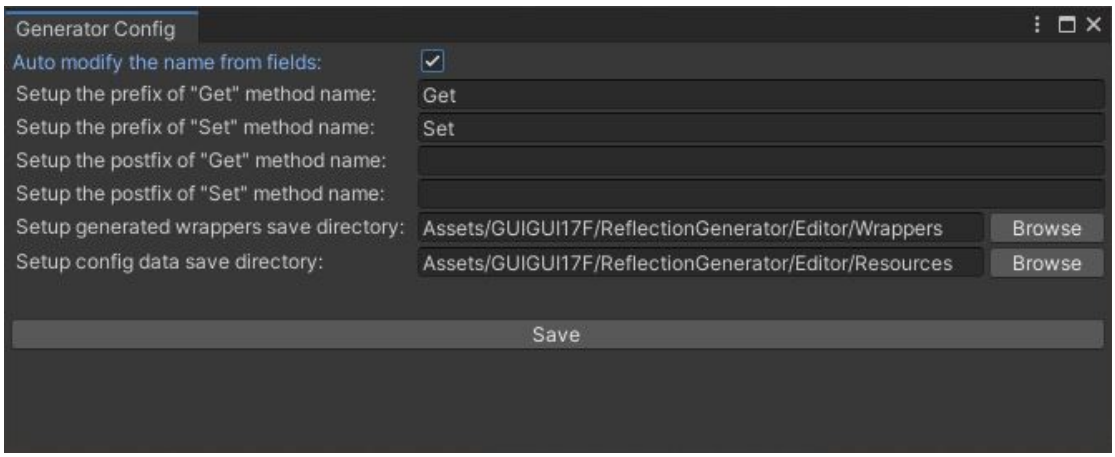
Reflection Generator User Guide

v1.0.0

The Reflection Generator is a plugin used to generate reflection wrappers for classes, thus you can use the wrappers to read or change non-public field values in these classes. This simple guide will show you how to use it in 5 minutes.

I. Setup Config

After importing Reflection Generator into the project, the first thing we should do is to configure the plugin. You can open the config window by “Tools -> ReflectionGenerator -> GeneratorConfig”.



Most options are used to modify the generated “Get” and “Set” method names in reflection wrappers.

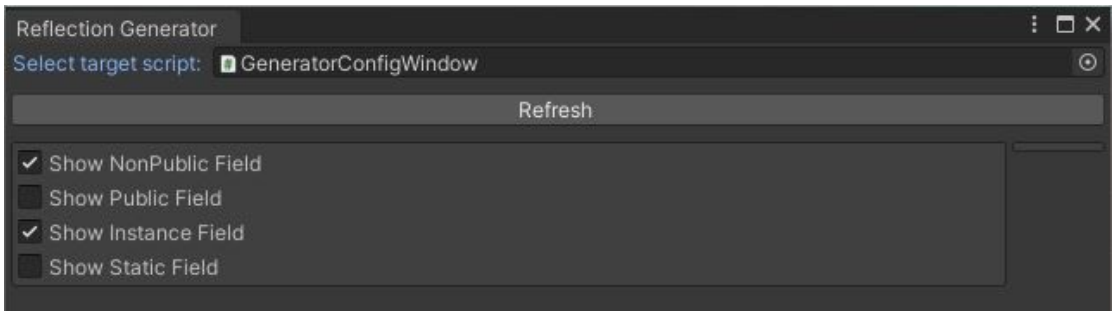
For example, a field named “Color” will get a “Get” wrap method named “[GetPrefix]Color[GetPostfix]” and a “Set” wrap method named “[SetPrefix]Color[SetPostfix]”, with the [GetPrefix], [GetPostfix], [SetPrefix], [SetPostifx] configured in this window.

The “auto modify” option means whether we should modify the origin field name to make our generated method name more normalization. For example, when the “auto modify” turn off, a field named “_shape” will get two wrap methods named “[GetPrefix]_shape[GetPostfix]” and “[SetPrefix]_shape[SetPostfix]”. But when it turned on, the generated method names will became “[GetPrefix]Shape[GetPostfix]” and “[SetPrefix]Shape[SetPostfix]”.

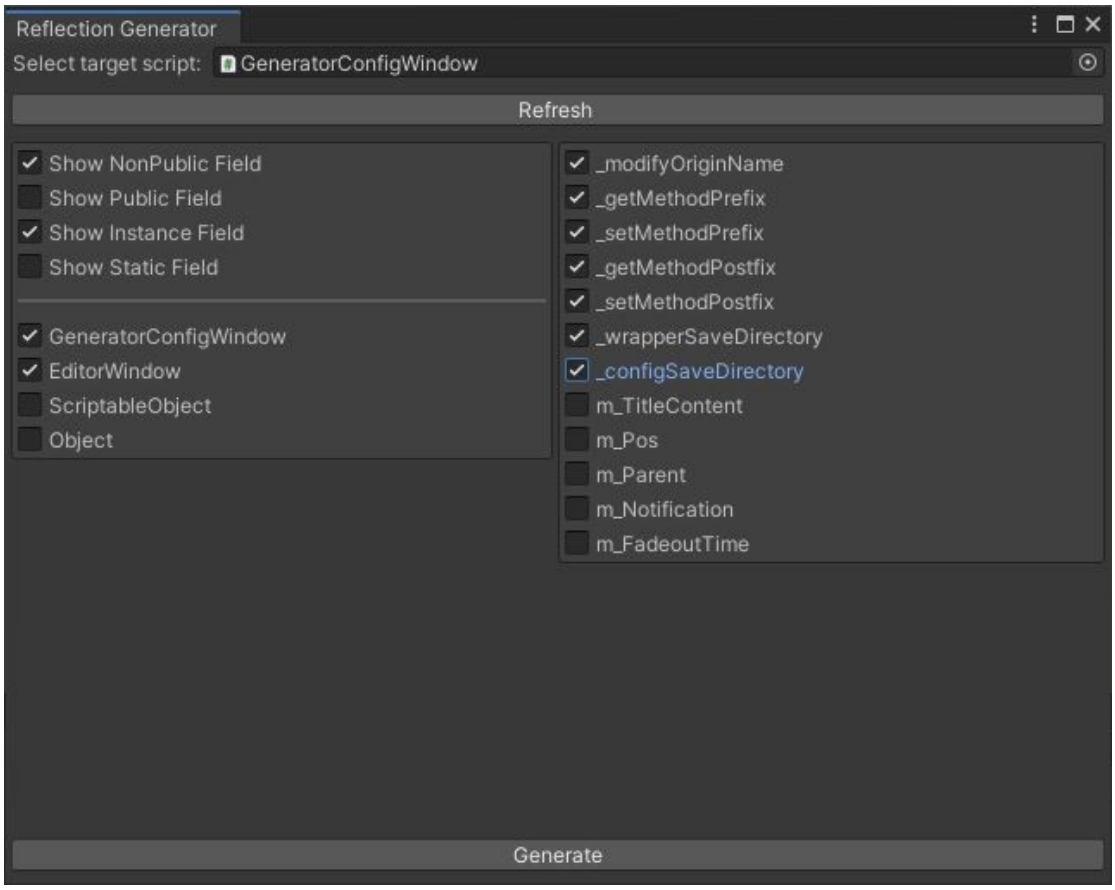
The last two options are used to configure where to save the plugin configuration data and the generated class wrappers. If you changed the plugin folder path, it’s recommended to update these config again.

II. Generate Reflection Wrappers

The plugin main window is in “Tools -> ReflectionGenerator -> GeneratorWindow”.



Step 1, drag or select your target class script into the “target script” field and click “refresh”.



Step 2, after refreshing the window, you can see a base type list has shown below and a field list has shown on the right. The type list is from the inheritance hierarchy of current target, and the field list is based on your filter and type selections.

Use the filters (NonPublic, Public, Instance, Static) to choose what kind of fields you are interested, and the type options to choose in which types the fields declared should be shown.

Step 3, select the fields you want the wrapper contains, and click “generate”. Now, the target class wrapper will generated into the “wrappers save directory” and ready to be used.

III. Use Wrappers

The wrapper's name will be "[OriginType]Extensions", but in fact you needn't to remember it. Every class wrapper will use the "Extension Methods" technology and the namespace same as the origin type, so you can just call the generated wrap methods by a simple ".".

For example, if you configure the "GetPrefix" as "Get", "SetPrefix" as "Set", keep "auto modify" on, and left the postfixes empty, a field value named "_speed" in a instance named "ball" can be read by "ball.GetSpeed()" and changed by "ball.SetSpeed(value)".

IV. Notices

- As you know, the reflection technology in .NET framework is slow for game environment, and may cause incompatibility issues in some platforms, so it's strongly recommended to use the generated warpers only in Editor tools.
- After moving the plugin folder, you may want to update the plugin configuration to make sure everything works properly.
- As you know, the "Extension Methods" can only be used by class instance. To access methods for static fields in a wrapper, you can just create a new class instance. Don't worry, the instance value will be ignored when handling static fields.

- You may find many Unity build-in types won’t show any field even you checked it and turned every filter on, that’s because most thing you used such as “transform” or “color” are just properties which wrapping an external method. It’s always recommended to change the build-in type values by using the engine public APIs.