Obfuscator (2.5.*)

Technical Documentation

Table of Contents

Best Practices

Buttons

For stronger obfuscation, consider assigning button clicks programmatically:

```
using UnityEngine;
using UnityEngine.UI;
using Beebyte.Obfuscator;
public class Buds : MonoBehaviour
{
    public Button Button;
    public void Start()
    {
        Button.onClick.AddListener(CodedClick);
    }

    //Assigned in the Start() method
    private void CodedClick() //This gets obfuscated
    {
        Debug.Log("Coded Click");
    }

    // Assigned through the inspector within On Click () +
    // so requires [SkipRename] when obfuscating public mono methods
    [SkipRename]
    public void InspectorClick() //Visible
    {
        Debug.Log("Inspector click");
    }
}
```

In this example 'CodedClick' will be obfuscated.

Methods

More methods result in better obfusation. Following good software practices such as S.O.L.I.D. will not only improve maintainability of your code, but will be tougher to reverse engineer.

Check your protection

If your build target contains DLLs then consider using an assembly inspector to verify you're happy with the level of obfuscation and/or tweak Obfuscator options to improve it.

There are many different assembly inspectors available. Examples include DotPeek (by JetBrains & free) and ILSpy (open-source).

If you're targetting multiple build targets in a cloud environment or have a customised build process then this check is strongly recommended.

IL2CPP

- By default it is considerably harder for someone to read code built to IL2CPP and so is highly recommended.
- The Obfuscator is still beneficial with IL2CPP enabled, but to a lesser degree than Mono targets.
- A simple way to see effects of obfuscation is to view the global-metadata.dat file in a text editor and search for the names of your methods.
- There are no additional steps required to instruct the Obfuscator to obfuscate IL2CPP builds.

Configuration

Assemblies

Obfuscate all assembly definitions (2017.3 onwards)

- Significantly strengthens obfuscation when enabled.
- This is an alternative to specifying each assembly definition class in the 'assemblies' list.

Assemblies

- A list of assemblies to be obfuscated that are first created by the Unity build process.
- The file extension must be included.
- e.g.:
  - Assembly-CSharp.dll
  - MyAssemblyDefinitionName.dll

Compiled Assemblies

- A list of assemblies to be obfuscated that have been pre-compiled before executing a build.
- The file extension must be included.
- e.g. if you compile a DLL called Wheel through your IDE and place it in the Assets folder then this list would contain Wheel.dll

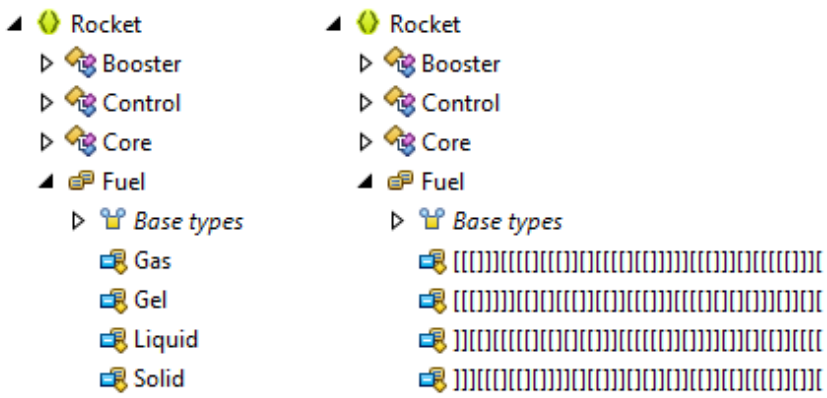
Extra Assembly Directories

- If you have an AssemblyResolutionException, find the referenced DLL on your machine and add the directory of the DLL to this list. The Obfuscator will then check this directory when trying to locate the DLL as part of the build process.

Rename

- Significantly strengthens obfuscation when enabled.
- Given only the obfuscated DLL, it is near impossible to reverse engineer the original names.
- Given the obfuscated DLL and the 'Random seed' (Naming Policies) a hacker would need to guess names, hash them, then search the code to see if those hashes are used anywhere. It is near impossible to return the original name without this guessing step.
- Enabling a member type (classes, methods etc) will always obfuscate private instances of that type unless the Obfuscator detects it would cause a conflict.
- Enabling 'Protected' will obfuscate protected instances.
- Enabling 'Public' will obfuscate public instances.

Include enum constants



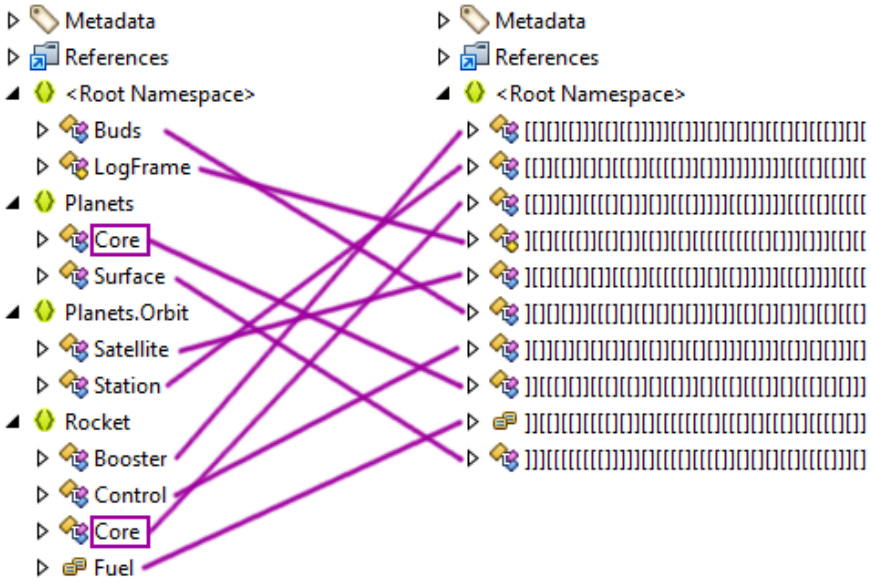
- Strengthens obfuscation when enabled.
- Be careful if you have code like the following:

```
public class Booster {
    private readonly Fuel _fuel;

    public Booster(Fuel fuel) {
        _fuel = fuel;
    }

    public void Ignite() {
        // Ignited ]][[][[[[[][][][][]]]][[[[[[]][]]]][]][][[]][[[[ booster!
        LogFrame.WriteMessage("Ignited " + _fuel + " booster!");
    }
}
```

Strip Namespaces



- Strengthens obfuscation when enabled by moving classes into the default namespace.
- 'Skip Namespaces' is always searched before 'Strip Namespaces' is applied.
- Having two classes with the same name in two different namespaces is allowed – the Obfuscator will assign unique names to avoid any conflict.

MonoBehaviours

Include public mono methods

- Strengthens obfuscation when enabled
- For public methods on MonoBehaviour objects to be obfuscated, this option must be enabled in addition to Rename->Methods->Public.
- Typically you want to enable this but be prepared to use [SkipRename] on methods that have been selected within the Unity Inspector, i.e. Button Clicks.

Include public mono fields

- Strengthens obfuscation when enabled
- Streamed assets require this to be disabled. Alternatively annotate streamed asset fields with [SkipRename]

Obfuscate MonoBehaviour Class Names

- Significantly strengthens obfuscation when enabled.
- When enabled, obfuscates classes deriving from MonoBehaviour.
- Increases build time.
- Streamed classes will not work with this option enabled unless you annotate them with [SkipRename].
- Precompiled DLLs containing MonoBehaviours will not be renamed with this feature. They will be treated as though annotated with [SkipRename]. Consider using Unity's Assembly Definition files instead and adding the assembly reference to temporaryDLLs instead of permanentDLLs in Config.cs.
- This is the only option that has to touch the original source files (renaming), but will restore them after the build.
- In the event of a failed build, the sources are restored.
- In the event of a catastrophic event where the Unity IDE closes, the sources are restored when the project is next opened within the Unity IDE.
- Make a backup of your project before enabling this option for the first time.
- A file called `_monoBehaviourTranslations` will be temporarily created in the root of the project during this build process, and removed on source file restoration.

Non-standard Source Paths

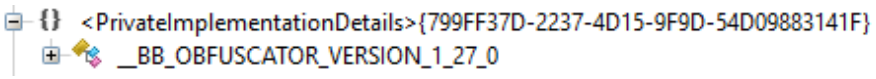
- Used to instruct the Obfuscator where to find certain MonoBehaviours if it gets confused.
- Leave this empty unless prompted by the Obfuscator when building.

Abstract MonoBehaviours

- Holds a list of abstract MonoBehaviours that can't be renamed.

Miscellaneous

Add Obfuscation version attribute



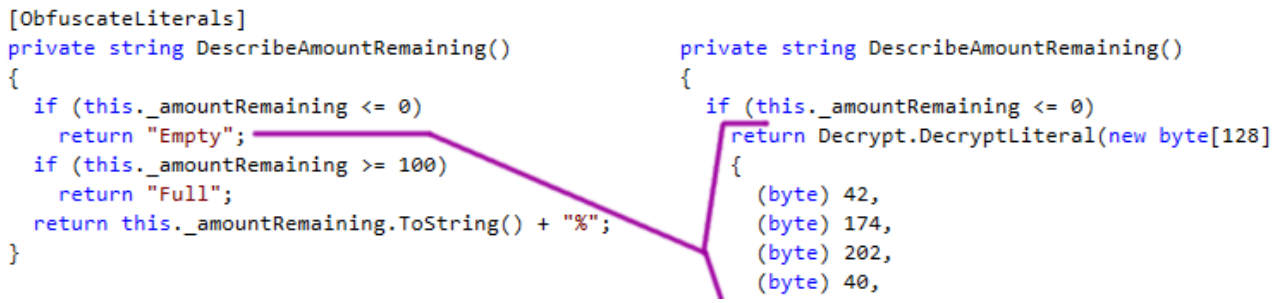
- Prevents a DLL from being obfuscated twice which would otherwise cause many things to stop working.
- If you have a custom build script that launches obfuscation, it's recommended to enable this. Otherwise it can be safely disabled.

Progress Bar Detail

- Shows the obfuscation status during a build.
- The "Detailed" setting has been known to increase the build time of large projects by around 40% but provides the most informative progress.

String obfuscation

Obfuscate String Literals



- Strengthens obfuscation when enabled and annotations are present in your code
- There are two ways to obfuscate literals
  - [ObfuscateLiterals] on a method (recommended):

```
[ObfuscateLiterals]
private string DescribeAmountRemaining() {
    if (_amountRemaining <= 0) {
        return "Empty";
    }
    if (_amountRemaining >= 100) {
        return "Full";
    }
    return _amountRemaining + "%";
}
```

  - Using a marker:

```
private string DescribeAmountRemaining() {
    if (_amountRemaining <= 0) {
        return "^Empty^";
    }
    if (_amountRemaining >= 100) {
        return "^Full^";
    }
    return _amountRemaining + "^%^";
}
```

- String literals obfuscation (two-way obfuscation) is not as strong as member renaming (one-way obfuscation).
- Given only the obfuscated DLL it is possible to reverse engineer the original string.
- The application needs to read the original string (two-way obfuscation) so a hacker could, in theory, apply the same technique to read it too. Original source class and method names are not required by the application and so a one-way obfuscation is applied to those.
- Caching is NOT applied to string obfuscation due to security considerations when memory is dumped.

Obfuscation Marker Unicode

- If using a marker to obfuscate, the marker can be changed using this value.
- The value is in decimal notation
- The default value is 94: ^

Use RSA

- Strengthens obfuscation when enabled.
- In theory this requires more CPU to run, but in the majority of cases it is imperceptible.
- When enabled, the length of the obfuscated byte arrays are the length of the string rounded to the nearest (keyLength / 8). This makes it hard to guess which string is being obfuscated based on its length alone.
- When disabled, the length of the obfuscated byte arrays are equal to the length of the string.
- When disabled, no cryptographic libraries are used for string obfuscation.

RSA Key Length

- Slightly strengthens obfuscation when using higher lengths.
- A higher value typically means more bytes are used, so it's less clear which strings are being obfuscated.

Obfuscate Literals in all Methods

- Strengthens obfuscation when enabled.
- Equivalent to annotating every method with [ObfuscateLiterals]

Only Obfuscate Literals in Obfuscated Methods

- Weakens obfuscation when enabled.
- When enabled literals that would normally be obfuscated will not be obfuscated if its parent method is excluded from obfuscation.

Strip Markers on Non-Obfuscated Literals

- If enabled then any string that starts and ends with the obfuscation marker unicode character will have that character removed from both ends.
- This is to allow disabling of string obfuscation without the need to change all occurancies of the obfuscation character within your source code.
- If you have never used string marker obfuscation and never intend to, you can safely disable this option.

Fake Code

Add fake code

- Strengthens obfuscation when enabled.
- Increases file size.
- Increases obfuscation build time.
- Does not change the code flow.
- Clones existing methods and subtly modifies the copy in ways to misdirect people.

Min false methods per class

- A class capable of having fake methods will attempt to have this minimum number of cloned methods.
- The recommendation is to stick with the default value.

Max false methods per class

- A lower value reduces file size and build time.
- A higher value can increase security, but benefits soon become negligable.
- A single class will not exceed this number of injected fake methods.
- The recommendation is to stick with the default value.

Max instructions for cloning

- Another way to limit the filesize and build time, useful if you have only a few large methods.
- The recommendation is to stick with the default value.

Naming Policies

Unicode start in decimal

- Strengthens obfuscation when changed to unusual characters.
- The value is in decimal notation.
- The default value is 65 'A'.
- Some special unicode values can't be input to avoid lots of things breaking.
- Be careful changing this if you also use the 'Obfuscate MonoBehaviour Class Name' feature, since part of that feature involves temporarily renaming files to names using these characters. Some older or uncommon operating systems might not like unusual characters, however many will accept them.

N, where Number of characters = (2^N)

- Strengthens obfuscation when lowered.
- Lowering the value increases the length of obfuscated names and reduces the number of different characters used.
- Seeing [[]]]]][[]]][[][]]][[]]]]]]]][]][[[[][]][]]][][[(character 91, N = 1) is visually much harder to recognise than AGTNEHWK (character 65, N = 4)

- The default value is currently 4, but may change to 1 in a future release.

Random Seed

- Randomly generated when the Obfuscator is installed.
- The value directly changes obfuscated names.
- A team of developers using the same seed building the same source code will generate DLLs that have the same obfuscated names.
- This seed is printed at the head of created 'nameTranslation.txt' files.
- If the seed is compromised, hackers will still struggle to find the original source names. They would first need to guess a name and look within the DLL to see if the hash of the guess is present.

Name Mapping History

Create name translation file

- Creates a file in the root of the project containing the seed used along with a mapping of newly obfuscated names to old.
- Required to translate stack traces reported by your clients.

Name Translation File

- The name of the name translation file!

Reverse arrow order per line

- If enabled, mappings are new -> old
- If disabled, mapping are old -> new
- This was a backwards compatibility feature added for customers who had already started to write automated stack trace parsing tools.

Name padding delimiter

- Only change this if you intend to create a tool that parses stack traces.
- It provides a way for an automated tool to recognise a string that should be translated.
- The default value is 0
- Changing this can cause issues with <Insert issue>

Show simplified hashes

<To be removed>

Translate fake methods

- If enabled the names of injected fake code is also shown in nameTranslation.txt.
- Since fake code would never normally be executed, it's common to leave this option disabled.

Reflection and RPC

Search for Unity reflection methods

- Prevents runtime errors when enabled.
- Unity reflection methods are methods such as StartCoroutine that take a string and look for a method in your code with the same name.
- This option may be removed in the future (with a default of enabled).

Obfuscate Unity reflection methods

```
public void LogStatus()
{
  LogFrame.WriteMessage("[Status] Control");
}

private void Start()
{
  ((Component) this).get_gameObject().SendMessage("LogStatus");
}
```

```
public void ][][]]][[[]][[]]][][[]]]]][[]]][[[[[]][]]]][[ ()
{
  LogFrame.[[[[[][[[[[[]][[[]]]]][[][[[[]][[[]]]]][][[]]]("[Status] Control");
}

private void Start()
{
  this.gameObject.SendMessage("][][]]][[[]][[]]][][[]]]]][[]]][[[[[]][]]]][[");
}
```

- Strengthens obfuscation when enabled.

Obfuscate and replace literals for RPC methods

- Strengthens obfuscation when enabled, but may restrict seed changes.
- When disabled, [RPC] annotated methods are not renamed.
- If enabled then changing the random seed would mean old clients won't be able to talk to newly built servers and visa-versa.

Alternate RPC Annotations

- Some assets provide their own annotations that act like [RPC]. This is a way to tell the Obfuscator to treat such annotations in the same way it handles [RPC].

Replace literals even on skipped classes

- It's recommended to leave this enabled to protect against runtime errors.
- If disabled and a class annotates a method called 'StartLevel' with [ReplaceLiteralsWithName], then a string literal of "StartLevel" would only be replaced with its obfuscated counterpart if the container class would normally be obfuscated too.
- The use cases for wanting this disabled this is very limited.

Replace Literals

- Methods declared here will change any string containing that method name with the obfuscated counterpart.
- This is used for assets that use reflection to look up methods they've asked you to create in MonoBehaviours that aren't known to the base MonoBehaviour class.
- This is equivalent to annotating each declared method with [ReplaceLiteralsWithName]

Deletion

Attributes to remove if obfuscated member

- Strengthens Obfuscation when used
- If your code looks like this:

  [Custom("Launches Drone")]
  private void SomeMethod() { ..

  Then adding "Custom" to this list means that it will obfuscate to something like:

  private void JEQQKAEFJJ() { ..

  i.e. the [Custom] attribute will have been deleted.
- "Custom" will still work even if the attribute class is called CustomAttribute (C# maps Custom to CustomAttribute and the Obfuscator handles this)
- If SomeMethod() is skipped and not obfuscated then the [Custom] attribute will NOT be removed.
- This was introduced for Unity's attributes that interact with the Unity Inspector Window or Menu Bar that took string parameters that described what the method or class did.
- Works for types, methods, parameters, fields, properties, and events.
- Note that Beebyte attributes are always removed.
- [System.Obfuscation.Reflection] attributes are also removed unless their StripAfterObfuscation property is set to True.

Preservation

Only Obfuscate Specified Namespaces

- When enabled, no namespaces are obfuscated apart from those declared in 'Obfuscate Namespaces'.
- It can be useful if you have a gigantic project and want to gradually introduce obfuscation.

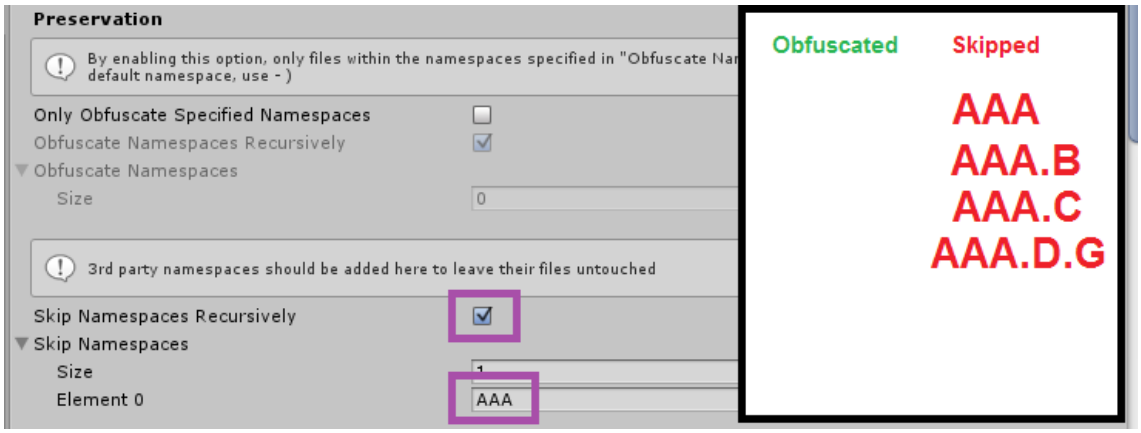
Obfuscate Namespaces Recursively

- When enabled, child namespaces are also obfuscated.

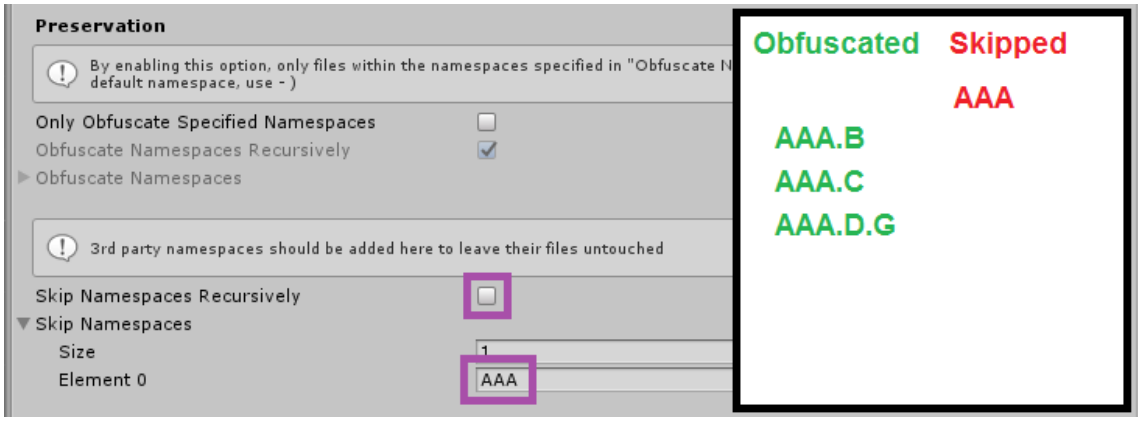
Obfuscate Namespaces

- The list of namespaces that will be obfuscated.
- If you only want to obfuscate the default namespace, use the hyphon/minus '-' character.

Skip Namespaces Recursively

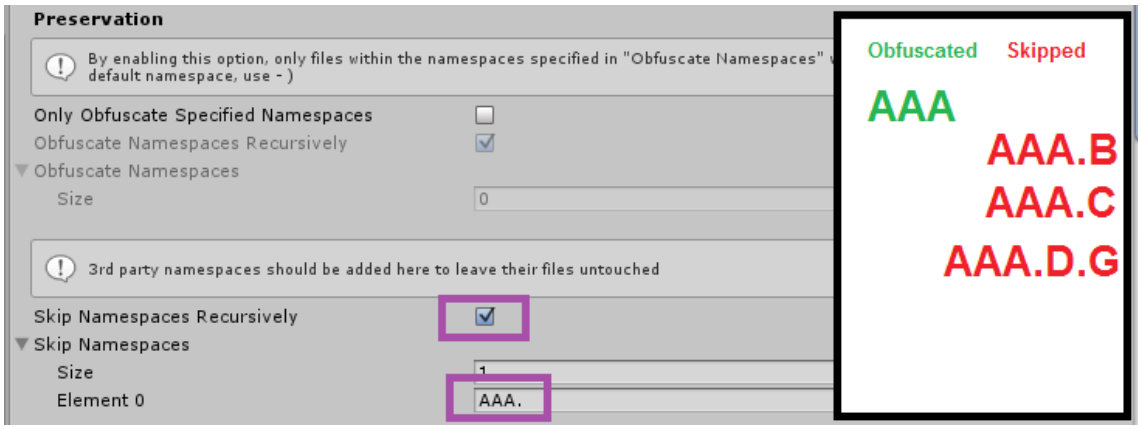


- When enabled, child namespaces are also skipped.



- When disabled, only exact namespaces are skipped

Skip Namespaces



- The list of namespaces that will not be obfuscated.
- If you want to exclude the default namespace, use the hyphon/minus '-' character.

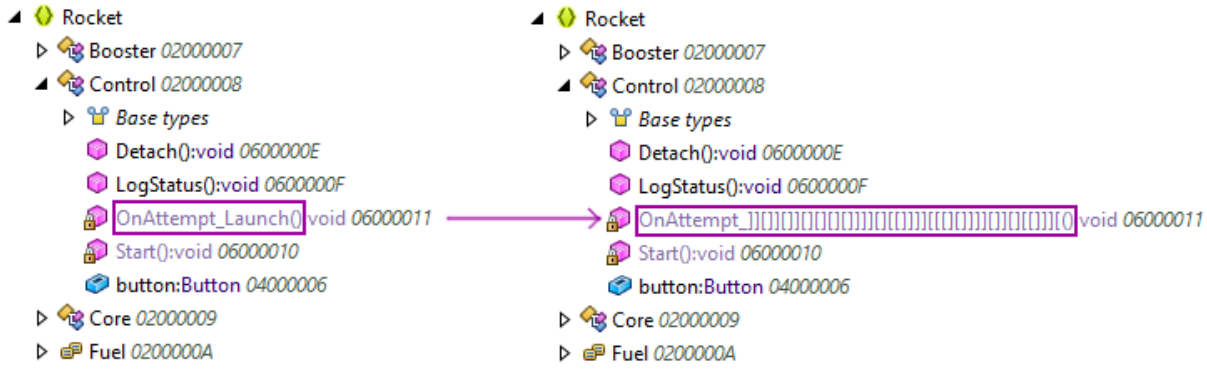
Skip Classes

- A list of classes that will not be obfuscated.
- Equivalent to annotating the class with [Skip].

Unity Methods

- Methods declared here that exist on a class derived from MonoBehaviour will not be obfuscated.
- Equivalent to annotating the declared methods with [SkipRename] if they are on a class derived from MonoBehaviour.
- Only event style methods that are found using reflection should be declared here. There is no need (but no harm) in declaring methods that are extended from the base class.

Preserve Prefixes



- Methods starting with the declared strings will use them as a mask when obfuscating.
- Useful for assets that expect methods to start with a particular string.

Alternative Attribute Names

- Any attribute declared in one of the lists will be treated as the Beebyte.Obfuscator attribute counterpart.
- e.g. Adding JsonProperty to the Skip Rename list will mean you no longer have to annotate each property with [SkipRename] if [JsonProperty] was already present.

Attributes

.NET Framework

[System.Reflection.Obfuscation]

- Equivalent to [Skip]

[System.Reflection.Obfuscation(ApplyToMembers=false)]

- Equivalent to [SkipRename]

[System.Reflection.Obfuscation(Exclude=false)]

- Instructs the Obfuscator to obfuscate the annotated target where it might not have otherwise done so.
- It is ignored if the Obfuscator knows renaming will definitely cause failures.

Beebyte.Obfuscator

[Skip]

- Instructs the Obfuscator not to obfuscate the target or any of its children.
- Usable on classes, methods, interfaces, structs, fields, parameters, events, enums, properties, and delegates.
- When used on a class, it's equivalent to declaring it in the "Skip Classes" list in options.

[SkipRename]

- Instructs the Obfuscator not to obfuscate the target's name, however children may be obfuscated.
- Usable on classes, methods, interfaces, structs, fields, parameters, events, enums, properties, and delegates.

[ReplaceLiteralsWithName]

- Instructs the Obfuscator to replace all string literals within the obfuscated assemblies that match the target's name with the newly obfuscated name.
- Usable on classes, methods, interfaces, structs, fields, properties, events, enums, and delegates.

[Rename]

- Instructs the Obfuscator to change the target's name to the argument passed into this annotation.
- Usable on classes, methods, interfaces, structs, fields, properties, enums, and delegates.

[ObfuscateLiterals]

- Instructs the Obfuscator to apply string obfuscation on all string literals declared within the annotated method.
- Usable only on methods.

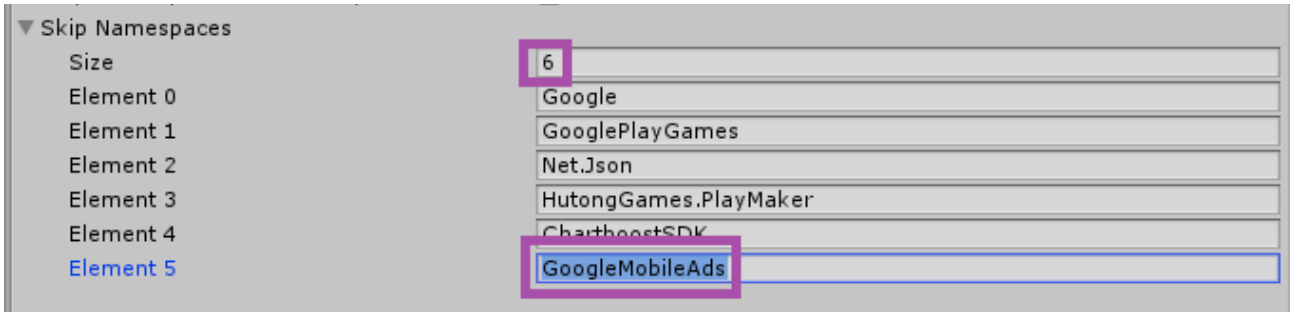
[DoNotFake]

- Instructs the Obfuscator to not spawn fake methods for the given target.
- Usable on classes or individual methods.
- You might want to use this on a single gigantic method so that you can benefit from having many fake methods for smaller methods without too much impact on file size. However you should strongly consider refactoring that large method into new classes and methods for cleaner code and stronger obfuscation results.

Troubleshooting

Parts of my game no longer works!

If you know the problem relates to a specific plugin, you might consider adding that plugin's namespace to the list of Skip Namespaces:



This can fix issues where that plugin's code is compiled alongside your project's code. Usually the plugin relies on reflection in some way.

If you're obfuscating a large complex project, start with only a small set of options enabled (i.e. start with only obfuscating class names) then gradually re-introduce more options.

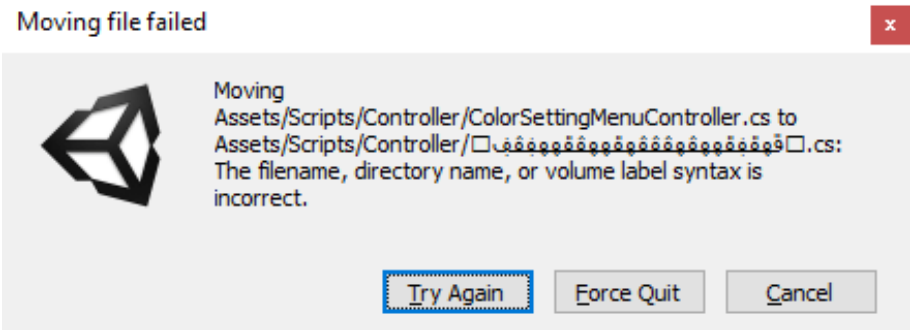
Keep in mind you can choose to prevent entire namespaces being obfuscated by using the Skip Namespaces section in options.

AssemblyResolutionException

This means the Obfuscator could not find the referenced DLL by default. Check the following:

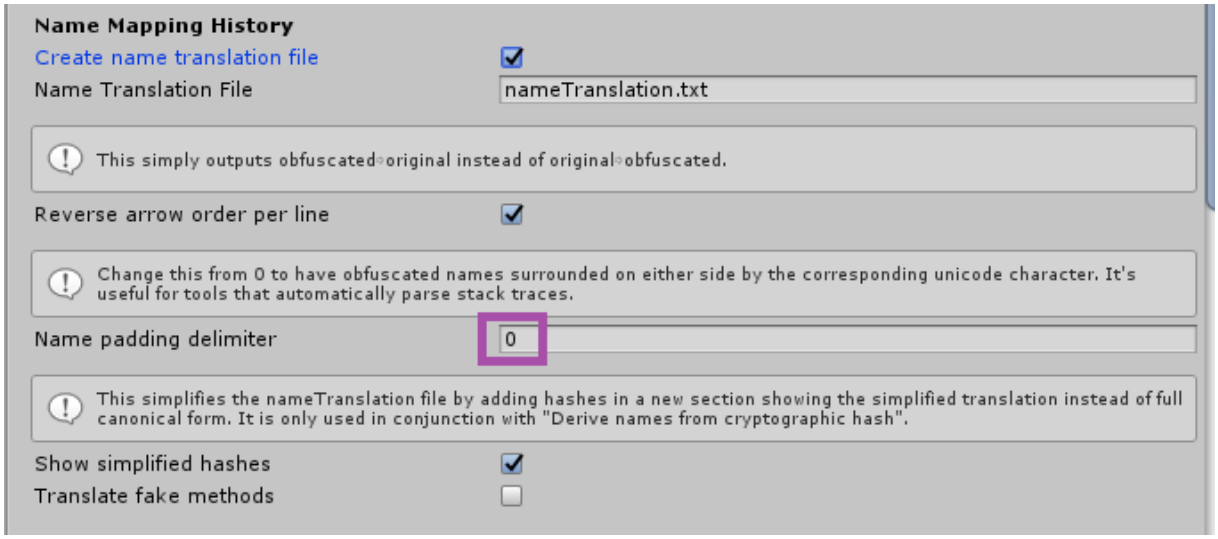
- If you renamed the DLL from its original name, rename it back.
- More likely, you need to tell the Obfuscator where to find this DLL by adding the DLL's directory to the list of "extraAssemblyDirectories" defined in Obfuscator Options in the Assemblies section.
  The Obfuscator will now consider that directory when looking for referenced assemblies.
- If you use a custom build process other than using the default (Postbuild.cs), make sure you call Obfuscator.SetExtraAssemblyDirectories(_options.extraAssemblyDirectories) before any call to Obufscate().

Moving file failed

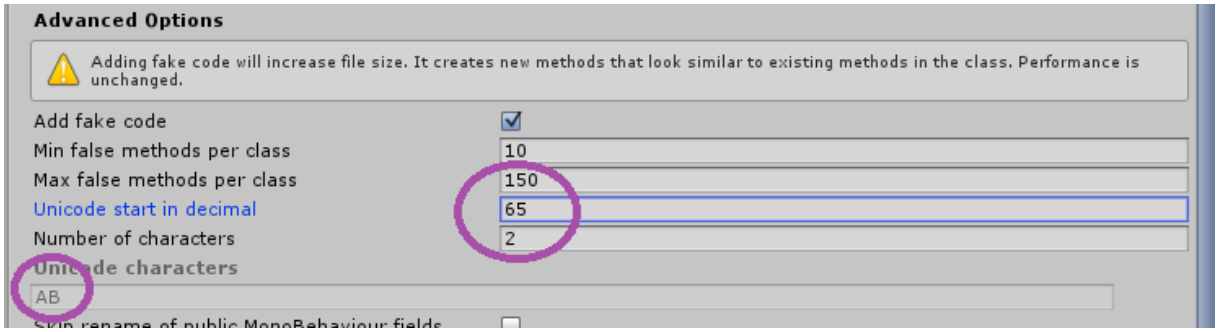


This can happen when obfuscating MonoBehaviour class names.

Try resetting the name padding delimiter (or change it to a file-format-friendly character):



If the error still occurs then change the character codes too:



MonoSymbolFileException

If you come across this error, please email us with as much information as possible to support@beebyte.co.uk including your Unity version, build target, and whether it was run in development mode.

It has been known to occur when the "Obfuscate literals in all methods" option is enabled.

A workaround that might work for you is to edit the Postbuild.cs file and make the following IgnoreSymbols call:

```
Obfuscator.IgnoreSymbols(true);
Obfuscator.Obfuscate(dlls, compiledDlls, _options,
EditorUserBuildSettings.activeBuildTarget);
```

Alternatively, disable the option "Obfuscate literals in all methods".

It takes too long to obfuscate in the build process

The culprit is almost always "Fake Code". Consider reducing the values within its options, or disable it.

If it still takes a long time to obfuscate, you can get a clearer idea of the cause by calling Obfuscator.SetPrintChronology(true) just before calls to Obfuscate() (See Postbuild.cs). Then on successful builds you will see additional information in the success message:

```
Obfuscation successful! (total time: 1831ms)
Initialisation: 1481ms
Parameters: 8ms
Methods Part 2: 162ms
Fields: 5ms
Properties: 1ms
Events: 1ms
Types: 2ms
References: 4ms
References 2: 1ms
LiteralObfuscation: 61ms
Finalisation: 82ms
NameTranslationFile: 16ms
```

Messages sent from Android aren't working

If in Java code you have:

```
public void SayHello() {
    UnityPlayer.UnitySendMessage(UNITY_GAMEOBJECT_HOOK, "OnSayHello", EMPTY);
}
```

then you need to annotate the OnSayHello method within your C# project with [SkipRename] to instruct the Obfuscator to leave that method untouched.

I need anonymous classes to be skipped

In the source code anonymous classes look like the 'anon' variable shown here:

```
public class NewBehaviourScript : MonoBehaviour
{
    public Text uiTextPanel;

    void Start ()
    {
        var anon = new {count = 1};

        uiTextPanel.text = anon.ToString();
    }
}
```

When compiled, they'll be given a name like <>__AnonType0`1

If you need these anonymous classes to be skipped, then add the following line to the "Equivalent Attributes for Skip" section:

System.Runtime.CompilerServices.CompilerGenerated

This works because anonymous classes will have the [CompilerGenerated] attribute applied.