
Auto
Singleton

Contents

Namespace 3

Automatically creating a Singleton 3

- Basic 3
- Advanced 4

Retrieving a Singleton 4

- Single Instance 4
- Multiple Instances 4
- Selecting an Instance 5
- Find Method 5

Manually creating a Singleton 6

- Explanation 6
- Methods 6

Top bar menu 7

Demo 8

Troubleshooting 8

- Caching a singleton reference from another one 8
- Corrupted singleton list 9

Final words 9

Namespace

All Auto Singleton classes are in the **AutoSingleton** namespace.

Add this at the top of your scripts to access them:

```
using AutoSingleton;
```

Automatically creating a Singleton

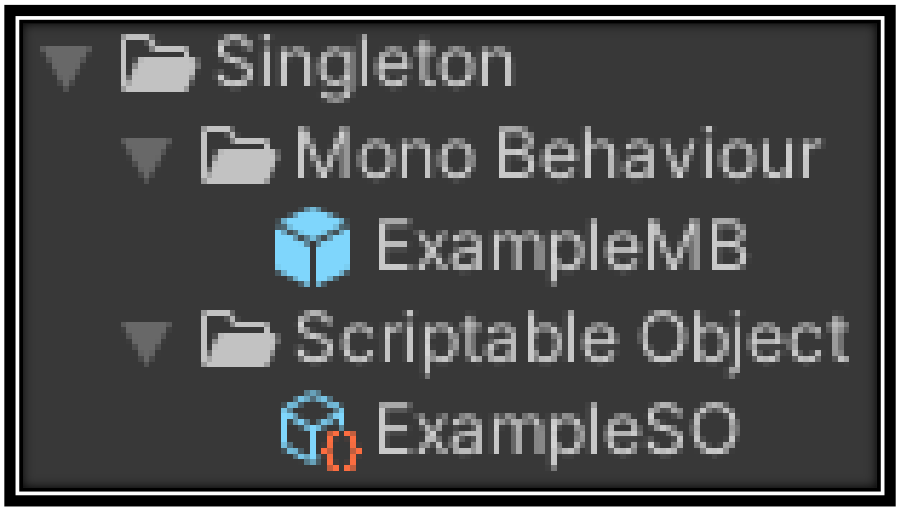
Basic

To create a singleton managed automatically, simply add the **Singleton** attribute over a class derived from **MonoBehaviour** or **ScriptableObject**:

```
[Singleton]
public class ExampleMB : MonoBehaviour
```

```
[Singleton]
public class ExampleSO : ScriptableObject
```

An instance of it will be created in the project **Assets** folder.



You can freely rename, move, and modify it. If you delete the singleton component from a prefab asset, a new one will be created.

Advanced

The **Singleton** attribute have different fields that you can set:

```
[Singleton(
    inherited = true,
    displayName = "Example Display Name",
    folderPath = "Folder/SubFolder"
)]
```

inherited
If true, this attribute will also be applied to any derived classes.

displayName
Choose the name that the singleton instance will have in **Assets** (not inherited).

folderPath
Choose where the singleton instance will be created, relative to the **Assets/** folder (inherited).

Retrieving a Singleton

Single Instance

To access a singleton from scripts, all you have to do is use the **Singleton<T>** class **Instance** property:

```
ExampleMB singletonInstance = Singleton<ExampleMB>.Instance;
```

Multiple Instances

You can also get multiple singletons derived from a parent class using **Instances**:

```
IReadOnlyList<ParentMB> allMBs = Singleton<ParentMB>.Instances;
```

You can even do it using an interface that your singletons implement:

```
var allImplementer = Singleton<IInterface>.Instances;
```

Selecting an Instance

Using the **Instance** property when multiple singleton instances correspond to the chosen template parameter of **Singleton<T>** will throw an **Exception**.

You can use the **SelectInstance** method to choose which singleton to get when calling **Instance**.

This lets you have one piece of code define what singleton to use, and all your other scripts adapt by using the chosen instance.

```
// In one script...
Singleton<ParentMB>.SelectInstance((pmb) =>
{
    return pmb.enabled;
});

// And in another...
Singleton<ParentMB>.Instance.DoSomething();
```

Find Method

Using the **Find** method in **Singleton<T>**, you can get every singleton that matches the given predicate (a function that returns either true or false).

```
Object[] match = Singleton.Find((singleton) =>
{
    return singleton.name.StartsWith("a");
});
```

This will allocate a new array on each call; you should cache the result if you need it later.

Manually creating a Singleton

Explanation

In some advanced scenarios, you may want to have more control over how singletons are managed.

To account for that, you can add and remove singletons to the list using the **Singleton** static class (only in play mode).

This lets you have singletons of ANY reference type, even if they don't inherit from **MonoBehaviour** or **ScriptableObject**.

This also lets you manage their lifetime yourself.

Singletons added this way can be accessed like automatically managed ones (see above).

Methods

Add:

Add the given instance(s) to the singleton list.

This will throw an **Exception** if this instance or a singleton of the same type already exists in the singleton list.

Note that if you add a **MonoBehaviour**, DontDestroyOnLoad won't be called on it.

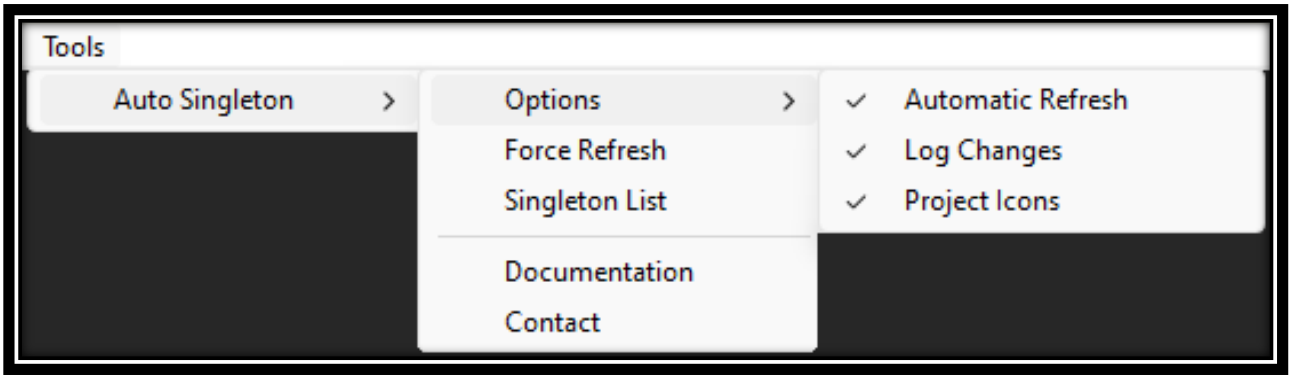
Remove:

Remove the given instance(s) from the singleton list.

This will throw an **Exception** if this instance is not in the singleton list, or if it is managed automatically (if it was not added using the **Add** method).

Top bar menu

In **Tools/Auto Singleton** you will find different menu item:



Options/Automatic Refresh
If enabled, the singletons will be created and deleted automatically after every successful compilation.

Options/Log Changes
If enabled, we will log in the console every singleton changes.

Options/Project Icons
If enabled, we will display an icon on singleton instances in the **Project** (Assets) window.

Force Refresh
Trigger the creation and deletion of singletons according to the **Singleton** attribute usage.

Singleton List
Open in the inspector the list of all singletons, handy to easily find an instance in the **Assets** folder. It lets you disable any singleton, preventing them from being instantiated at runtime.

Documentation
Open this pdf.

Contact
Open a web page giving you ways to contact the tool creator.

Demo

The demo is a simple Tic Tac Toe game made following an MVC pattern.

It showcases 5 singletons, check them out to see how to use the tool in different scenarios.

It is recommended to delete the demo folder once you've explored it, to avoid instantiating useless singletons and to remove unused assets.

Troubleshooting

Caching a singleton reference from another one

You may want to cache a singleton reference in a field.

If you do this from another automatically managed **MonoBehaviour** singleton, do NOT do it in the **Awake** method, do it from **Start** instead.

```
[Singleton]
class MySingleton : MonoBehaviour
{
    AnotherSingleton anotherSingleton;

    void Start() // Do not do this in Awake.
    {
        anotherSingleton = Singleton<AnotherSingleton>.Instance;
    }
```

Note that this is barely an optimization because the **Instance** property of **Singleton<T>** is pretty much as optimized as possible during runtime.

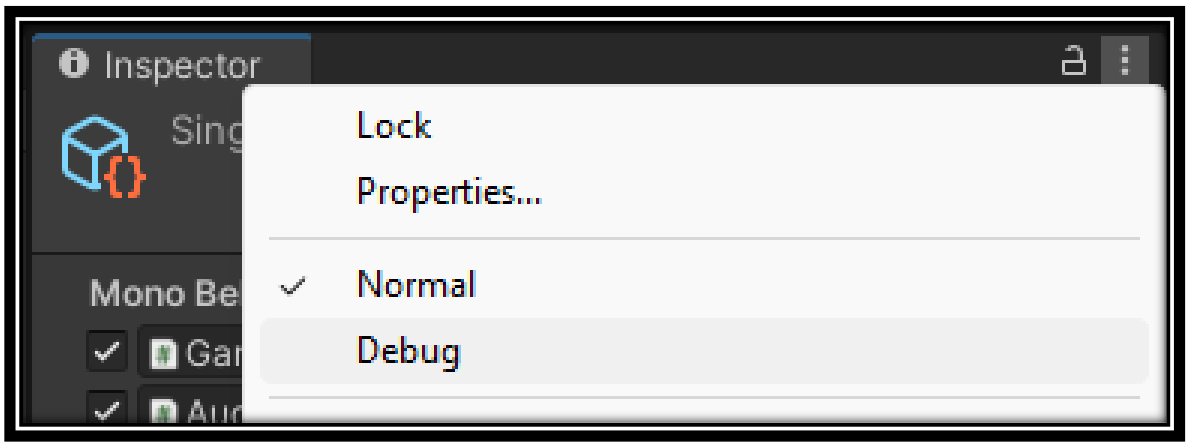
Corrupted singleton list

You may run into a situation where the automatically managed singleton list no longer holds the correct asset instances (if you modify it yourself, from Unity or outside of it).

If this happens the tool will populate it with newly created assets, which could be a problem if you don't want to loose your singletons data.

To fix this, you can access the singleton list in debug mode and assign the asset references yourself.

Click on the three dot in the top right corner of the inspector window, and select "**Debug**".



Final words

Thank you for buying this tool!

If you have any questions, discover a bug, or have a feature idea, please don't hesitate to contact me at justetools@gmail.com.

If you enjoy **Auto Singleton**, please consider leaving a review. Your feedback helps improve the tool!