MethodClicker user manual

In this document you will find the instructions on how to set up and use MethodClicked.

If you have any question, feature request or bug report about this asset please contact me by email: celtic.gua@gmail.com

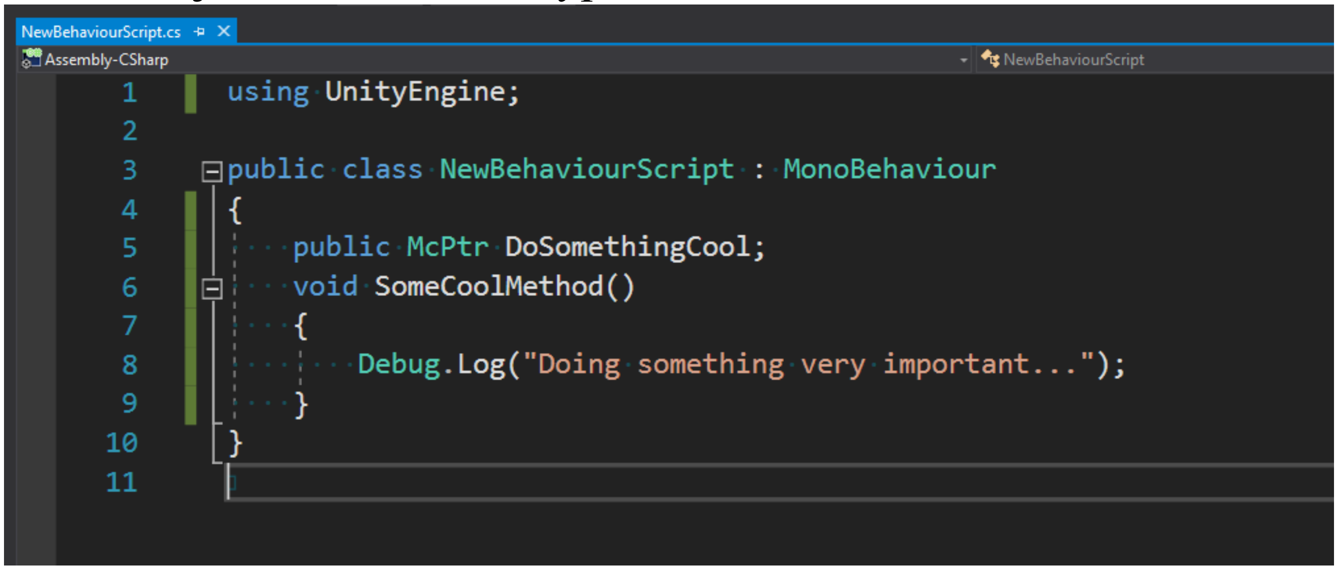
How to set up

In order to start using Method clicker just import it in Unity and you ready to go!

How to use

First you need a method that you want to be able to invoke. It must return **void** and accept no arguments.

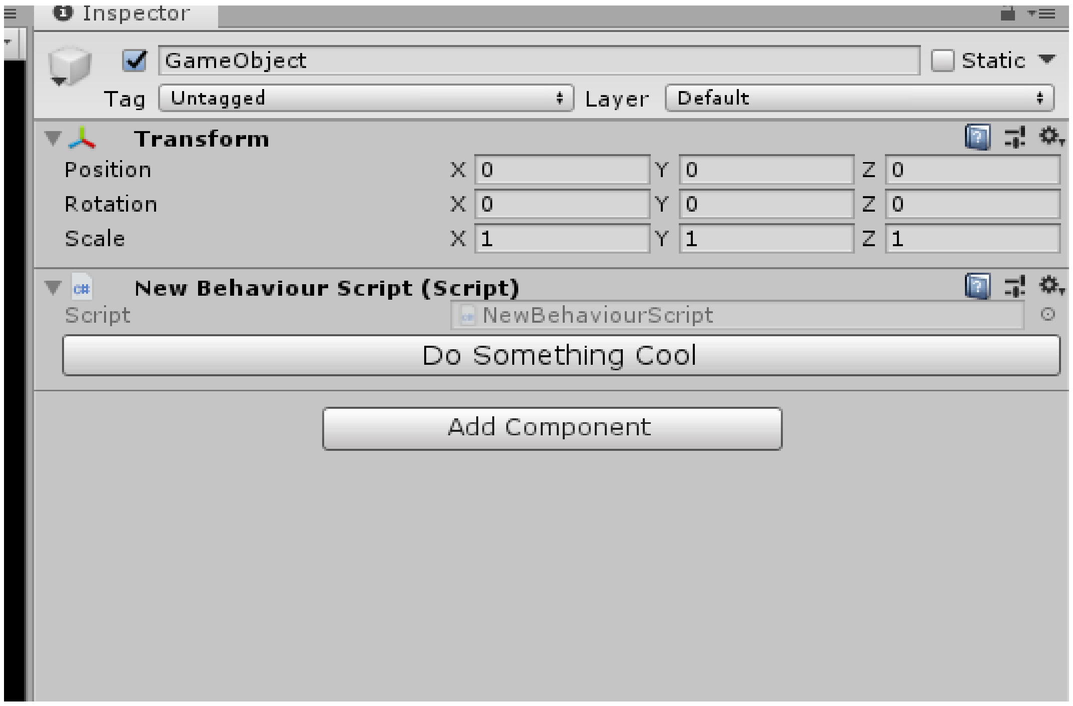
Now just add a field of type ***McPtr*** before it:



NewBehaviourScript.cs
Assembly-CSharp
NewBehaviourScript

```
using UnityEngine;

public class NewBehaviourScript : MonoBehaviour
{
    public McPtr DoSomethingCool;
    void SomeCoolMethod()
    {
        Debug.Log("Doing something very important...");
    }
}
```

That's it. Now if you check the inspector of that class you'll see the button:



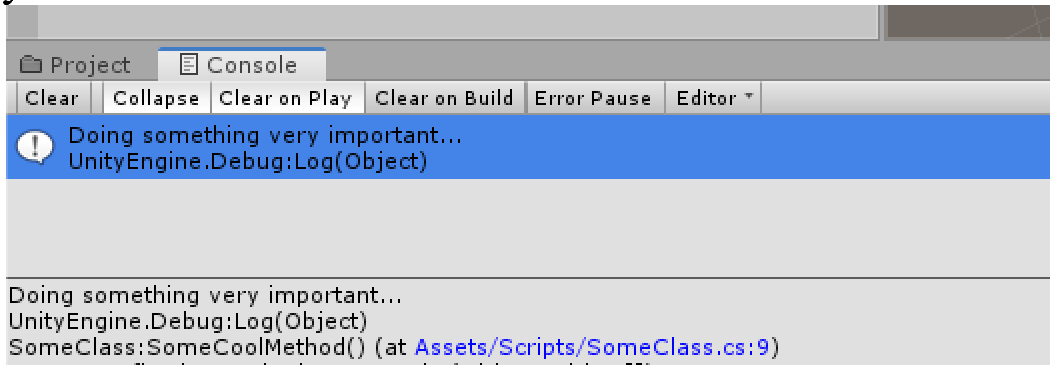
Inspector
GameObject Static
Tag Untagged Layer Default

Transform

Position	X	0	Y	0	Z	0
Rotation	X	0	Y	0	Z	0
Scale	X	1	Y	1	Z	1

New Behaviour Script (Script)
Script NewBehaviourScript
Do Something Cool
Add Component

If you click on it – it'll invoke the method:



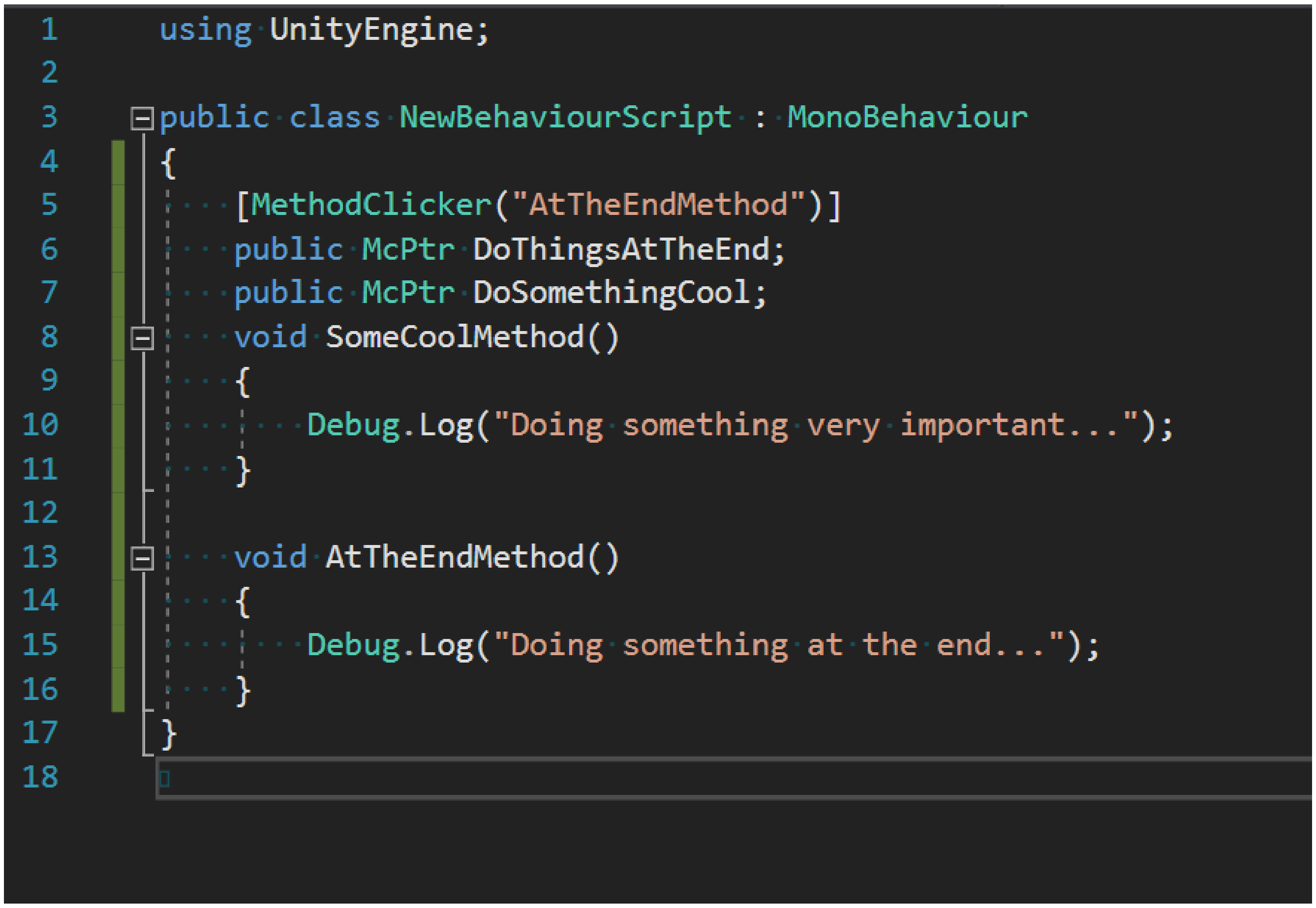
```
Doing something very important...
UnityEngine.Debug:Log(Object)
```

```
Doing something very important...
UnityEngine.Debug:Log(Object)
SomeClass:SomeCoolMethod() (at Assets/Scripts/SomeClass.cs:9)
```

Important Note: the field of type ***McPtr*** must be public or private with **SerializeField** attribute before it.

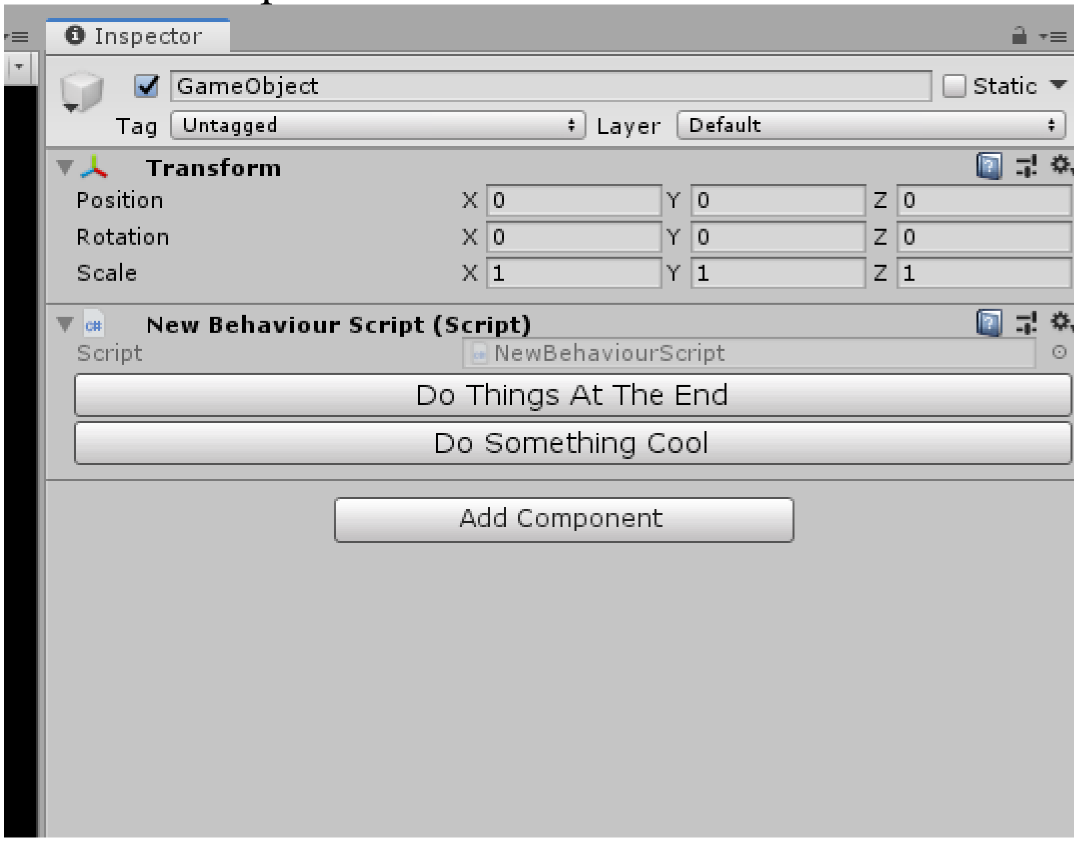
MethodClicker attribute

You're not limited with only the method next to the **McPtr** field. You can bind **McPtr** to any method in that class by attaching ***MethodClicker*** attribute to it. Add it to the ***McPtr*** field and specify the name of a function you want it to be binded to:

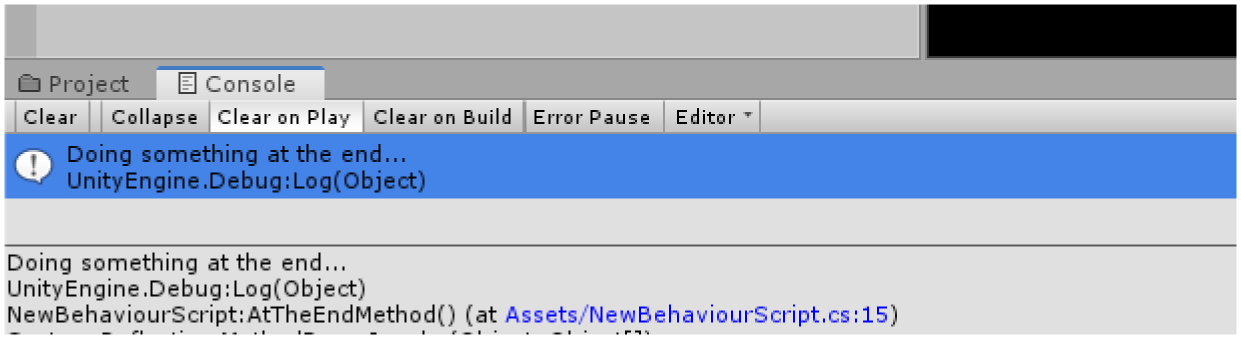


```
using UnityEngine;

public class NewBehaviourScript : MonoBehaviour
{
    [MethodClicker("AtTheEndMethod")]
    public McPtr DoThingsAtTheEnd;
    public McPtr DoSomethingCool;
    void SomeCoolMethod()
    {
        Debug.Log("Doing something very important...");
    }

    void AtTheEndMethod()
    {
        Debug.Log("Doing something at the end...");
    }
}
```

The look of inspector:



When we click on "Do Things At The End" button, we see that *AtTheEndMethod* is invoked:

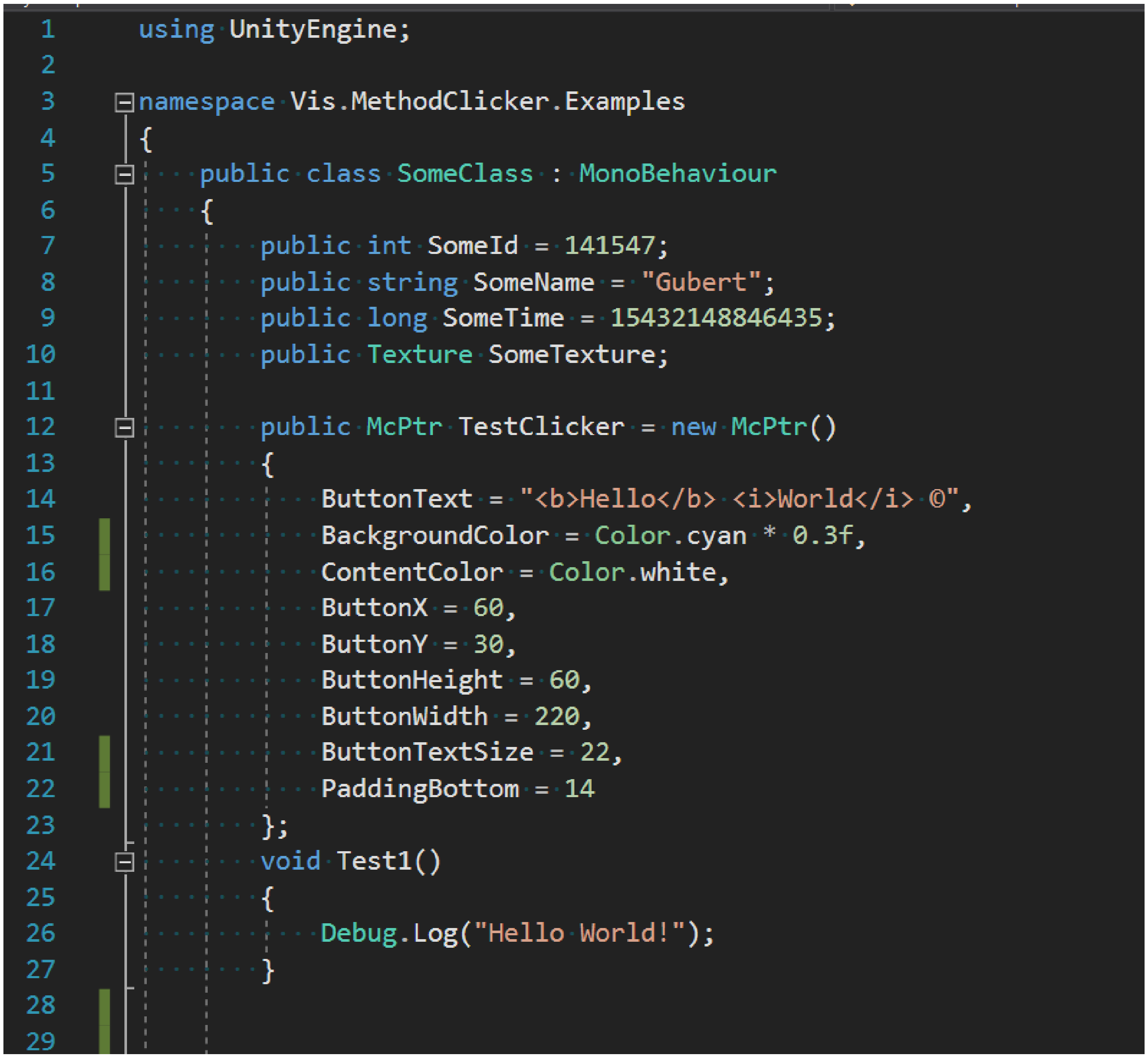


```
Doing something at the end...
UnityEngine.Debug:Log(Object)
```

```
Doing something at the end...
UnityEngine.Debug:Log(Object)
NewBehaviourScript:AtTheEndMethod() (at Assets/NewBehaviourScript.cs:15)
```

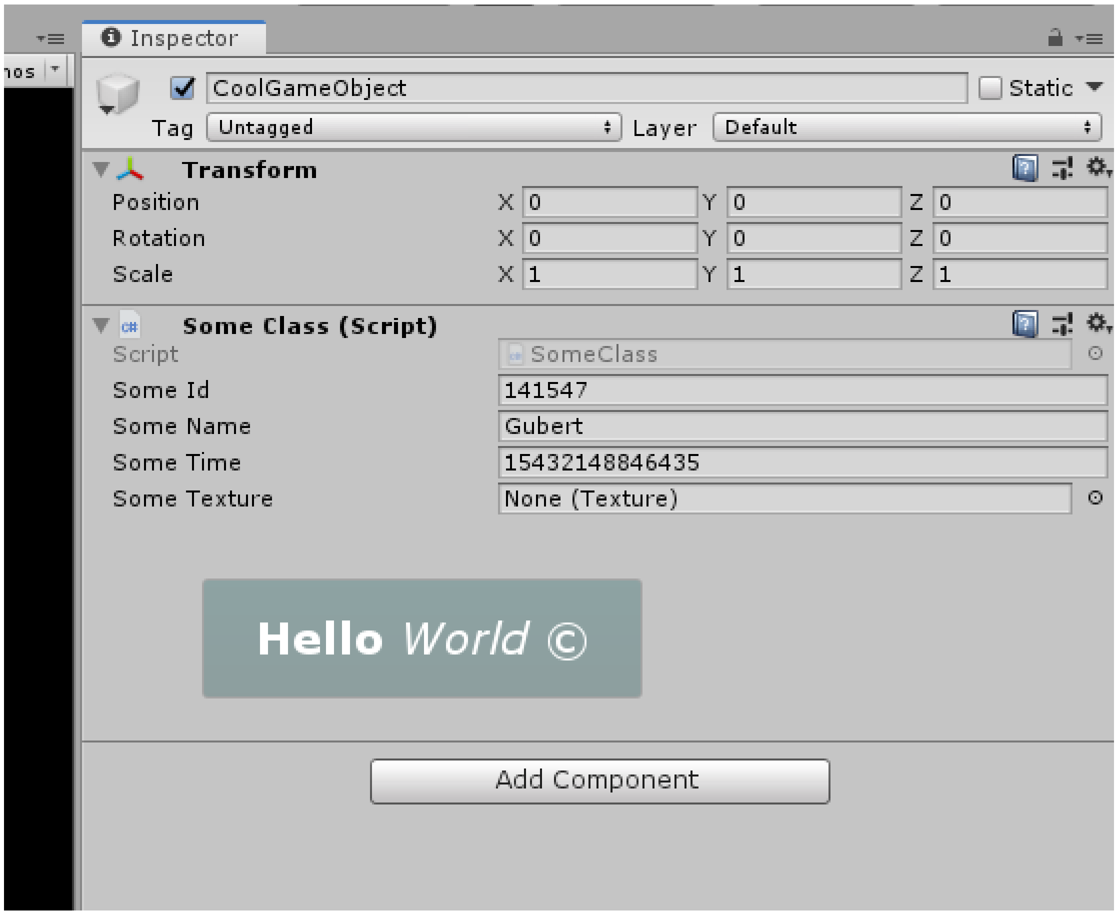
Visual features

You can make buttons look any way you want with **McPtr** class fields. Just create and instance of **McPtr** class, define the fields you want and customize the visual appearance as you want. The names of the fields are self explanatory and easy to use:



```
using UnityEngine;

namespace Vis.MethodClicker.Examples
{
    public class SomeClass : MonoBehaviour
    {
        public int SomeId = 141547;
        public string SomeName = "Gubert";
        public long SomeTime = 15432148846435;
        public Texture SomeTexture;

        public McPtr TestClicker = new McPtr()
        {
            ButtonText = "<b>Hello</b> <i>World</i> ©",
            BackgroundColor = Color.cyan * 0.3f,
            ContentColor = Color.white,
            ButtonX = 60,
            ButtonY = 30,
            ButtonHeight = 60,
            ButtonWidth = 220,
            ButtonTextSize = 22,
            PaddingBottom = 14
        };
        void Test1()
        {
            Debug.Log("Hello World!");
        }
```

Here's what type of look this code produces:



CoolGameObject

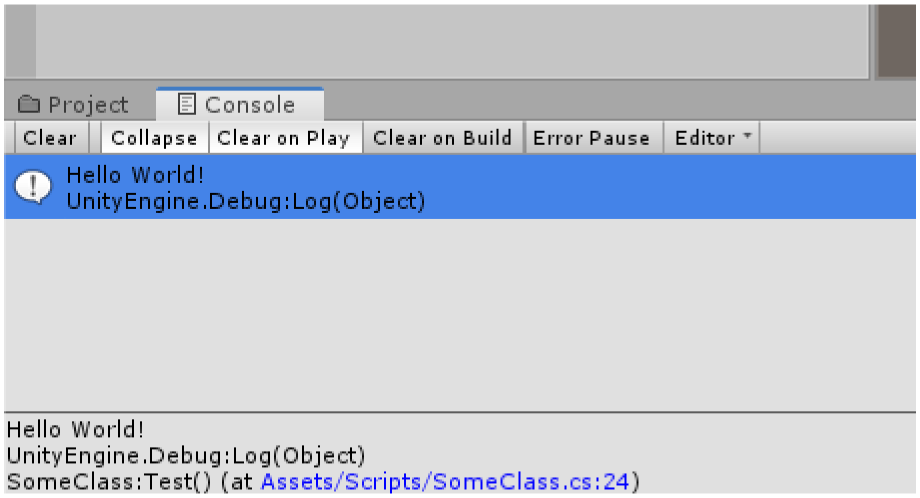
Tag Untagged   Layer Default

Position	X 0	Y 0	Z 0
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

Script	SomeClass
Some Id	141547
Some Name	Gubert
Some Time	15432148846435
Some Texture	None (Texture)

Hello *World* ©

And it works as the usual button when clicked:



```
Hello World!
UnityEngine.Debug:Log(Object)
```

```
Hello World!
UnityEngine.Debug:Log(Object)
SomeClass:Test() (at Assets/Scripts/SomeClass.cs:24)
```

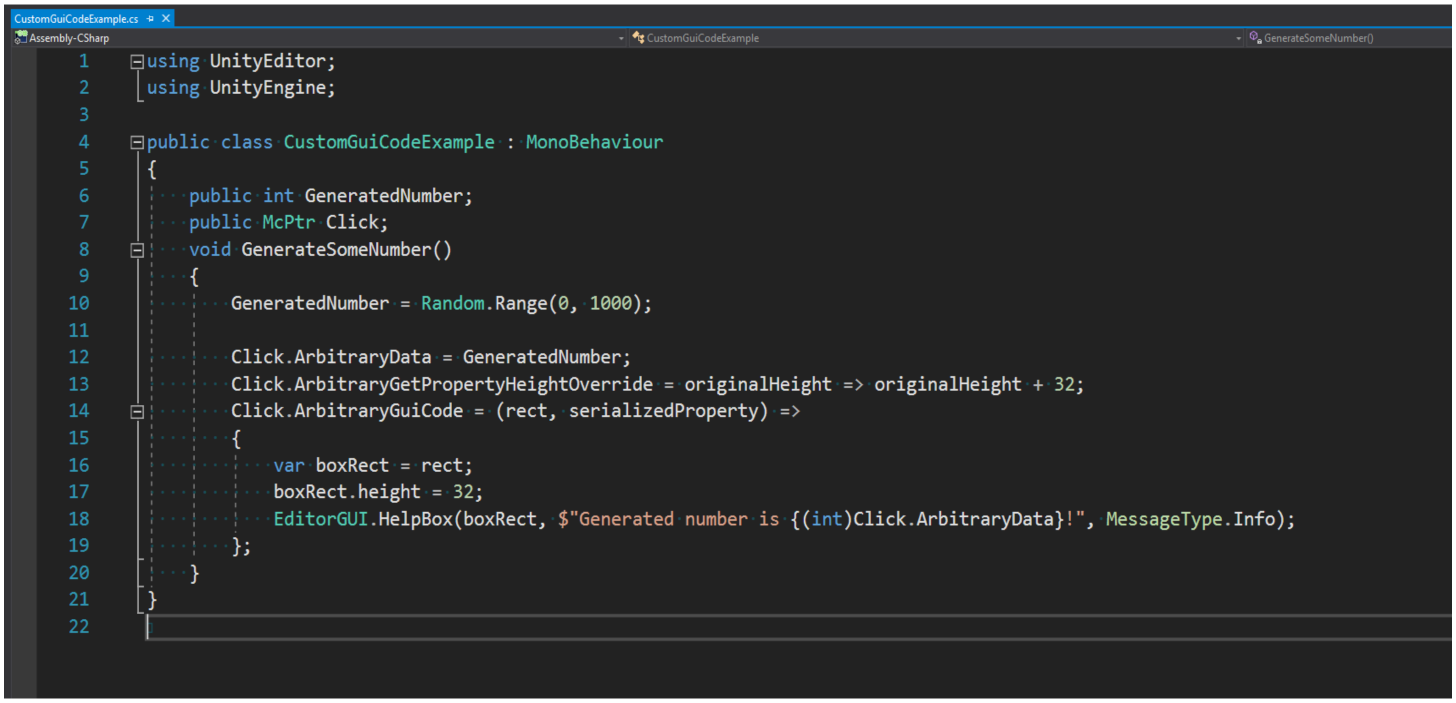
Arbitrary GUI

You can specify any additional editor gui as you want with *ArbitraryGuiCode* field of **McPtr** type. This field is of *Action<Rect, SerializedProperty>* type and it's invoked right after Button drawing complete, so you can add any additional things you want.

There's also and *ArbitraryGetPropertyHeightOverride* field of type *Func<float, float>*. It serves as a PropertyDrawer's field area height modifier hook. This function passes the height of the field after the button was drawn and expects the final height of field area, so you could allocate the required height for your GUI.

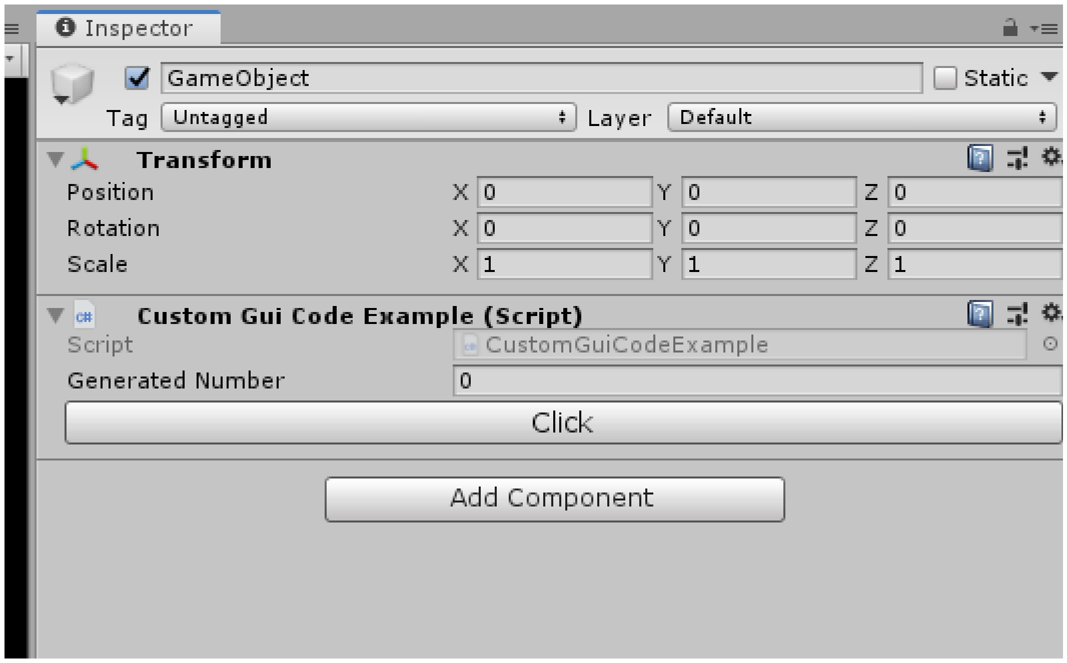
ArbitraryData field serves as a container for any state you want to persist for your logic.

In the following example the *HelpBox* is drawn when user clicks on the *Click* button generated by **McPtr** and shows some data produced by that method:



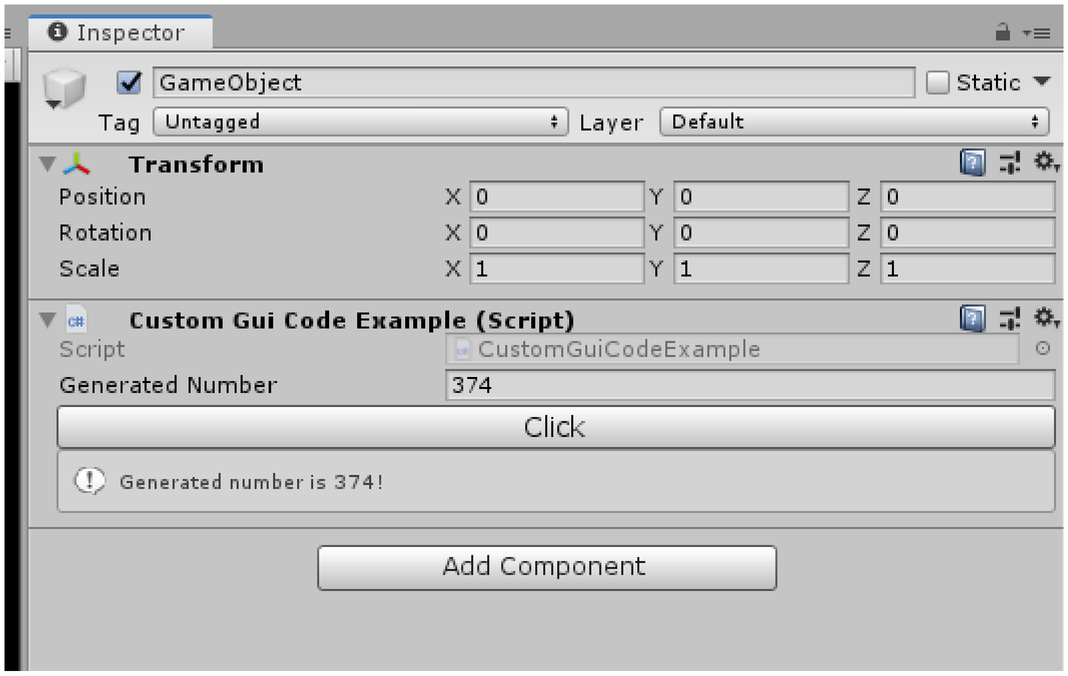
```
using UnityEditor;
using UnityEngine;

public class CustomGuiCodeExample : MonoBehaviour
{
    public int GeneratedNumber;
    public McPtr Click;
    void GenerateSomeNumber()
    {
        GeneratedNumber = Random.Range(0, 1000);

        Click.ArbitraryData = GeneratedNumber;
        Click.ArbitraryGetPropertyHeightOverride = originalHeight => originalHeight + 32;
        Click.ArbitraryGuiCode = (rect, serializedProperty) =>
        {
            var boxRect = rect;
            boxRect.height = 32;
            EditorGUI.HelpBox(boxRect, $"Generated number is {(int)Click.ArbitraryData}!", MessageType.Info);
        };
    }
}

```

Here's how it looks before clicking the button:



```
GameObject                         Static
Tag  Untagged          Layer  Default
Position     X 0   Y 0   Z 0
Rotation     X 0   Y 0   Z 0
Scale        X 1   Y 1   Z 1
Script              CustomGuiCodeExample
Generated Number    0
```

And after clicking button - *HelpBox* is drawn:



```
GameObject                         Static
Tag  Untagged          Layer  Default
Position     X 0   Y 0   Z 0
Rotation     X 0   Y 0   Z 0
Scale        X 1   Y 1   Z 1
Script              CustomGuiCodeExample
Generated Number    374
Generated number is 374!
```

Important Notes

The **MethodClicker** asset utilizes standard *PropertyDrawer* API of UnityEditor, so it'll work in any place where PropertyDrawers work. That is: Components, Assets, ScriptableObjects, StateMachineBehaviours etc...

Note that the **McPtr** fields should be serialized by Unity serializer in order to work – i.e. they must be public or private with *SerializeField* attribute on it. The consequence of that is that such fields are included in final build, so the small footprint will take place. That was made for the sake of usability and extendability – no other way would allow you to make such customizable button in only one line of code.

Please beware that ***SerializedProperty*** class that used in *ArbitraryGuiCode* field of **McPtr** class is **editor-only** class, so if you use that field in your code make sure that this part of code will be ignored by final build compiler. Otherwise it'll cause exceptions during build. One possible way to prevent it is to wrap this code inside #if UNITY_EDITOR … #endif region. That will make this code visible only for Unity Editor.