Runtime C# Behaviours

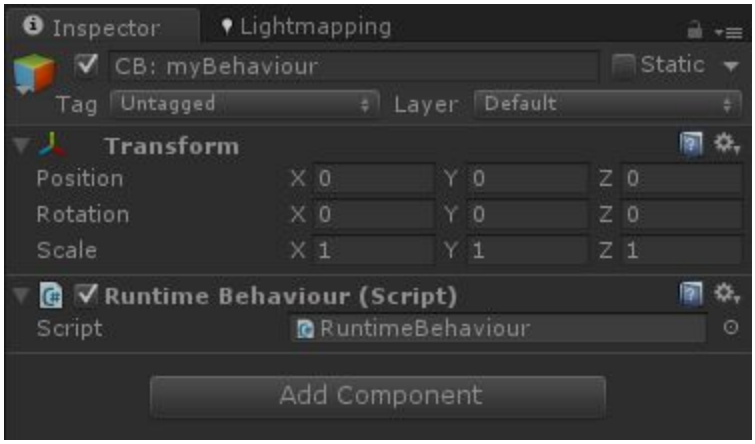
When executing C# code at runtime, sometimes you will want execute code in one of the standard Unity event functions (`Update()` for example). Unfortunately, due to the code being generated at runtime, creating a new `MonoBehaviour` class isn't possible. TNet has this covered with the `RuntimeBehaviour` class. It can create a GameObject with a listener class on the fly.

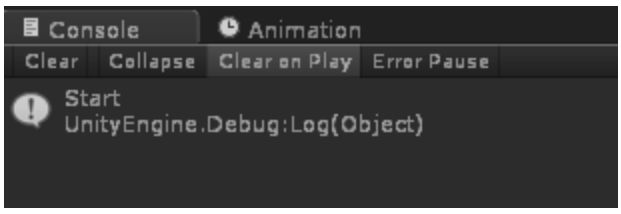
The static `Create()` function is used to generate a behaviour and can take a string as its unique name. If no name is provided it will generate a new one. Each behaviour must define at least one function, or it will be deleted on start.

RuntimeBehaviour supports the following Unity functions: `Start`, `Update`, `FixedUpdate`, `OnDestroy` and one `Custom` function. If you define a custom function, it will need to be triggered manually, and is mainly used to pass delegate references between RuntimeBehaviours when this is required.

```
RuntimeBehaviour myBehaviour = RuntimeBehaviour.Create("myBehaviour");

myBehaviour.onStart = delegate (RuntimeBehaviour rb)
{
	Debug.Log("Start");
};
```





This code snippet creates a new object called “CB: myBehaviour” and attaches a runtime behaviour class to it, which then writes to the console log.

At this point you may be wondering, how do I create member variables? C# has a handy feature which we can exploit in this case. Anonymous delegates in C# have access to **all local variables** in the scope where they were declared.

```
int myInteger = 5;

RuntimeBehaviour.Create("myBehaviour").onUpdate = delegate (RuntimeBehaviour rb)
{
	Debug.Log("Update " + myInteger);
	rb.onUpdate = null;
};
```

This example will write “Update 6” once in the console log. How is this useful? These variables can be used to do what you would normally do by declaring class fields, such as cache references to objects in the scene.

```
Transform camera = GameObject.Find("Main Camera").transform;
Quaternion target = Quaternion.Euler(0f, Random.Range(0f, 180f), 0f);

RuntimeBehaviour.Create("myBehaviour").onUpdate = delegate (RuntimeBehaviour rb)
{
	camera.rotation = Quaternion.RotateTowards(
		camera.rotation, target, 0.5f);
};
```

In this example an Update function is added, it stores a reference to the main camera’s transform in the external `camera` variable, and then rotates towards the target. If you would like to use the `+=` and `-=` functionality of delegate callbacks, you can store a reference to the anonymous delegate in a variable.

```
RuntimeBehaviour.Callback myDelegate = delegate(RuntimeBehaviour rb) {};
RuntimeBehaviour.Create("myBehaviour").onUpdate += myDelegate;
```

Example

Here is an example from a Windward modification which changes the look of the world and uses most of the features explained here. This particular code is saved in plain text and gets executed using `RuntimeBehaviour.ExecuteFile()`.

```
RuntimeBehaviour.Create().onUpdate = delegate(RuntimeBehaviour rb)
```

```
{
    if (UILoadingScreen.isVisible) return;
    rb.onUpdate = null;

    float reflectionTint = 0.2f;
    Color ambientLight = new Color(0.3f, 0.35f, 0.4f);
    Color water0 = new Color(1f, 1.4f, 1.6f, 1f);
    Color water1 = new Color(0f, 0.2f, 0.3f, 1f);

    Color willow0 = new Color(0.35f, 0.6f, 0f, 1f);
    Color willow1 = new Color(0.1f, 0.2f, 0f, 1f);
    Color spruce0 = new Color(0.25f, 0.4f, 0f, 1f);
    Color spruce1 = new Color(0f, 0f, 0f, 1f);

    string slopeTexture = "Textures/Temperate 0 C";
    string slopeNormal = "Textures/Temperate 0 CN";
    string shoreTexture = "Textures/Temperate 1 A";
    string grassTexture = "Textures/Temperate 3 B";
    string hillTexture = "Textures/Temperate 2 C";

    GameObject go = GameObject.Find("Procedural Zone");
    PaintTerrain paint = go.transform.GetComponentInChildren<PaintTerrain>();
    paint.slopeTexture = ResourceLoader.LoadTexture(slopeTexture, true);
    paint.slopeNormal = ResourceLoader.LoadTexture(slopeNormal, true);
    paint.shoreTexture = ResourceLoader.LoadTexture(shoreTexture, true);
    paint.grassTexture = ResourceLoader.LoadTexture(grassTexture, true);
    paint.hillTexture = ResourceLoader.LoadTexture(hillTexture, true);
    paint.Repaint();

    RenderSettings.ambientLight = ambientLight;

    if (TasharenWater.instance != null)
    {
        Renderer ren = TasharenWater.instance.GetComponent<Renderer>();
        Material mat = ren.material;
        mat.SetColor("_Color0", water0);
        mat.SetColor("_Color1", water1);
        mat.SetFloat("_ReflectionTint", reflectionTint);
    }

    RuntimeBehaviour rb2 = RuntimeBehaviour.Create("TexUpdate");
    ZoneCreator.onTerrainModified -= rb2.Custom;

    rb2.onCustom = delegate(RuntimeBehaviour r)
    {
        RuntimeBehaviour.Create().onUpdate = delegate(RuntimeBehaviour rb3)
        {
            if (UILoadingScreen.isVisible) return;
            rb3.onUpdate = null;

            Renderer[] rens = GameObject.Find("Trees").GetComponentsInChildren<Renderer>();

            foreach (Renderer ren in rens)
            {
                Material mat = ren.material;

                if (mat.name.StartsWith("Willow"))
                {
                    mat.SetColor("_Color", willow0);
                    mat.SetColor("_Secondary", willow1);
                }
```

```
                else if (mat.name.StartsWith("Spruce"))
                {
                    mat.SetColor("_Color", spruce0);
                    mat.SetColor("_Secondary", spruce1);
                }
            }
        };
    };

    rb2.onDestroy = delegate(RuntimeBehaviour r)
    {
        ZoneCreator.onTerrainModified -= r.Custom;
    };

    if (rb2.onCustom != null) rb2.Custom();

    ZoneCreator.onTerrainModified += rb2.Custom;
};
```