STORY GRAPH

Documentation for:

StoryGraph: A Node Based Visual Scripting Tool

Version #1.00

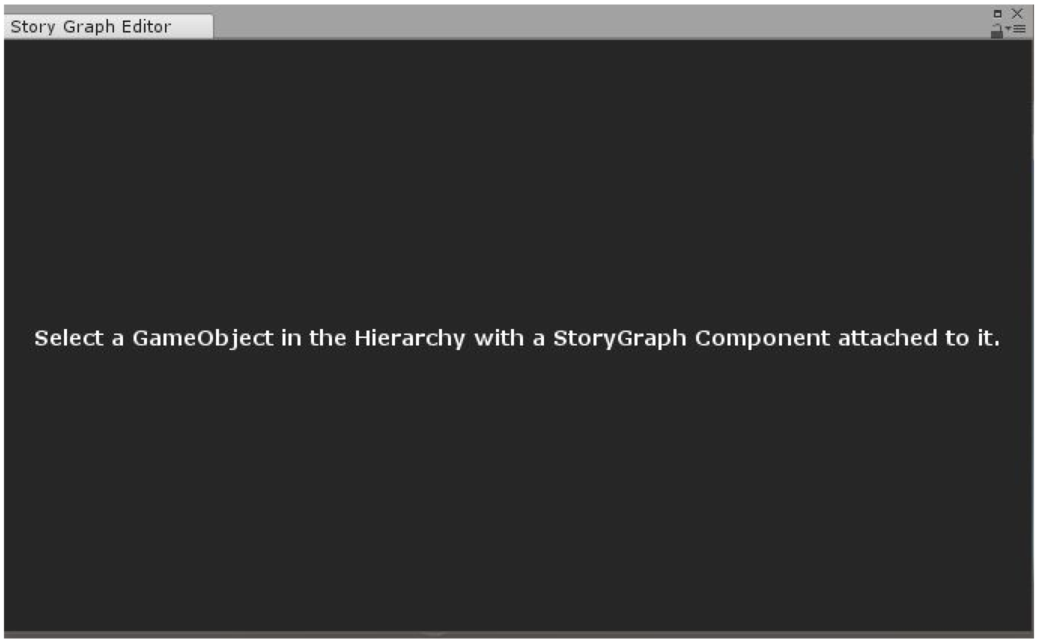
Support Email: mattywagar@gmail.com

Table of Contents

Editor Quickstart

1)

To start using the Story Graph, just go to Window -> Story Graph Editor

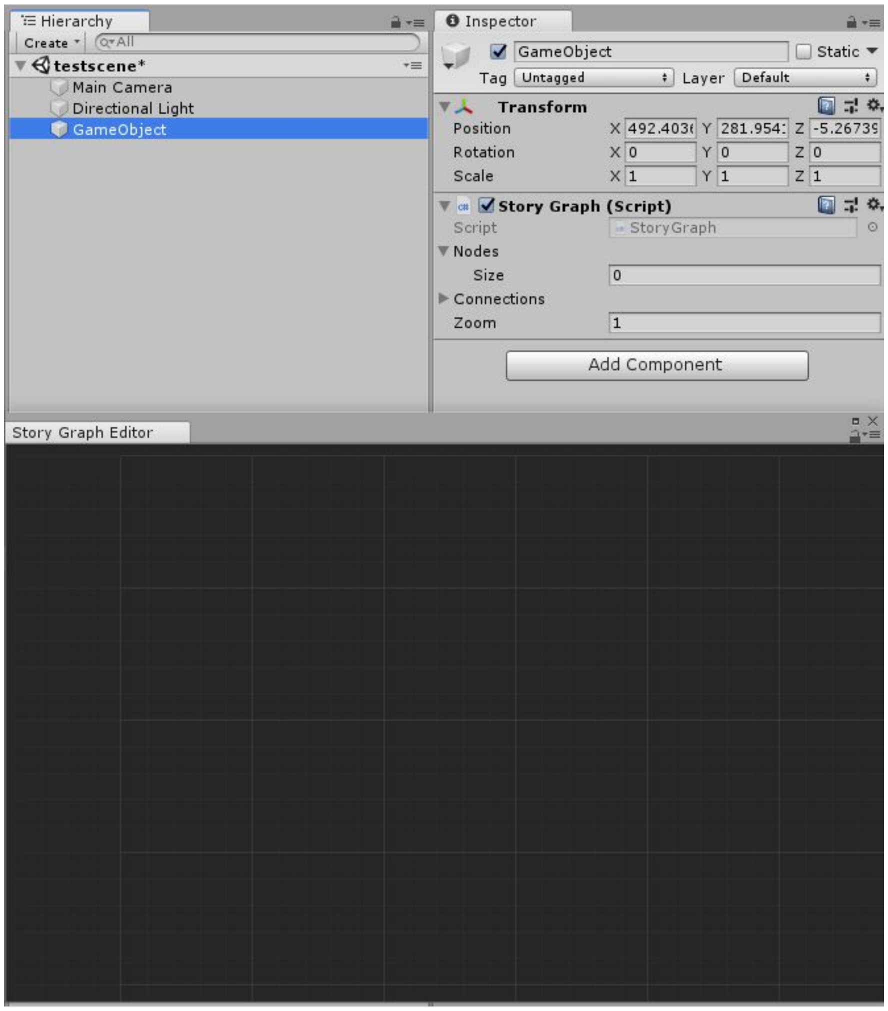


Select a GameObject in the Hierarchy with a StoryGraph Component attached to it.

2)

It will prompt you to add a Story Graph component to a GameObject.

After you add the Story Graph component, your window will look like this:



GameObject

Tag: Untagged   Layer: Default

Transform

	X	Y	Z
Position	492.403(	281.954:	-5.2673S
Rotation	0	0	0
Scale	1	1	1

Story Graph (Script)

Script: StoryGraph

Nodes

Size: 0

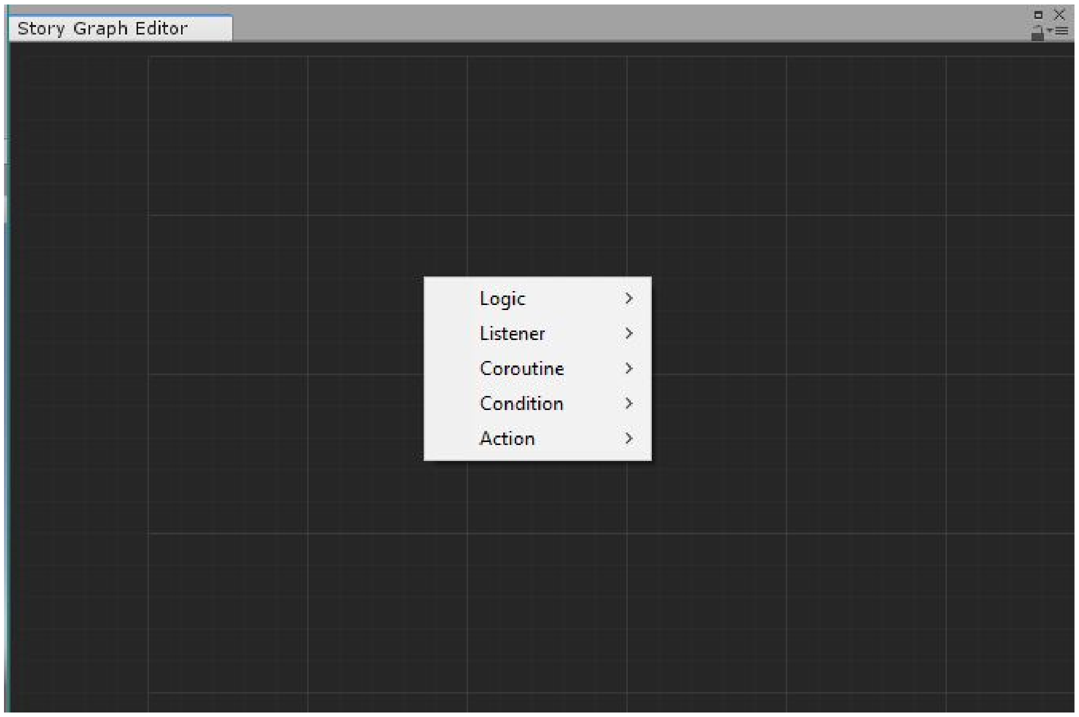
Connections

Zoom: 1

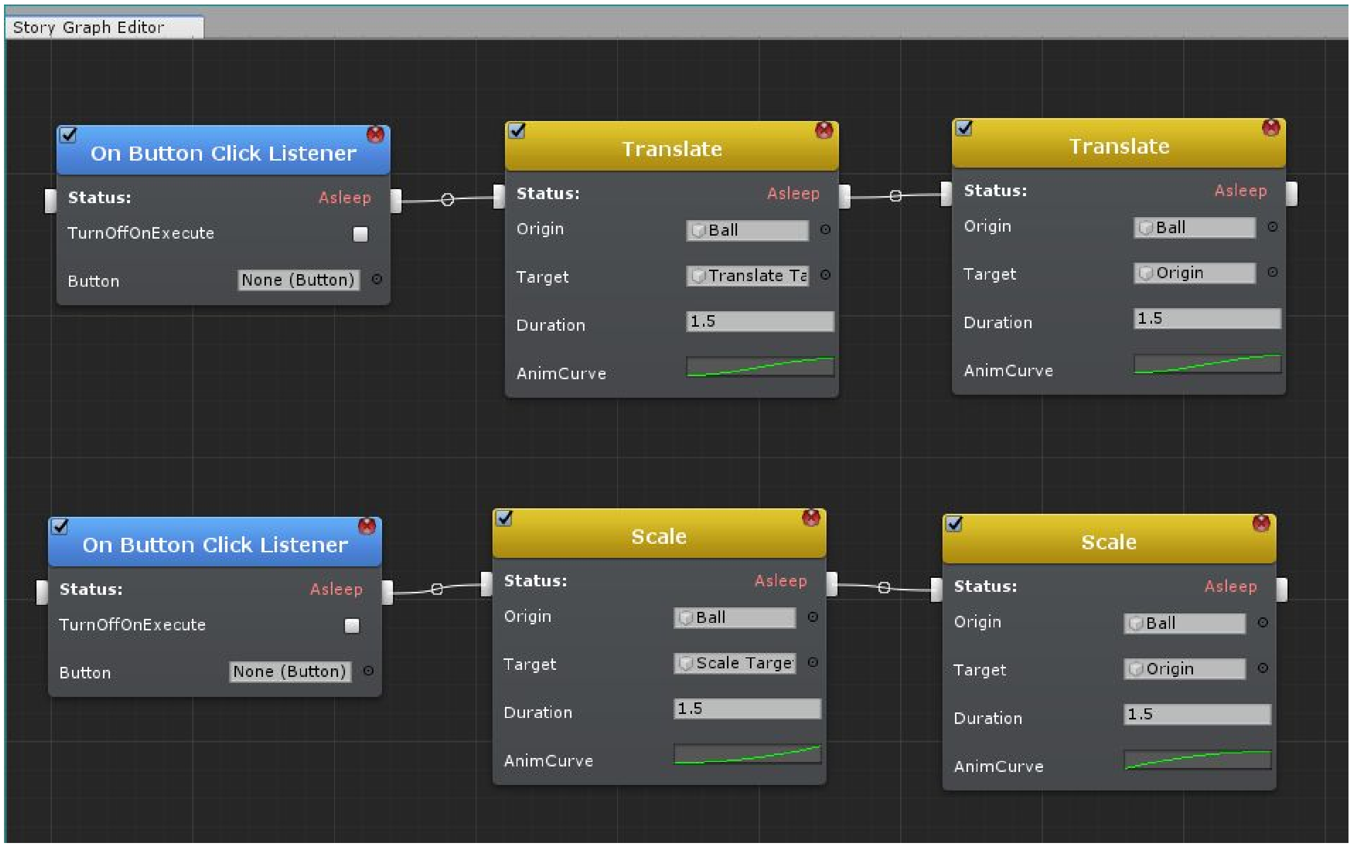
3)

From there you are ready to start scripting!

Right click anywhere on the Story Graph window and select the type of node you want to use.



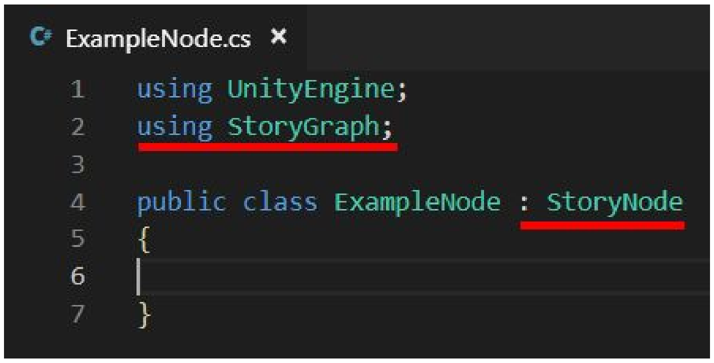
4) Here is an example of a Story Graph in the UI Demo scene. In the StoryGraph logic flows from left to right and starts where ever there is no “In Point”. Therefore, this StoryGraph starts at the blue node named “On Button Click Listener”. This is an Event node that listens for the “On Click” Event on a Canvas UI Button. Once this button is clicked in Play mode it will go to the yellow Coroutine node called “Scale”. Coroutines in Unity are used for multiframe operations such as animations. In this case it it will scale the Ball for 1.5 seconds



Scripting Quickstart

Although the StoryGraph has many nodes for uses such as UI operations, animations, and logic sequencing, you might find that you need functionality specific to your game. Thankfully, the StoryGraph API is designed to make coding minimal and easy if you want to start creating your own nodes. This is an overview of how to create a new node.

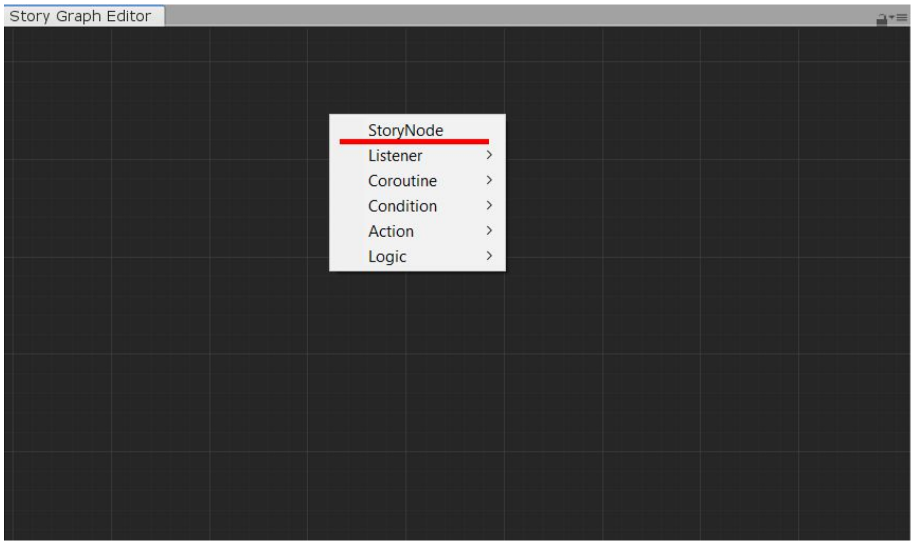
1) Create a new script in your project. I called mine ExampleNode.cs.
2) Go into your script. Make sure you are *using StoryGraph;* and inheriting *StoryNode*.



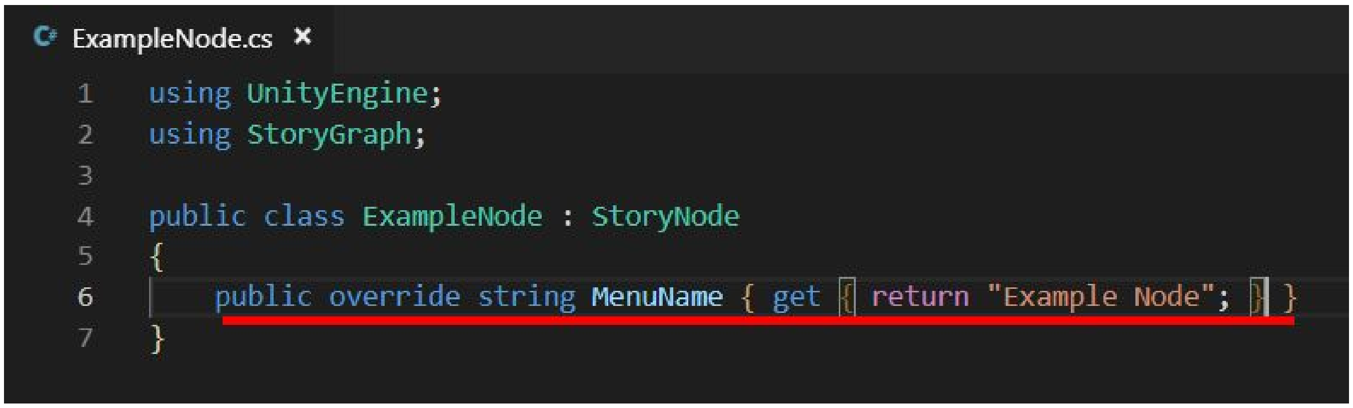
ExampleNode.cs

```
using UnityEngine;
using StoryGraph;

public class ExampleNode : StoryNode
{

}
```

3) Now when you right click inside your StoryGraph Editor you will see your script at the root of the menu called “StoryNode”. Let’s change that name.

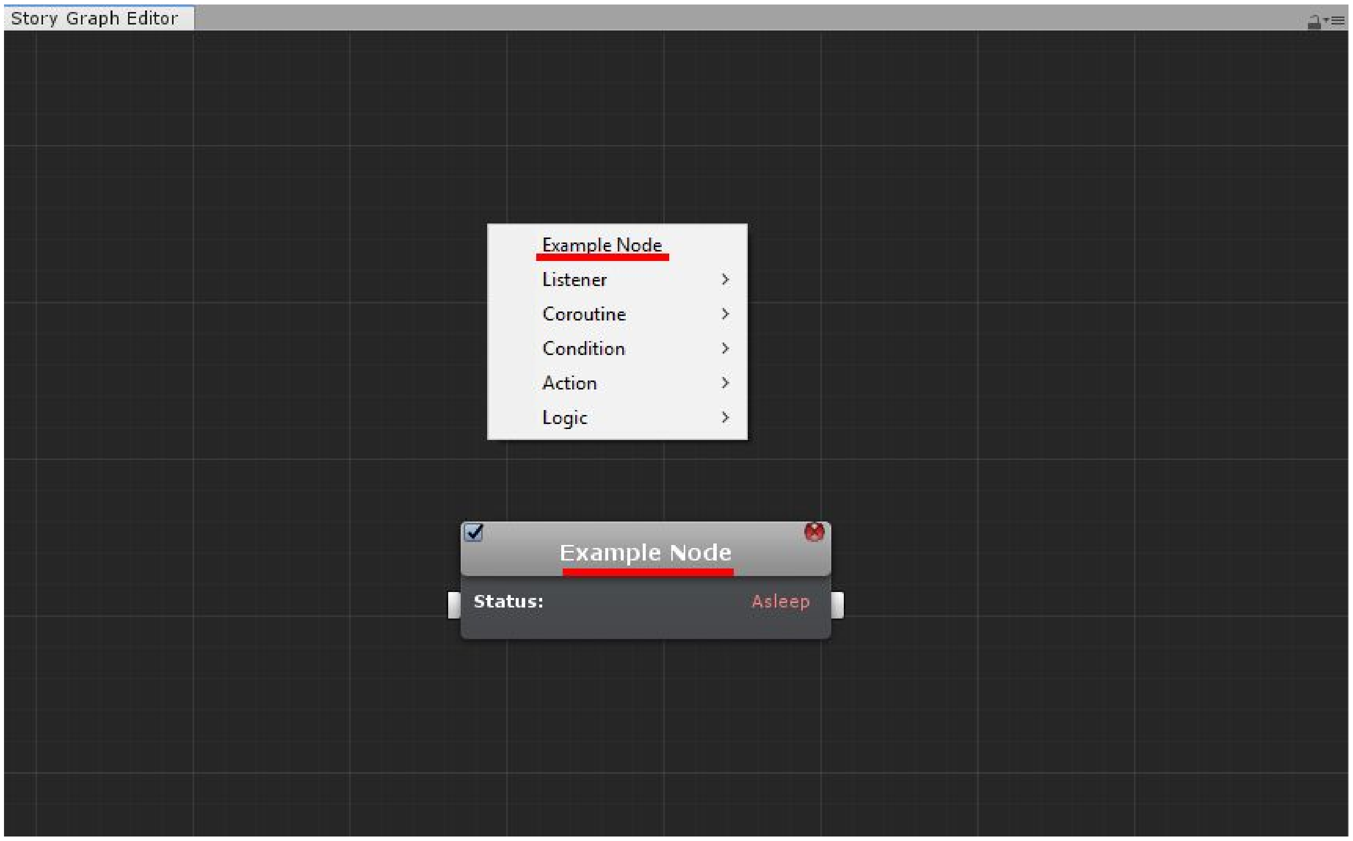


4) In the first line of your script add the following line. Where it says “Example Node” replace it with the name you want your node to be in the StoryGraph Editor



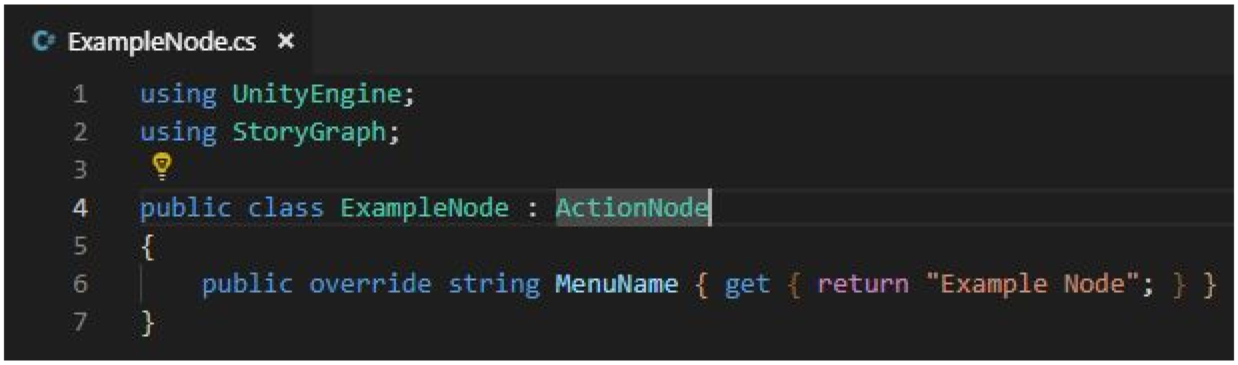
```
ExampleNode.cs
using UnityEngine;
using StoryGraph;

public class ExampleNode : StoryNode
{
    public override string MenuName { get { return "Example Node"; } }
}
```

Now if you test it in the StoryGraph Editor you will see your node's name displaying correctly. However, when you click on it, no styling will be applied to it. Let's change that.



5)

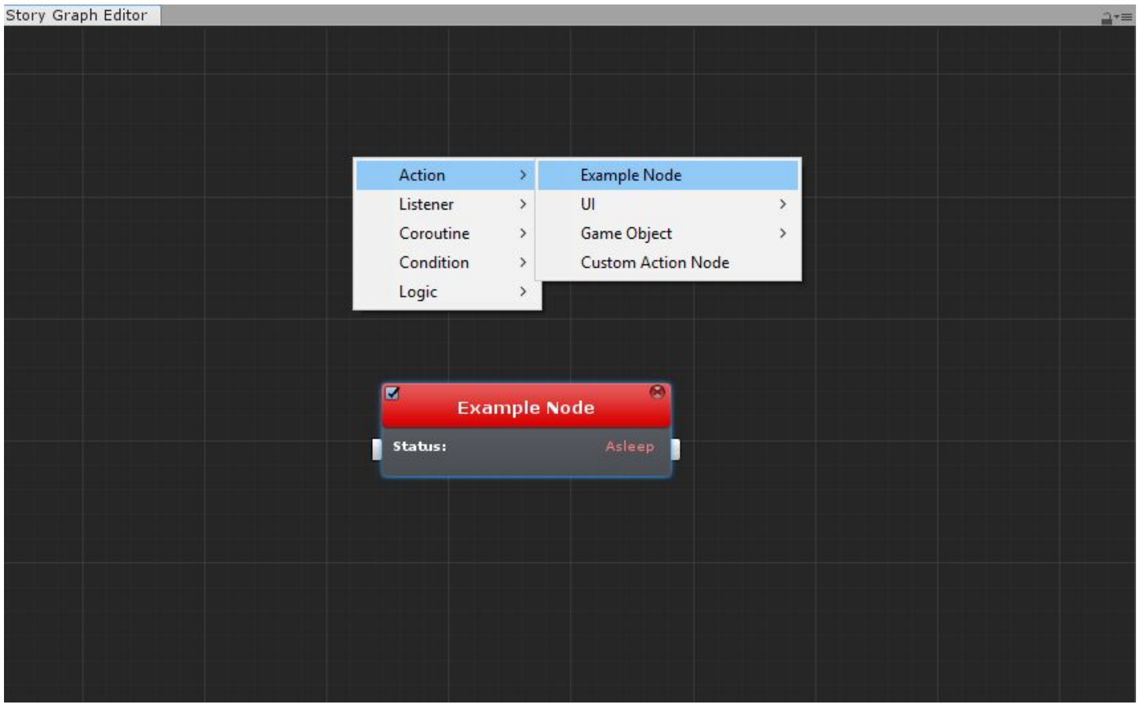
For the sake of this example, let's make our node an Action node. Go back to your script and change *StoryNode* to *ActionNode*.



ExampleNode.cs

```
using UnityEngine;
using StoryGraph;

public class ExampleNode : ActionNode
{
    public override string MenuName { get { return "Example Node"; } }
}
```

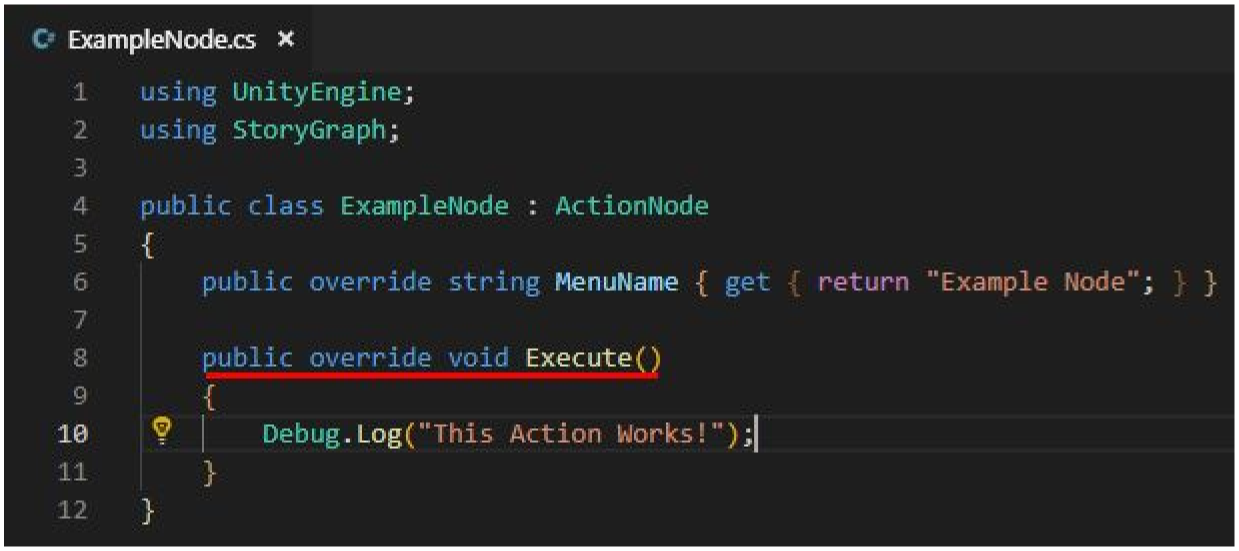
Now let's look at it in the Editor.



The Example Node is now under the Action Submenu and when you click on it the node is now red like all other actions. Let's make our Action Node actually do something.

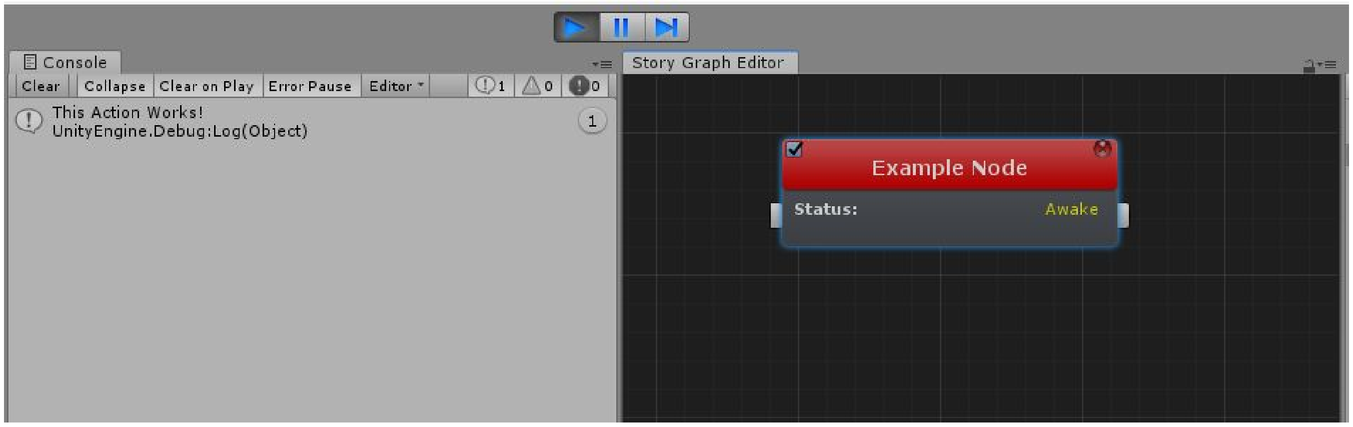
6)

In your script add a function called *Execute* like so:


ExampleNode.cs

```
using UnityEngine;
using StoryGraph;

public class ExampleNode : ActionNode
{
    public override string MenuName { get { return "Example Node"; } }

    public override void Execute()
    {
        Debug.Log("This Action Works!");
    }
}
```

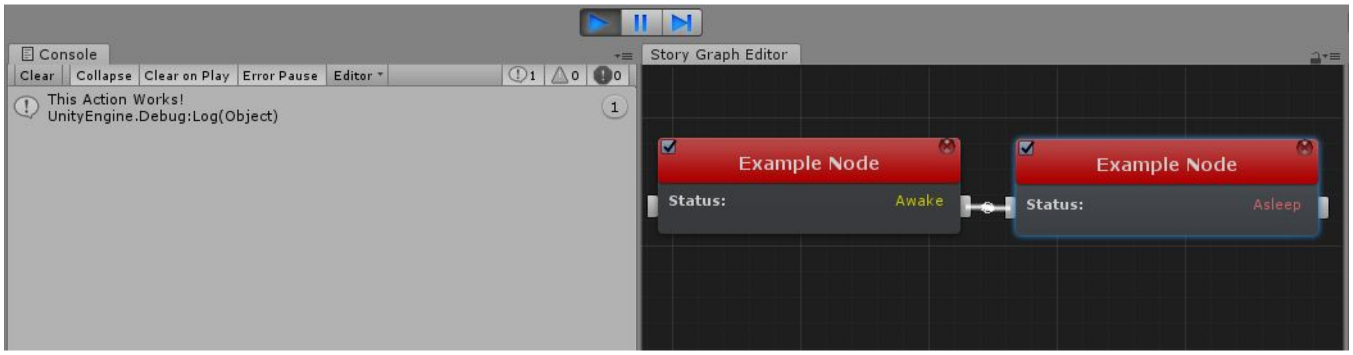
The *Execute* function is called whenever a node is awake. In this example since there are no other nodes, Example Node will run instantly in playmode. Action nodes are used for functions you want to run instantly.



```
This Action Works!
UnityEngine.Debug:Log(Object)
```

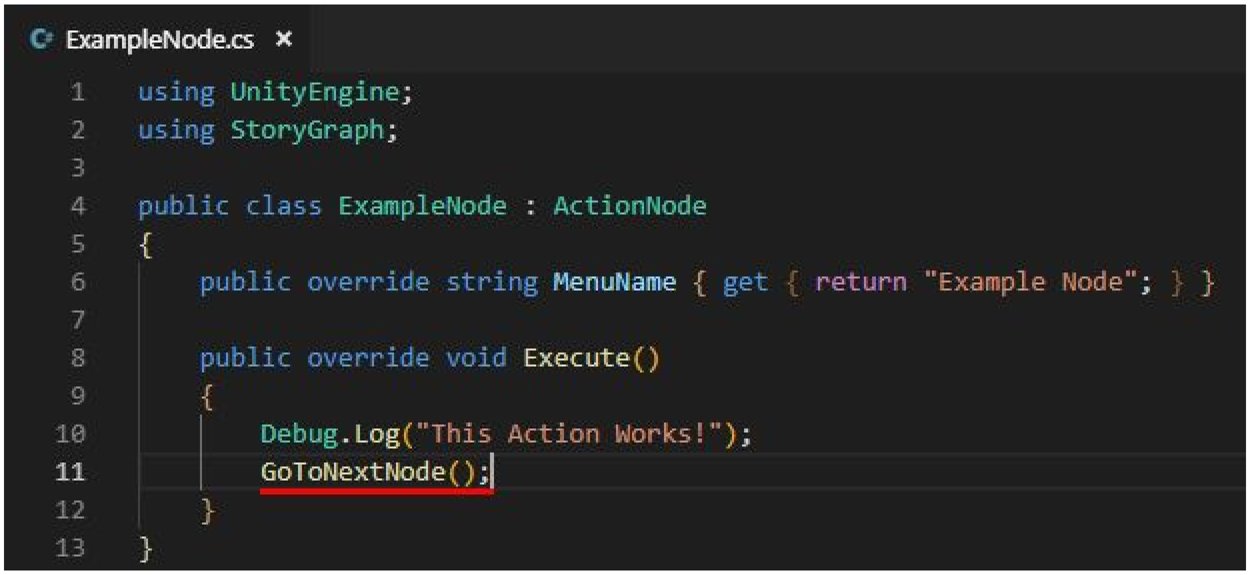
It now works!

But, what happens if you try to connect multiple nodes? It should print out a log twice, but as seen in the picture below, it doesn't reach the second node. Let's fix that!



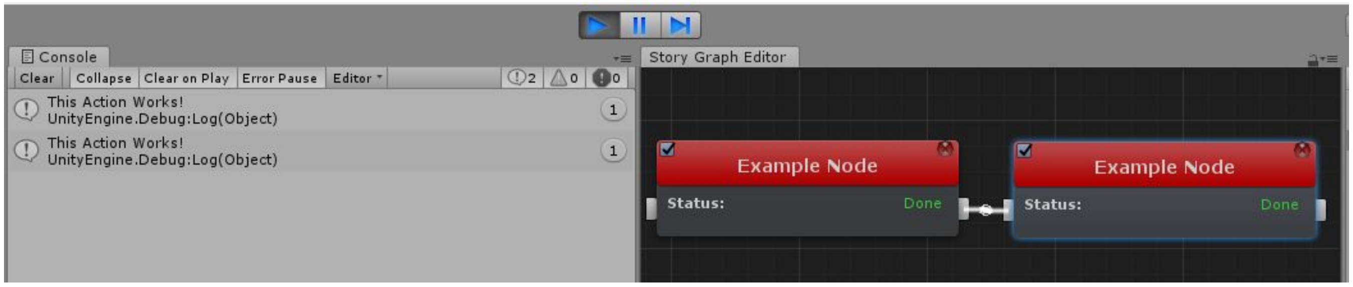
```
This Action Works!
UnityEngine.Debug:Log(Object)
```

7)
Back in your script add the function at the end of your Execute function “GoToNextNode();” When this function is called the StoryGraph will go to the next node.



```
using UnityEngine;
using StoryGraph;

public class ExampleNode : ActionNode
{
    public override string MenuName { get { return "Example Node"; } }

    public override void Execute()
    {
        Debug.Log("This Action Works!");
        GoToNextNode();
    }
}
```

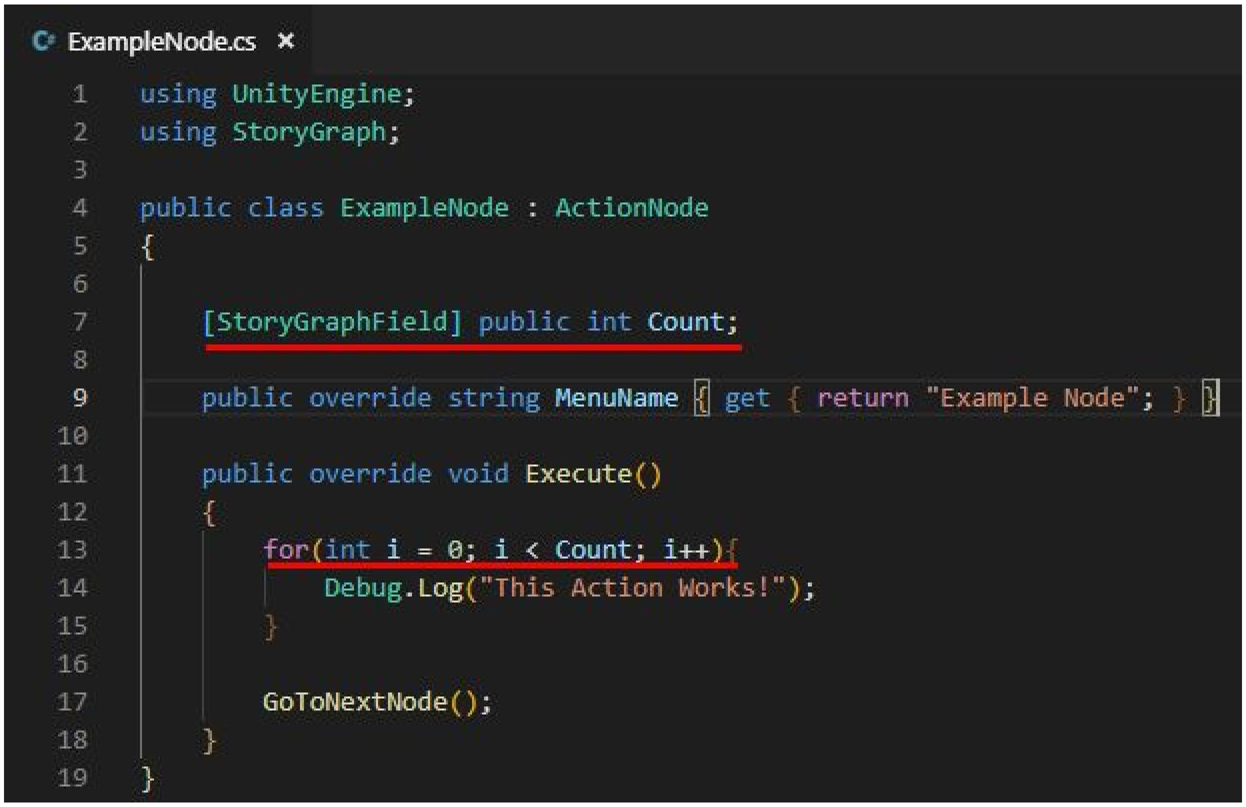
Let’s test this out:



```
This Action Works!
UnityEngine.Debug:Log(Object)
This Action Works!
UnityEngine.Debug:Log(Object)
```

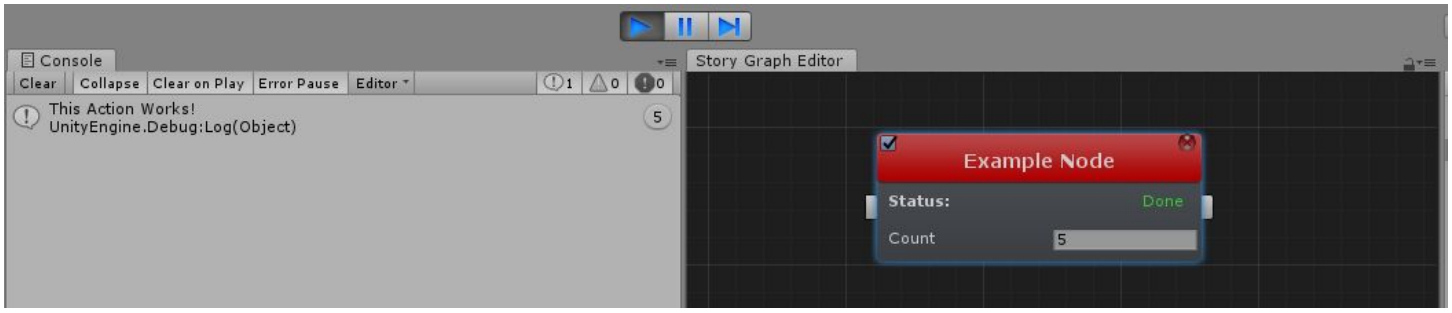
Now it’s working correctly!

8)
Our Nodes look a little empty. Let’s expose some properties.



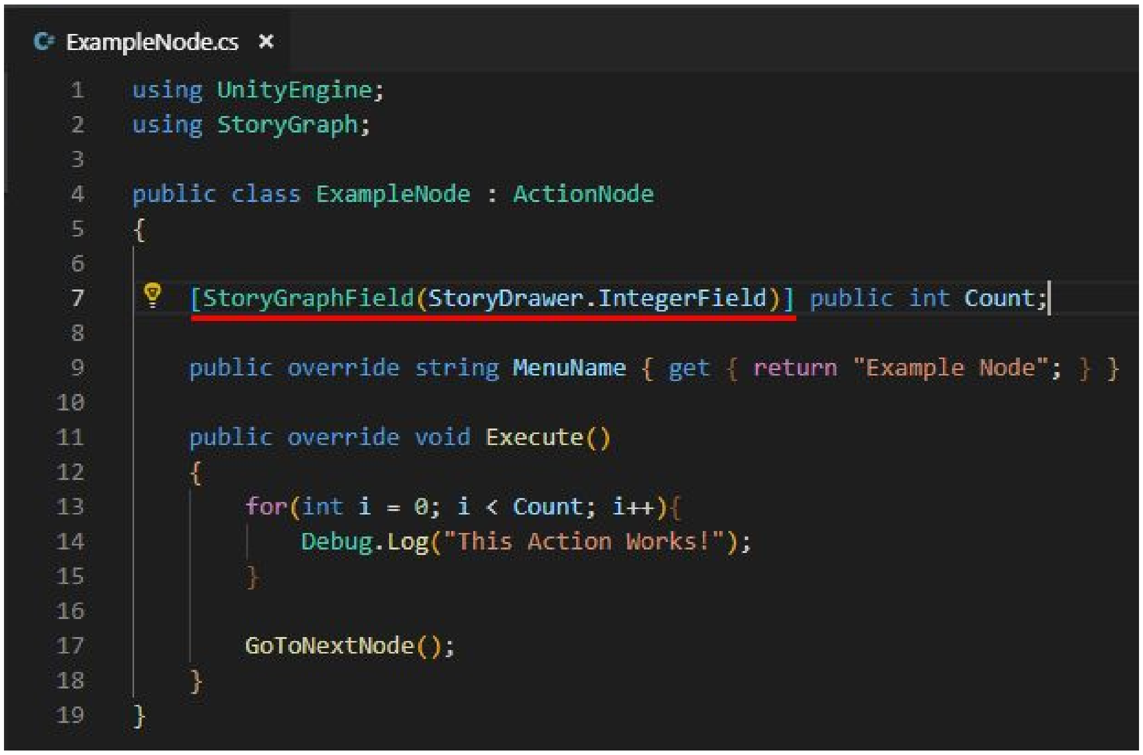
```csharp
using UnityEngine;
using StoryGraph;

public class ExampleNode : ActionNode
{

    [StoryGraphField] public int Count;

    public override string MenuName { get { return "Example Node"; } }

    public override void Execute()
    {
        for(int i = 0; i < Count; i++){
            Debug.Log("This Action Works!");
        }

        GoToNextNode();
    }
}
```

To add a field to the node just code “[StoryGraphField]” then your public variable. For the ExampleNode it will now repeat a lot for however many times we specify in the node:



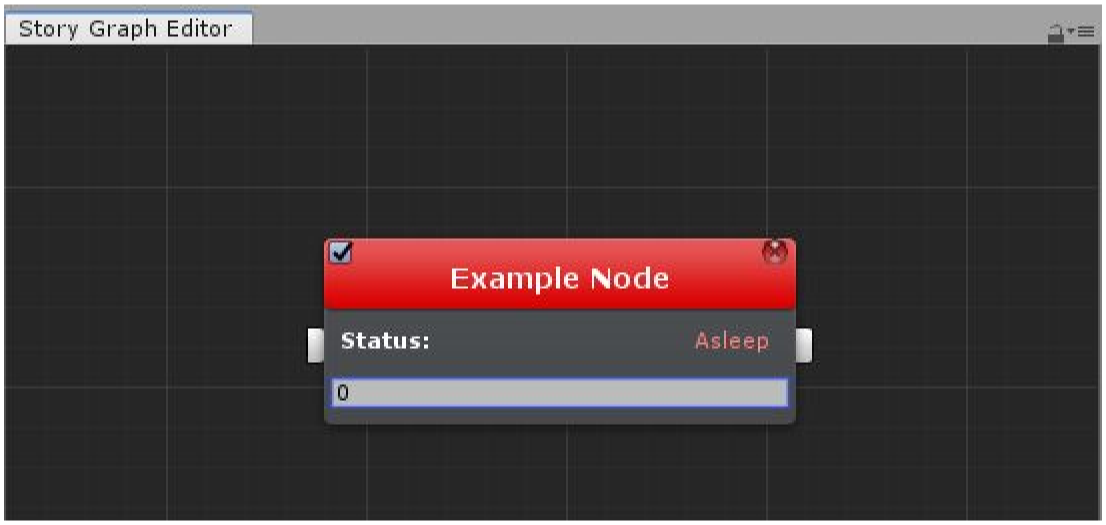
```
This Action Works!
UnityEngine.Debug:Log(Object)                5
```

Let’s say we want our variable to not display “Count” and just show the number. Just replace “[StoryGraphField” with “[StoryGraphField()]”

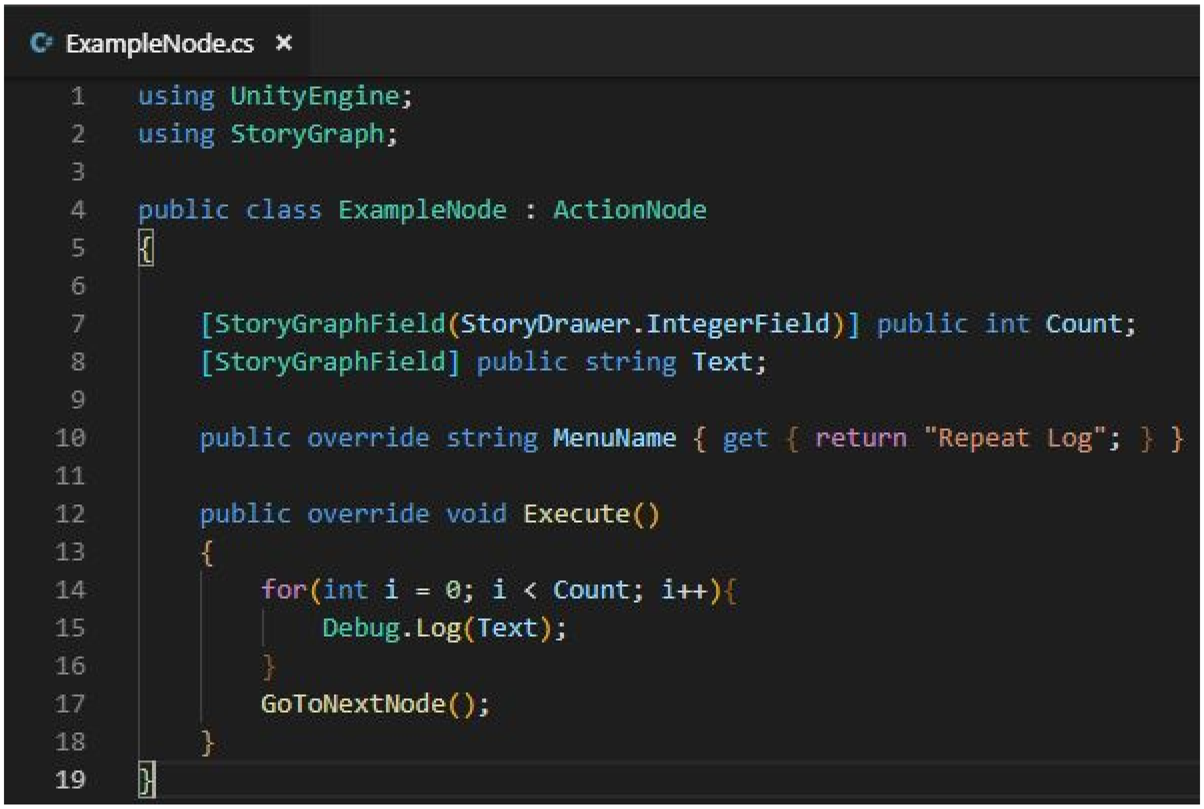

ExampleNode.cs

```
using UnityEngine;
using StoryGraph;

public class ExampleNode : ActionNode
{

    [StoryGraphField(StoryDrawer.IntegerField)] public int Count;

    public override string MenuName { get { return "Example Node"; } }

    public override void Execute()
    {
        for(int i = 0; i < Count; i++){
            Debug.Log("This Action Works!");
        }

        GoToNextNode();
    }
}
```

Now in the editor it displays only the number. This can be useful when your node understandably only needs a single integer value.

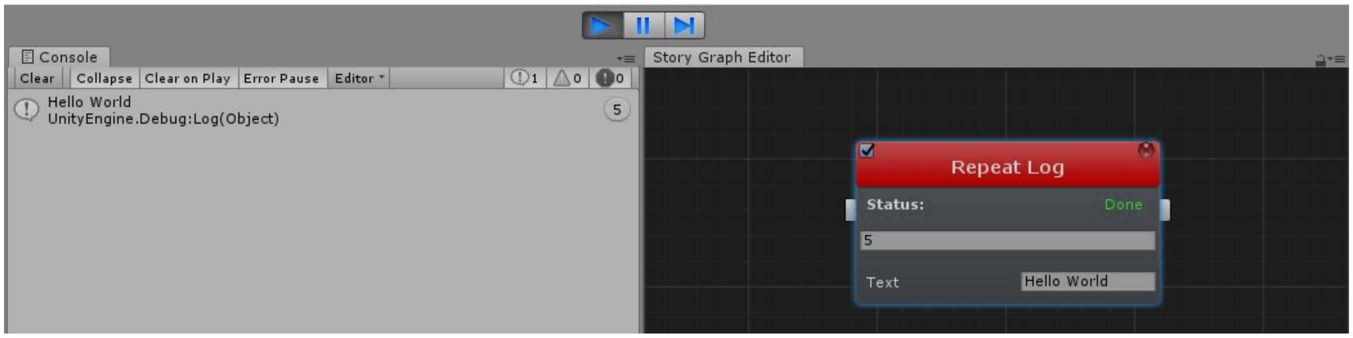


Let's rename our node again so that it's more understandable.



ExampleNode.cs

```
using UnityEngine;
using StoryGraph;

public class ExampleNode : ActionNode
{

    [StoryGraphField(StoryDrawer.IntegerField)] public int Count;
    [StoryGraphField] public string Text;

    public override string MenuName { get { return "Repeat Log"; } }

    public override void Execute()
    {
        for(int i = 0; i < Count; i++){
            Debug.Log(Text);
        }
        GoToNextNode();
    }
}
```



```
Hello World
UnityEngine.Debug:Log(Object)        5
```

Here I changed the MenuName to “Repeat Log”, and added a string Text parameter so that we can also change the text that is output in the console.

It is important to use the StoryDrawer only when it makes sense. Try to make your nodes as readable as possible so that your logic is understandable and easy to follow. Try not to overfill you nodes with too much logic. Space things out to make it as modular as possible.

Congrats, you created a Node in the StoryGraph!

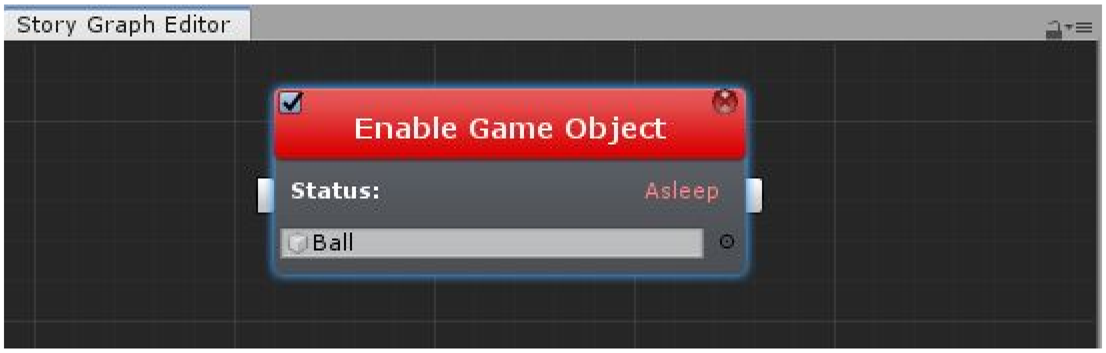
Types of Nodes

Each type of node listed are guidelines for how these nodes work, but they aren't strictly confined to these definitions. For ease of understanding, these types offer a way of categorizing how nodes are used. If you're creating a new node, when you choose which type of node to inherit, it is only reorganizing where it appears in the menu, and what color it displays.

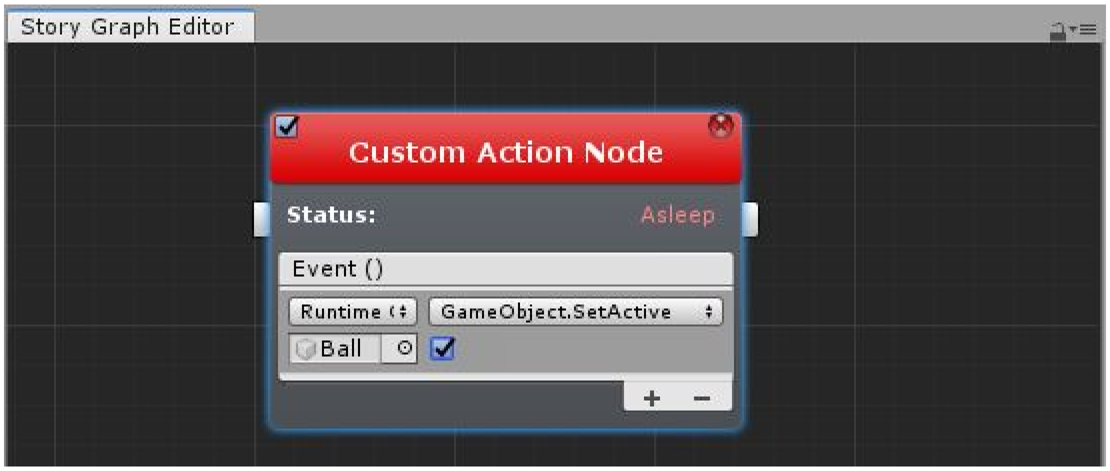
Action Nodes

When the StoryGraph goes to an action node it will be run instantaneously on the frame that it is called. This means that in play mode it will usually either be Asleep or Done. If it is Awake then the function either has an error, missing the "GoToNextNode();" function, or can't reach the "GoToNextNode()" function.

In this example the "Enable Game Object" node will enable the Ball GameObject:

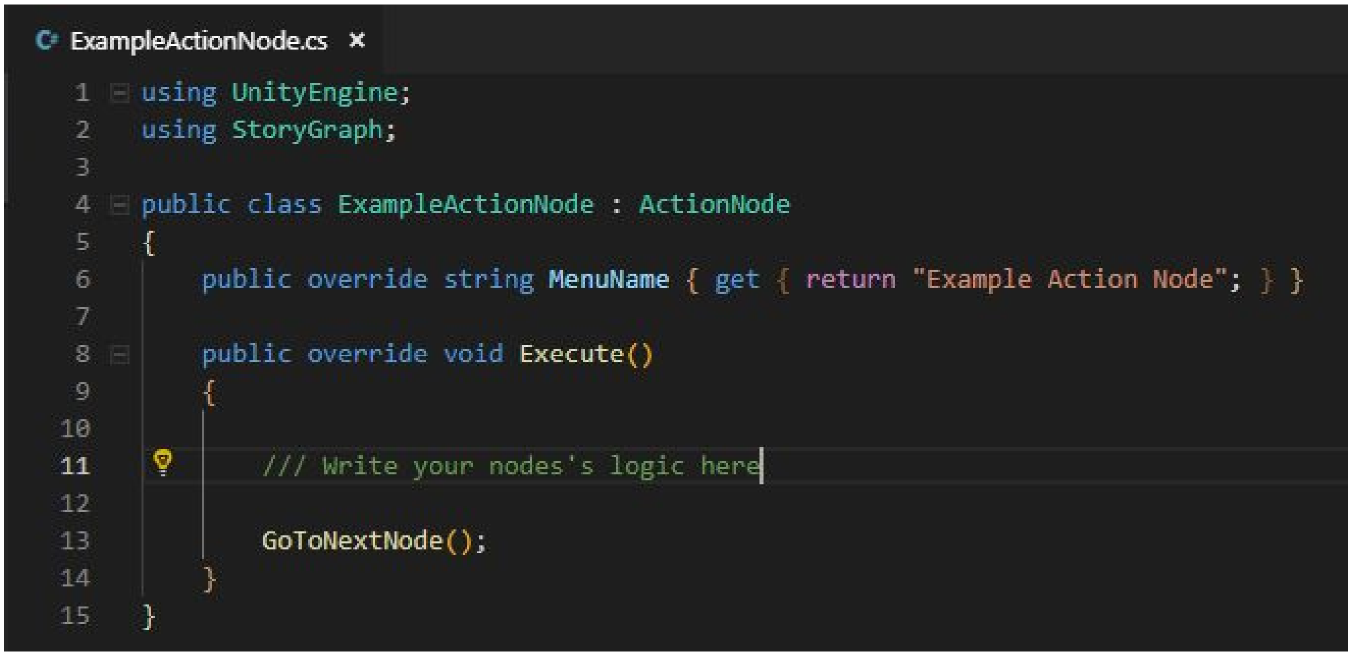


If you're in a rush, or want to use a function from an existing GameObject use the Custom Action Node. Just like the previous example, the example below will also enable the Ball GameObject:



Inside the node just press (+) at the bottom, drag in your GameObject on the left, and select which function you want to use.

When scripting an Action Node, here is the basic template of how it should look:



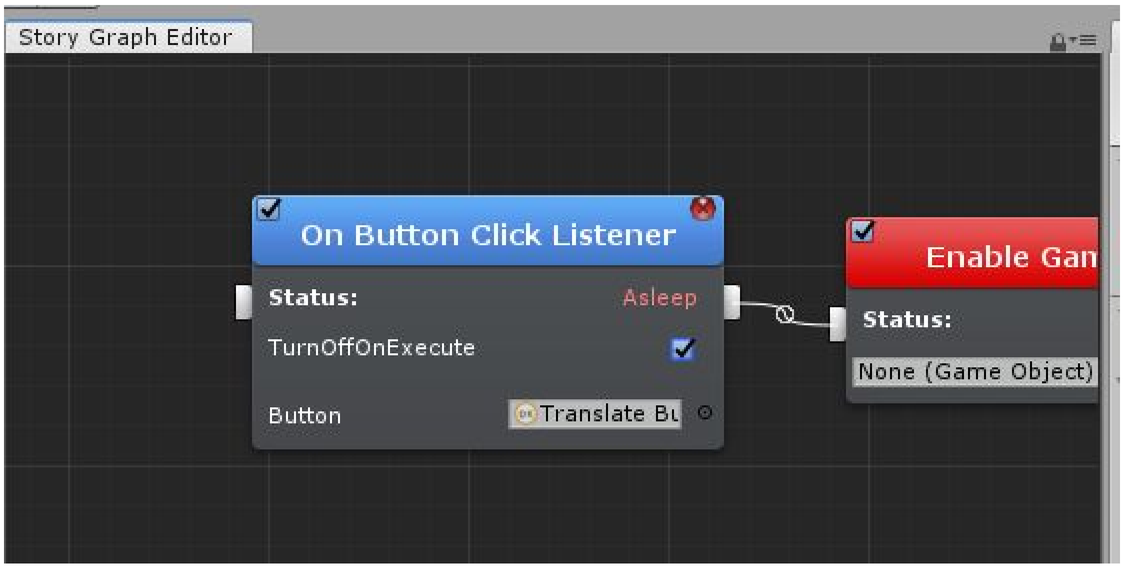
ExampleActionNode.cs

```csharp
using UnityEngine;
using StoryGraph;

public class ExampleActionNode : ActionNode
{
    public override string MenuName { get { return "Example Action Node"; } }

    public override void Execute()
    {

        /// Write your nodes's logic here

        GoToNextNode();
    }
}
```

Follow the Scripting Quick Start for a more in-depth understanding of the ActionNode.

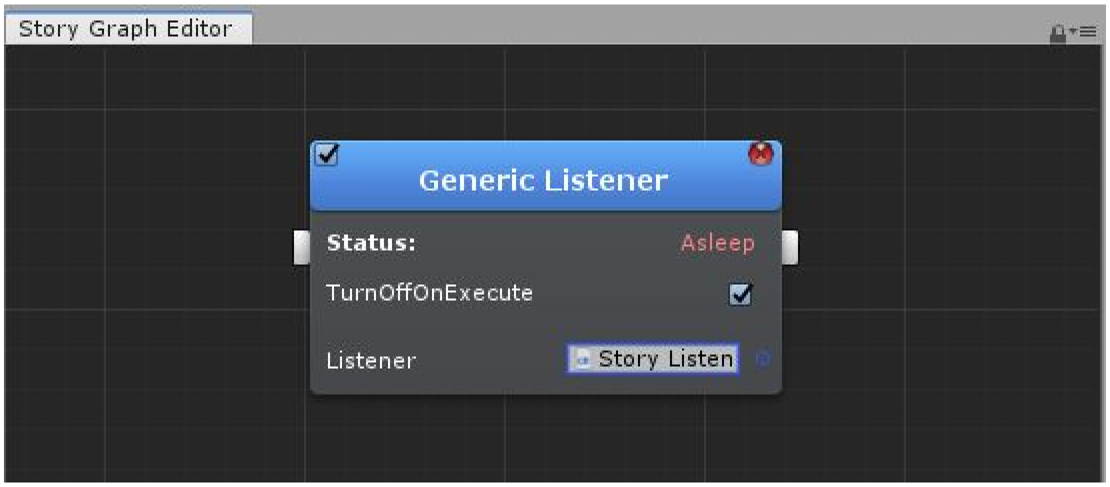
Listener Nodes

When the StoryGraph goes to a listener node it listens for an event to fire before going to the next node. This is useful for built-in Unity events such as button clicks, but it can also be very useful for listening to events from other scripts. When the StoryGraph goes to this node, it will be Awake until it listens for an event. After that it will be Done. Many of the built in listener nodes have a “TurnOffOnExecute” checkbox. When this is checked, it mean that this event will only fire once. If you want it to fire every time this event happens uncheck the “TurnOffOnExecute” parameter. Otherwise if you want it to turn on via another node, just have a node that feeds into it.

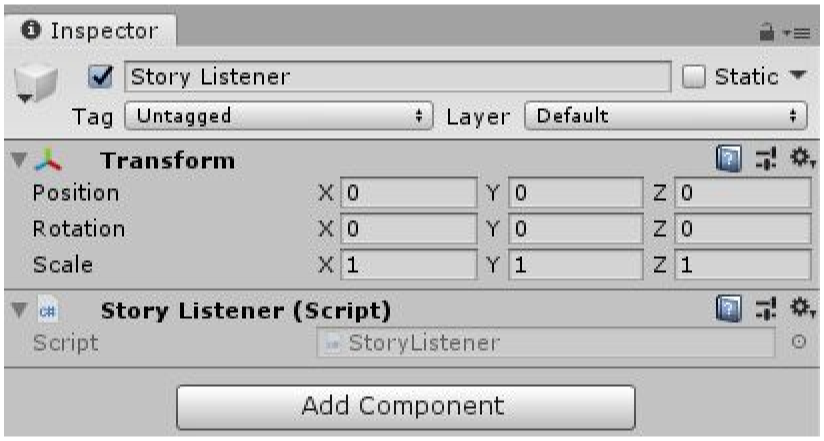


In this example when you press the Translate Button it will go to the next node.

If you want to create your own events, use the Generic Listener Node with a Story Listener on the GameObject with the event you want to track.



Status: Asleep
TurnOffOnExecute ✓
Listener: Story Listen



Story Listener ✓ — Static ☐
Tag: Untagged — Layer: Default

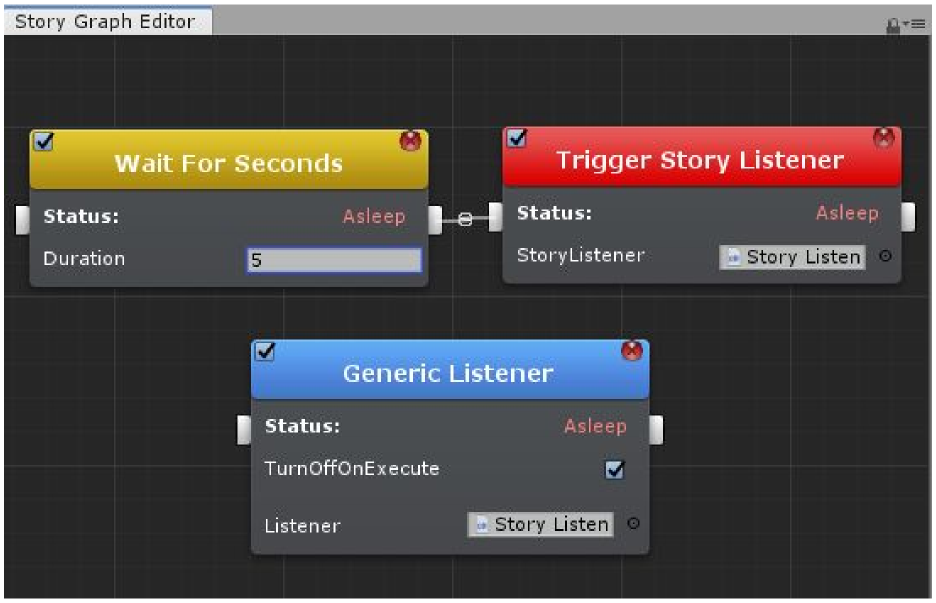
Transform

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

Story Listener (Script)
Script: StoryListener

To trigger this even you can either create a Trigger Story Listener Action Node or use this function on the StoryListener from a separate script:

```
StoryListener.StoryListenerAction.Invoke();
```



Wait For Seconds — Status: Asleep; Duration: 5

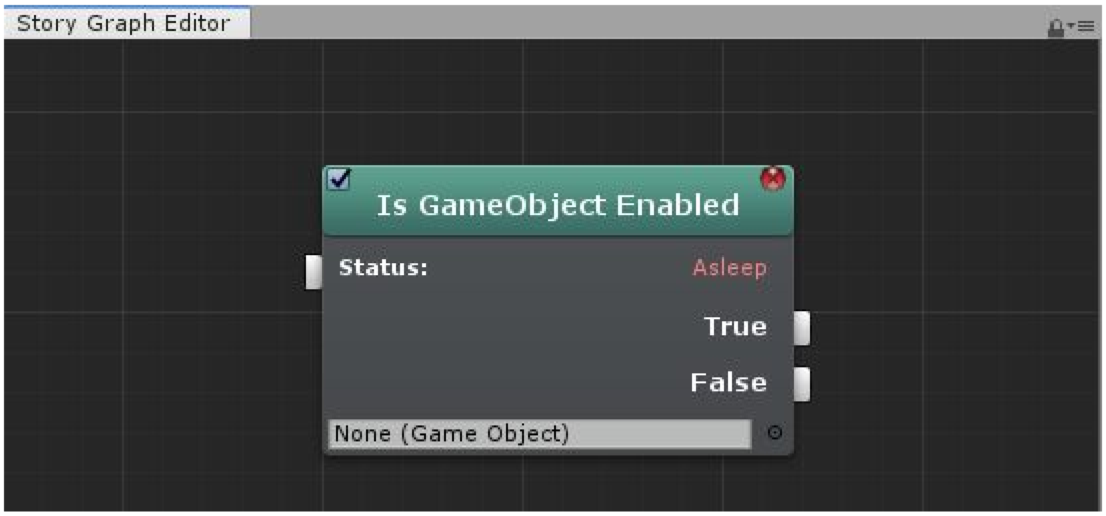
Trigger Story Listener — Status: Asleep; StoryListener: Story Listen

Generic Listener — Status: Asleep; TurnOffOnExecute ✓; Listener: Story Listen

In this example after 5 seconds it will trigger the Generic Listener.

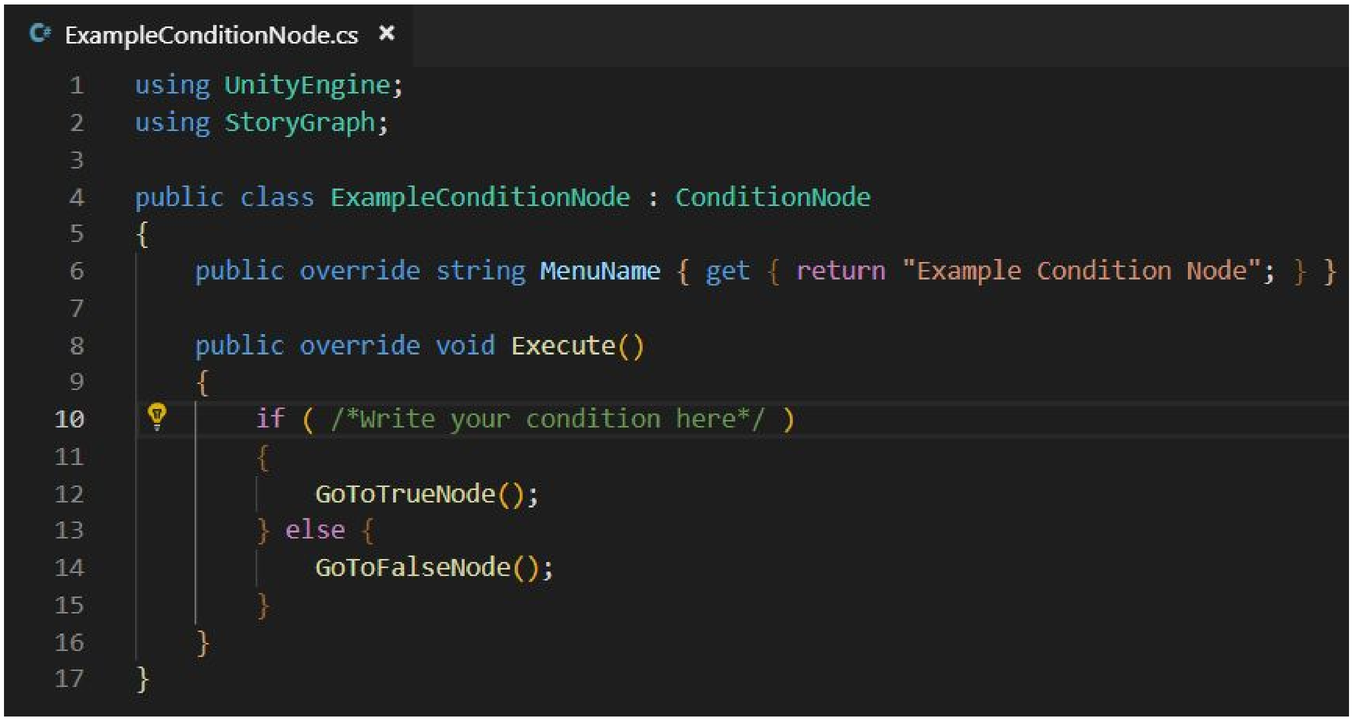
Condition Nodes

Condition Nodes work like a boolean function for your StoryGraph. Just like an Action node it runs instantaneously meaning it will either be Asleep or Done. Unlike all the other node types, Condition Nodes have an output of True or False. If the node evaluates as True, it will traverse that path, otherwise it will traverse the False path.



In this example “Is GameObject Enabled” checks to see if the Game Object placed into the first parameter is enabled or not enabled. If it is enabled, it will follow the True output. If it is False it will follow the False output.

If you want to script your own Condition Node just follow this template:



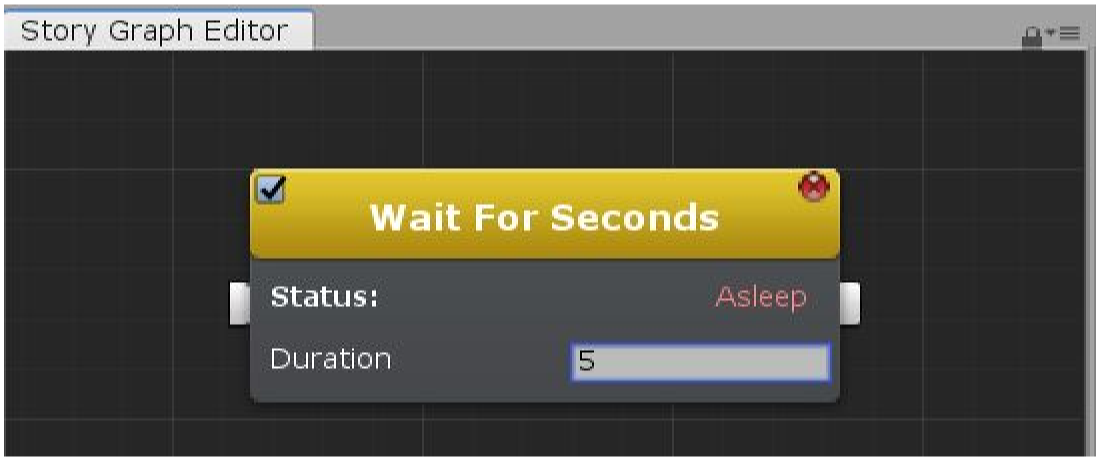
ExampleConditionNode.cs

```
using UnityEngine;
using StoryGraph;

public class ExampleConditionNode : ConditionNode
{
    public override string MenuName { get { return "Example Condition Node"; } }

    public override void Execute()
    {
        if ( /*Write your condition here*/ )
        {
            GoToTrueNode();
        } else {
            GoToFalseNode();
        }
    }
}
```

Just write your condition inside the if statement, then use the GoToTrueNode() and GoToFalseNode() respectively.

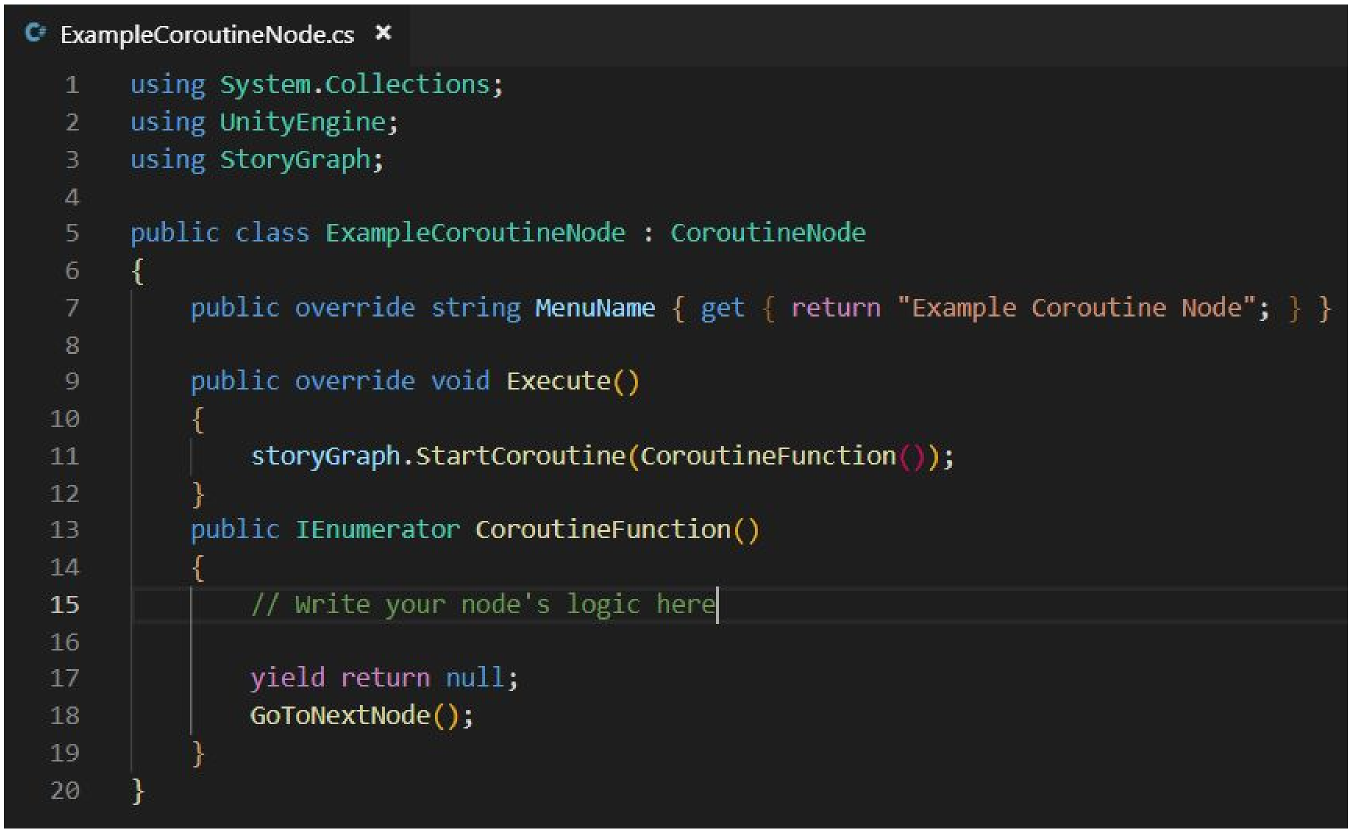
Coroutine Nodes

Coroutine Nodes are nodes that use Unity Coroutines. A Unity Coroutine is a function that is run over multiple frames. There are many uses for coroutines, but a popular one is for animation since it takes multiple frames. Coroutine Nodes are Awake until it has completed the Coroutine, which it will then switch to Done and go to the next node.



In this example, the Wait For Seconds node will wait for 5 seconds before going to the next node.

To code your own Coroutine Node follow this template:



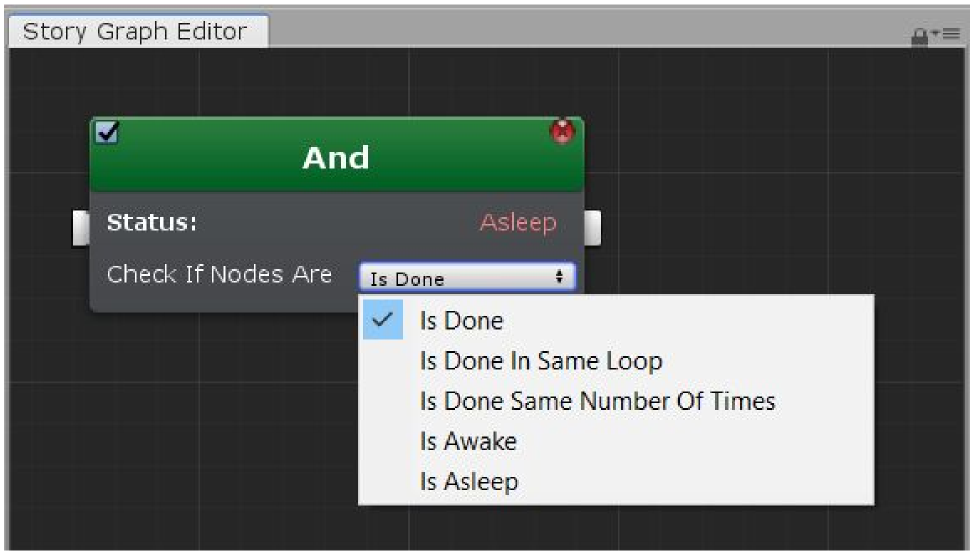
```
using System.Collections;
using UnityEngine;
using StoryGraph;

public class ExampleCoroutineNode : CoroutineNode
{
    public override string MenuName { get { return "Example Coroutine Node"; } }

    public override void Execute()
    {
        storyGraph.StartCoroutine(CoroutineFunction());
    }
    public IEnumerator CoroutineFunction()
    {
        // Write your node's logic here

        yield return null;
        GoToNextNode();
    }
}
```

Be sure to include “using System.Collections;” so that it can use the IEnumerator class. In this template feel free to replace the “CoroutineFunction” with any coroutine function, but make sure that it uses the GoToNextNode() function at the end of your coroutine.

If you’re still unsure of how Coroutines work, consult the [Unity Coroutine Documentation]

Logic Nodes

Logic Nodes are used for the logic gates “Or” and “And”. Logic gates are useful when you want to make sure multiple nodes are the same state. When you feed multiple nodes into a logic gate, if it doesn’t meet the conditions of the logic gate it will be Awake. Once it does meet the conditions of the logic gate it will be Done.



In this example, as you can see there are 5 states to choose from. Here is what they mean:

Is Done:	Checks to see if all nodes fed into the And Node are Done. If all nodes are Done multiple times, this will still run
Is Done In Same Loop:	If your StoryGraph has a loop, this state checks to see if all the nodes are done at the same time
Is Done In Same Number of Times:	If your StoryGraph has a loop, it will display a number of times the node has run. If it's all the same number of times, this node will pass.
Is Awake:	Checks if all the other nodes attached are Awake.
Is Asleep:	Checks if all the other nodes attached are Asleep.

For all intents and purposes stick with "Is Done". Logic Nodes will be expanded upon to make the workflow easier in future versions of the StoryGraph.