Closed Beta Quick Start Guide

v1.6

Detox Studios

Table of Contents

Disclaimer

Please be aware that this is beta software! **You should not rely on this software for actual production work at this time.** As we update the software through the beta process, you should expect to potentially lose work you have done with uScript.

We also do not recommend adding uScript to an existing project you may currently be working on without first backing that project up BEFORE installing uScript.

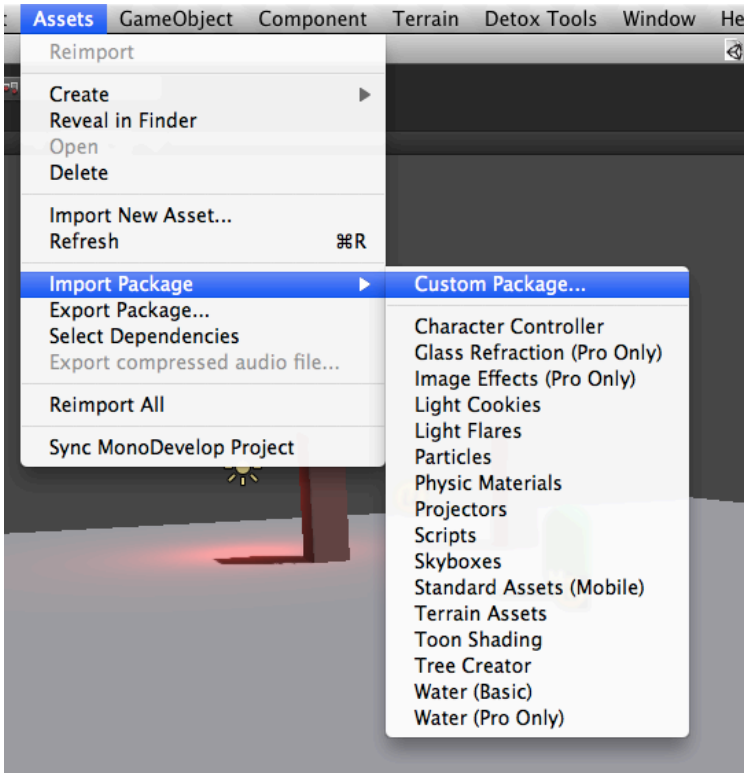
uScript has been designed for Unity 3 and will not work in earlier versions.

Installation

The uScript Visual Scripting Tool is an editor tool for version 3 of the Unity engine. Please follow these steps to install uScript

1. Download and unzip the ***uScript_ClosedBeta.zip*** file to a preferred location on your computer.

2. Run Unity 3 and select Import Package/Custom Package from the "Assets" menu and browse to the ***uScript.unitypackage*** file to install uScript:



3. When the package window comes up, make sure everything is selected and choose "Import".

4. Once uScript has installed, you should now have a uScript folder in the root of your Unity projects Assets folder. You can run the tool by going up to the "Tools" menu in Unity and selecting "uScript Editor" or by pressing Control+U (or Command+U on Mac).

a. *Note: there is a bug in Unity that may prevent this menu from showing up initially. Just click on any other menu to force Unity to refresh its menus.*

5. The first time you run uScript, it will create an "uScript Master GameObject" in your scene. This is needed for uScript to run. You will see this GameObject's uScript Gizmo icon appear in your scene:

uScript Key Concepts

The following are key concepts regarding uScript that will help you understand how uScript works and to make the most out of it. Many of these things are requirements to work with the way Unity does things.

uScript Master GameObject

uScript needs to make a GameObject for itself that contains some key uScript components. The name for this GameObject is '_uScript'. For your uScript to work in your scene, it must be assigned to a GameObject! If it was not assigned automatically by uScript when you first saved your file, you can drag the C# script (NOT the one with *_Nested* in the name) onto this GameObject in your Unity scene.

Output Files

uScript currently generates three files for each uScript "graph" you create. Let's assume you create a new uScript called "MyGreatGame"—these are the files uScript would generate:

- **MyGreatGame.uscript** – This is the "master" binary file that uScript uses to create your uScript. You don't want to delete this file (unless you want to actually delete the uScript permanently). This file can generate/re-generate the other two files below. Sometimes you will want to do this when uScript (or a node) has been updated and you receive warnings or errors.

- **MyGreatGame.cs** – This is a "wrapper" output C# script file. This is the file that would be assigned to the uScript master GameObject (see above) in order to have your uScript run in the scene. This file is automatically built/rebuilt by uScript from the .uscript binary file above.

- **MyGreatGame_Nested.cs** – This file is where all the "magic" is. This is the file that contains your entire uScript graph as pure C# script. This file can also be used in uScript as a Subsequence like in Unreal's Kismet. These files are actually more powerful/flexible than a Subsequence and a better term (and one we use) would be "nested uScripts". This file is automatically built/rebuilt by uScript from the .uscript binary file mentioned above.

Uniquely Named GameObjects

Currently uScript needs to be able to tell which specific GameObject you want it to use by its name. This means that you should have a unique name for any GameObject you wish to use or reference in uScript, as that is the only way uScript can identify a specific object between Unity sessions (a session means quitting and then running Unity again). While Unity does support unique IDs internally for all GameObjects in your scenes, unfortunately these unique IDs are not persistent. This means that we cannot rely on the GameObjects to have the same ID numbers in between closing and reopening Unity, leaving the only choice for the user to make sure they uniquely name their GameObjects they wish to use in uScript.

uScript Event Components

Some event types are triggered by a specific object and so a specific uScript component needs to be added to that object. As an example, if you plan to use a GameObject as a trigger and you want to fire off uScript logic based on that trigger, you need to make sure that GameObject has a ***uScript_Triggers.cs*** component assigned to it. In fact, if you try to assign a GameObject as an instance (see below) of a Trigger event in uScript, it will not let you assign it if the uScript_Triggers.cs script component has not been assigned to that GameObject.

More global-based event components are just assigned to the uScript master GameObject—like for OnGameStart (*uScript_Global.cs*) and OnKeyPress (*uScript_Input.cs*) event nodes.

Instances

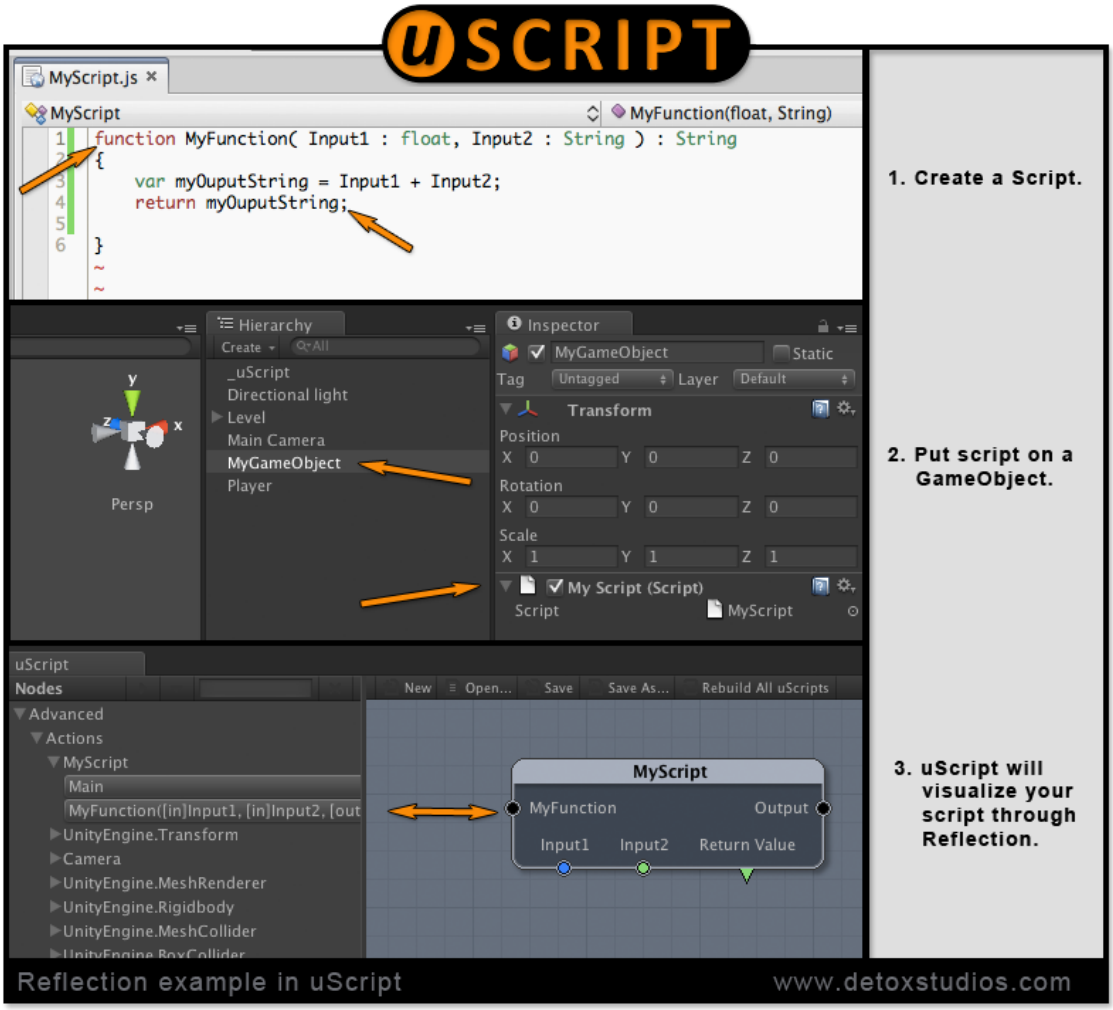
Many of uScript's events require you to sign a specific GameObject to it so uScript knows what object you want to fire that event. Using the Trigger example again, if you have placed a Trigger event node, you need to tell uScript what trigger specifically you want to make that event fire (you could have hundreds in your

scene!). This is done by assigning an instance (specific GameObject) to the "Instance" property of Event Nodes. Again, as mentioned above, that GameObject will also need to have the proper component assigned to it so uScript will be able to listen for when the event occurred (or think of it as the GameObject telling uScript when something relevant has happened to it).

Many things in reflection (see below) also require you to assign an instance in order to tell uScript exactly what GameObject you wish to use.

Reflection

A feature for the more technical, uScript will use dotnet reflection in order to visualize many aspects of your existing scripts in Unity. It is important to note though that uScript can only see scripts that are assigned to GameObjects/active in the scene. uScript will reflect any public Actions (methods/functions), Properties (Properties/Public Variables exposed to the Unity inspector – both Get and Set), and Variables (all variable types). You must assign an instance (see above) to them so uScript knows what GameObject you want to use.



Biology of a Node

Nodes are made up of a basic set of elements that perform a specific function. These elements and their purpose are explained below.

Border Color

Nodes have a specific border color in order to visually identify them at a glance. Currently there are two colors:

- **Orange** – Event Nodes
- **Grey** – Action Nodes

Sockets

At the edges of each node, there are connection sockets that are used by the node in three main ways.

Socket Rules

These simple rules are consistent between all node sockets:

1. Input sockets are always on the left side of a node.
    a. Event nodes do not have input sockets.
2. Output sockets are always on the right side of a node.
3. There can be multiple Input and Output sockets on a node.
4. Variable sockets are always on the bottom of a node.
    a. There can be multiple variable sockets on the bottom of a node.
    b. The most popular variable socket types are color coded, otherwise they will have a white center.
    c. Variable sockets are either Read-only (get) or Write-only (set).
    d. Read sockets are round like other sockets.
    e. Write sockets are triangle shapes pointing "out" from the node.
    f. Not all variable sockets are exposed by default.
    g. You can hide or show variable sockets through the checkbox on the properties panel.
        i. Variable sockets in use can not be hidden

Known Issues

External Connections

Currently External Connections cannot be moved by selecting them. We are working on this system right now and will have a fix in a future build. A workaround for now that you can use is to create a temp variable next to the External Variable you want to move, then box/marquee select both and click on the variable to drag both to the new location.

Uniquely Named GameObjects

uScript uses the name of GameObjects in order to find them. We cannot rely on Unity's unique ID number system, as Unity itself does not keep this information once Unity is closed—it will assign new unique IDs the next time you run the editor. This means that uScript cannot store unique IDs for GameObjects in it's own save files—they would be useless the next time you run Unity.

This means you should name GameObjects something unique if you plan to use them/reference them with uScript to ensure uScript is using the correct GameObject. Unfortunately we will not be able to fix this issue until the Unity editor itself supports maintaining unique IDs between Unity sessions.

Missing Event Nodes Section

When you first run uScript, the "Events" section may not be available. If this happens, just close uScript and re-open it.

Missing Detox Tools Menu

When first installing uScript, the "Tools" menu in Unity may not appear. Just click on any other menu item to force Unity to update its menus and you should then see the "Tools" menu appear.

Box Connection Lines

You cannot select just connection lines by using the box select method.

Hot Keys

Note: On Mac, replace Control (CTRL) with the Command (CMD) key.

Action	Key	Alt Key	Mouse	Alt Mouse	Context
General					
Cut	CTRL + X				
Copy	CTRL + C				
Paste	CTRL + V				
Undo	CTRL + Z				
Redo	CTRL + Y				
Close uScript	CTRL + W				

Action	Key	Alt Key	Mouse	Alt Mouse	Context
Canvas					
Full screen canvas	SPACE				
Delete selected nodes	DELETE	BACKSPACE			
Select all nodes	CTRL + A				
Center canvas on next Event node	]				
Center canvas on previous Event node	[				
Resets canvas to center (default position)	HOME	CTRL+H			
Deselect all	ESC		LMB		C
Toggle Grid	CTRL + G				
Place String variable where clicked			S + LMB		C
Place Vector3 variable where clicked			V + LMB		C
Place Int variable where clicked			I + LMB		C
Place Float variable where clicked			F + LMB		C
Place Bool variable where clicked			B + LMB		C
Place GameObject variable where clicked			G + LMB		C
Place Object variable where clicked			O + LMB		C
Place Comment node where clicked			C + LMB		C
Place External Connection where clicked			E + LMB		C
Place Log node where clicked			L + LMB		C
New node selection			LMB		N
Toggle node selection			SHIFT + LMB		N
Marquee - New node selection			LMB + Drag		C
Marquee - Add to selection			SHIFT + LMB + Drag		C
Marquee - Remove from selection			CTRL + LMB + Drag		C
Move node			LMB + Drag		N
Pan canvas			ALT + LMB + Drag	MMB + Drag	C, N
Context Menu			RMB		C, N

Context - What the mouse is over when the action occurs. (*N = node, C = Canvas*)

Setting Up Your First uScript

Coming Soon! For now we recommend you watch the videos we have posted showing the use of uScript.

YouTube: http://www.youtube.com/user/uScriptTool

Screencast: http://www.screencast.com/users/uScript/folders/uScript/

Creating Your First Action Node

Writing a custom node is relatively straightforward.

At its core you need to do only one thing:
1. Derive your class from uScriptLogic

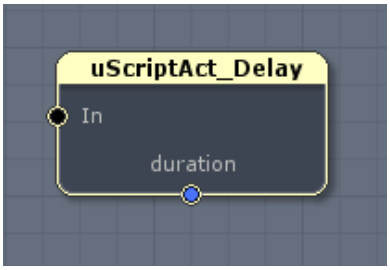
Now to get more specific: uScript looks at all the uScriptLogic classes and interprets them as visual nodes. Any public methods you have will be placed as sockets on the right side of the node. There is a limitation to be aware of: Only 1 parameter signature is allowed. This means all public methods of the same node must have the same argument signature.

For our example we will create a Delay node. This node will accept a number of seconds (for a countdown), fire off an immediate output and then fire off a signal when the count down reaches 0.

Let's create our stub:

```
using UnityEngine;

public class uScriptAct_Delay : uScriptLogic
{
   public void In( float duration )
   {
   }
}
```

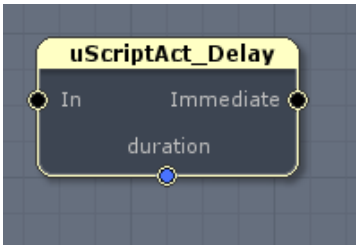
This code will be represented as a node with a single input, "In".



The first thing we want to do is to allow the graph to continue executing after they've called our In. uScript allows continual execution through public properties. Each public property you have will show up as an output socket on the right side of the node. At run time uScript will query that property to see if it can continue execution on that branch. For our Delay node we want execution to continue immediately so we add a public property which always returns true.

```
public class uScriptAct_Delay : uScriptLogic
{
   public bool Immediate { get { return true; } }

   public void In( float duration )
   {
   }
}
```

Now our node will have an In socket on the left and an Immediate socket on the right.



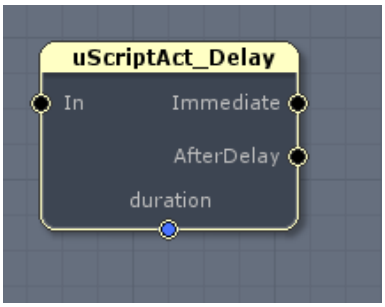
Next we want to fire an event when our countdown has reached 0. To do this we add an event handler and a public event.

```
public class uScriptAct_Delay : uScriptLogic
{
   public delegate void uScriptEventHandler(object sender, System.EventArgs
args);
   public event uScriptEventHandler AfterDelay;

   public bool Immediate { get { return true; } }

   public void In( float duration )
   {
```

```
    }
}
```

Please note, the current version of the uScriptLogic class does not allow a custom event argument, you must use System.EventArgs.

Now our node will have an In socket on the left, an Immediate socket on the right and an AfterDelay socket on the right.



Let's add the code which fires this AfterDelay event when the countdown reaches 0.

```
public class uScriptAct_Delay : uScriptLogic
{
   public delegate void uScriptEventHandler(object sender, System.EventArgs
args);
   public event uScriptEventHandler AfterDelay;

   public bool Immediate { get { return true; } }
   private float m_TimeToTrigger;

   public void In( float duration )
   {
      m_TimeToTrigger = duration;
   }

   public override void Update( )
   {
      if ( m_TimeToTrigger > 0 )
      {
         m_TimeToTrigger -= UnityEngine.Time.deltaTime;

         if ( m_TimeToTrigger <= 0 )
         {
            if ( AfterDelay != null ) AfterDelay( this, new
System.EventArgs());
         }
      }
   }
}
```

Notice we added an Update method. This method is called every tick by uScript. The current list of automatically called functions for a uScriptLogic node are as follows:

- Update
- LateUpdate
- FixedUpdate
- OnGUI

These are called by Unity, which falls through to your custom node.

Congratulations! You've written your first uScript node. Now let's pretty it up with some attributes to help your end user take full advantage of your node.

There are multiple class attributes available to you. Below I've added them to our Delay node, and I will follow with a brief explanation:

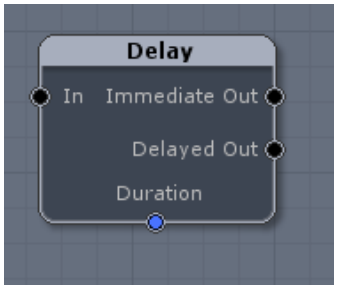
```
[NodePath("Action/Misc")]
[NodeLicense("http://www.detoxstudios.com/legal/eula.html")]
[NodeCopyright("Copyright 2011 by Detox Studios LLC")]
[NodeToolTip( "Delays execution of a script.")]
[NodeDescription("Delays execution of a script but can also fire off an
immediate response.")]
[NodeAuthor("Detox Studios LLC", "http://www.detoxstudios.com")]
[NodeHelp("http://uscript.net/manual/node_delay.html")]

public class uScriptAct_Delay : uScriptLogic
{
…
```

- ***NodePath*** is where this node will show up in the menu options.
- ***NodeLicense*** and ***NodeCopyright*** aren't currently used but will be leveraged in the future to display important licensing information pertaining to your node.
- ***NodeToolTip*** isn't currently used, but will be for tool tips pertaining to your node.
- ***NodeDescription*** shows a brief description in the uScript Reference Panel.
- ***NodeAuthor*** isn't currently used but will be shown in our uScript Reference Panel.
- ***NodeHelp*** is used by the Online Reference button in the uScript Reference Panel to open a web link.

Finally this brings us to the ***FriendlyName*** attribute. This attribute can be used with any of your parameters, properties and class names to give 'user friendly' text to your node. Anywhere you use a FriendlyName attribute, uScript will replace the parameter name with your FriendlyName text.

We can take full advantage of this with our Delay node.

```
[FriendlyName("Delay")]
public class uScriptAct_Delay : uScriptLogic
{
   public delegate void uScriptEventHandler(object sender, System.EventArgs
args);
   private float m_TimeToTrigger;

   [FriendlyName("Immediate Out")]
   public bool Immediate { get { return true; } }

   [FriendlyName("Delayed Out")]
   public event uScriptEventHandler AfterDelay;

   [FriendlyName("In")]
   public void In(
      [FriendlyName("Duration")] float duration
   )
...
```



Congratulations! You have just made your first uScript node more user friendly.

Please feel free to look at our nodes which ship with uScript for further ideas on how to extend uScript's functionality.

Glossary of Terms

uScript uses the following terminology:

Node – Sometimes referred to a logic blocks, they are visual block on the canvas that has input and output sockets and, optionally, bottom variable sockets. There are two major types of nodes.

Event Node – An orange node that is used to fire off based on a specific event happening (like on game start, a key is pressed, or when a trigger is entered). These nodes are always the start of a chain of node logic and do not have an input socket like other nodes.

Action Node – Also known as a logic node, these nodes perform actions/logic when fired. They usually have bottom variable sockets for inputting or outputting variables. They may also have multiple input and output sockets depending on the nodes purpose.

Socket – Sockets are the small connection points on nodes that connection lines draw to/from. Input sockets (always found on the left side of nodes) and Output sockets (always found on the right side of nodes) are round with black centers, while variable sockets (always found on the bottom edge of nodes) are either round if they input a variable value or a triangle if they output a variable value. Variable socked centers are color coded to match the most commonly used variable types the given socket supports (string = green, float = blue, bool = red, etc.). Less common variables will be white in color.

Connection Lines – Connection lines are the lines that are drawn between the nodes.

Canvas – the section of the uScript Tool where the visual programming graph is located.

Graph – sometime used to describe an entire uScript visual script or a uScript script (.cs) output file.