



Scripting Reference

```
(Drive=="RWD" || Drive=="4WD") {  
  colliderRL.motorTorque = MaxEngineSpeed * Input.G  
  colliderRR.motorTorque = MaxEngineSpeed * Input.G  
}  
  
colliderFR.steerAngle = colliderFL.steerAngle;  
  
ColliderCenterPointFL = colliderFL.transform.TransformP  
ColliderCenterPointFR = colliderFR.transform.TransformP  
ColliderCenterPointRL = colliderRL.transform.TransformP  
ColliderCenterPointRR = colliderRR.transform.TransformP  
  
if (Physics.Raycast( ColliderCenterPointFL, -colliderFL.tr  
  wheelFL.transform.position = hit.point + (colliderFL.t  
)else{  
  wheelFL.transform.position = ColliderCenterPointFL -  
}  
wheelFL.Rotate(colliderFL.rpm * 6 * Time.deltaTime, 0,  
  
if (Physics.Raycast( ColliderCenterPointFR, -colliderFR.t  
  wheelFR.transform.position = hit.point + (colliderFR.t  
)else{  
  wheelFR.transform.position = ColliderCenterPointFR -
```

Created By: A-Lab Software Limited
Date Created: July 2010
Date Modified: July 2010
Revision: 1.0

Table of Contents

Introduction	3
Understanding Program Flow	4
Script Event Handlers	5
Script Compilation Order	6
Accessing/finding GameObjects within a Scene	7
Getting and Setting a GameObject name	8
Changing a GameObject's position, rotation and size	9
Creating a GameObject Primitive	10
Accessing Scripts attached to GameObject's	11
Accessing the Components of a GameObject	12
Adding Components to a GameObject	13
Creating New Empty GameObject's	14
Connecting GameObject's together within the Scene Hierarchy	15
Copying/duplicating GameObject's	16
Reading the Keyboard	17
Reading the Mouse Buttons	18
Reading and Writing Text Files	19
Loading a Scene at Runtime	20
Re-using assets, and creating Prefabs	21
Ways to Pause a Game or Application	22
Split Screen and Camera Switching	23
Runtime GUI Scripting	24
Loading Resources	25
Connecting/linking Scenes together via a Menu System	26
Editor GUI Scripting	27
Getting and Setting the active GameObject	28
EditorGUI Common Dialog Boxes	29
Enabling/Disabling GUI Controls	30

Introduction

This reference document has been designed to help new and advanced users alike.

Whilst I have been learning Unity, I have found it a somewhat time consuming to find answers to seemingly simple questions regarding scripting syntax. I therefore started creating this document as a way to record my findings, over time the amount of information within the document has grown, and will continue to grow as I learn more and more about Unity.

The problem with scripting, is you find a way to do something, for instance change a GameObject's position, or rotation, and then a few months later I will be writing some scripting code, and want to do the same thing, however I cannot remember the exact syntax, or worse still in which script I previously wrote the lines of code.

So in this guide you will find snippets of scripting code that perform a particular action or actions.

This document is not a user guide or training manual, it is more of a quick reference guide!

Within this document you will find a number of links to the Unity Scripting Reference pages online, these have been added to allow you to jump straight to a particular topic regarding the snippet of code within that section. The amount of times I have read about a command within the Unity Scripting Reference pages online, and then I cannot find the page again when I go to refer back.

So here's the first link, it's a good place to start for an introduction to scripting within Unity is the **Scripting Overview**:
**** [LINK REMOVED FROM SNEAK PEAK VERSION](#) ****

One final comment before you jump into the document, the Unity scripting interface allows you to do absolutely everything you can from within the Unity Editor, after all the Editor is written in Unity script itself. So what I am trying to say is Unity is very powerful, and is such an exciting product for many uses, so have fun and create something amazing!

Understanding Program Flow

When I learnt to code some 20 years or so ago, I was programming in basic, and a programs execution was sequential, meaning that each line of code was executed one after the other, I had the option to jump about the code a bit with goto statements, but no functions or procedures were available back then.

Since those early days, programming languages have come along way, evolving along the way, even basic is still going in the form of languages such as Visual Basic.

Languages today support functions, procedures, and classes, and they are often event driven, meaning that they do not necessarily execute line by line anymore.

And Unity is no different, whether using Javascript or C#, it makes no difference, Unity is pretty much event driven and offers you a great deal of builtin events than can be triggered, and you can write code to react to those events. Refer to the next section **Script Event Handlers** to learn more about Unity event triggers.

One thing I struggled with when first starting to code in Unity, is how does the program flow work, where should I place code that I want to execute as soon as a scene is loaded, or to execute once every game cycle.

The confusion for me was that you can attach scripts to any gameobject in a scene, for instance when you start a new scene, all you have is a camera gameobject (yes a camera is a gameobject), and you can attach a script to the camera.

**** INFO REMOVED FROM SNEAK PEAK VERSION ****

Script Event Handlers

Within a Unity script, there are a number of different default functions that can be used; these functions get triggered by certain events within a Unity project.

These events are the life blood of a Unity application or game, by adding code within these event handlers, you can control what happens within your game and when!

**** INFO REMOVED FROM SNEAK PEAK VERSION ****

Script Compilation Order

Sometimes when you are scripting, and referencing other scripts, or type definitions from other scripts, you may receive errors within Unity about undefined types, or null reference objects.

These errors can sometimes be caused by the order in which Unity compiles scripts within your project.

**** INFO REMOVED FROM SNEAK PEAK VERSION ****

The following article is gold dust!

**** LINK REMOVED FROM SNEAK PEAK VERSION ****

Accessing/finding GameObjects within a Scene

This is probably the most used piece of scripting code, you will want to access GameObject's within a scene for many reasons. And thankfully it is a very quick and easy thing to do.

*** INFO REMOVED FROM SNEAK PEAK VERSION ***

Getting and Setting a GameObject name

Accessing or setting the name of a GameObject is very simple, to change the name use the following code:

***** CODE REMOVED FROM SNEAK PEAK VERSION *****

To get the name of a GameObject, use the following:

***** CODE REMOVED FROM SNEAK PEAK VERSION *****

Changing a GameObject's position, rotation and size

Accessing the transform parameters of a GameObject is very simple, to change the position of a GameObject you have the option to use either the localPosition or the position settings, localPosition is the position relative to the objects parent, and the position is relative to absolute world space.

*** INFO REMOVED FROM SNEAK PEAK VERSION ***

To change a GameObject's rotation to a particular angle like you would using the Inspector in the Unity Editor, you can do the following:

*** CODE REMOVED FROM SNEAK PEAK VERSION ***

To change a GameObject's scale, like you would using the Inspector in the Unity Editor, you can do the following:

*** CODE REMOVED FROM SNEAK PEAK VERSION ***

Creating a GameObject Primitive

Creating a primitive GameObject such as a cube or plane is very easy, if you know how that is, took me quite a while to find this in the documentation.

The following code will create a 1 metre square cube at location 0, 0, 0.

***** CODE REMOVED FROM SNEAK PEAK VERSION *****

Below is a list of the other primitives that can be created:

Sphere	A sphere primitive
Capsule	A capsule primitive
Cylinder	A cylinder primitive
Cube	A cube primitive
Plane	A plane primitive

Accessing Scripts attached to GameObject's

This is a topic I spent a long time investigating, trying to find out how to access scripts within a scene that are attached to other GameObjects, and the variables within those scripts.

***** CODE REMOVED FROM SNEAK PEAK VERSION *****

In the above code ScriptName is the actual name of the script attached to requiredGameObject, and otherScript is the reference you use to access the script, for example:

```
otherScript.someVariable = 10;
```

Where someVariable is a public variable from within the script attached to the GameObject requiredGameObject.

Accessing the Components of a GameObject

This is another topic I spent a long time investigating, trying to find out how to access components, such as rigidbodies, colliders, mesh renderers and there parameters.

Accessing a rigidbody is the easiest, just use `GameObject.Find` (refer to **Accessing/finding GameObjects within a Scene** section in this document), and then use the following syntax.

***** CODE REMOVED FROM SNEAK PEAK VERSION *****

or

***** CODE REMOVED FROM SNEAK PEAK VERSION *****

To access a collider attached to a `GameObject`, you need to use the `GetComponent` command.

***** CODE REMOVED FROM SNEAK PEAK VERSION *****

Now you can access the collider parameters as follows:

***** CODE REMOVED FROM SNEAK PEAK VERSION *****

You can also access other Components attached to a `GameObject`, the Mesh Renderer for instance, again use the `GetComponent` command.

***** CODE REMOVED FROM SNEAK PEAK VERSION *****

Here is a list of other Components that you can access if attached to a `GameObject`.

***** INFO REMOVED FROM SNEAK PEAK VERSION *****

Adding Components to a GameObject

This is a very handy feature, being able to add different components to GameObject's at runtime. And again it is very easy to do, see below:

*** CODE REMOVED FROM SNEAK PEAK VERSION ***

Creating New Empty GameObject's

This is something you may want to do quite often. To create a new GameObject at runtime, do the following:

*** CODE REMOVED FROM SNEAK PEAK VERSION ***

The above line creates a new GameObject called go, with a name of newGameObject.

Connecting GameObject's together within the Scene Hierarchy

Something you might want to do is connect a newly created GameObject to another GameObject within a scene. By default all newly created GameObject's are created in the root of the scene, you can however make GameObject's children of other GameObject's, making them the parent.

As with Unity this is easy to do, however it took me a long time to find out how! So here's how to do it!

***** CODE REMOVED FROM SNEAK PEAK VERSION *****

The above line of code makes newGameObject a child of existingGameObject, which is now a parent object.

Copying/duplicating GameObject's

This is an interesting topic, as there are many uses for duplicating GameObjects.

In Unity copying or duplicating a GameObject is known as cloning, and the command to clone a GameObject is called Instantiate.

*** INFO REMOVED FROM SNEAK PEAK VERSION ***

Reading the Keyboard

There are a few ways to read the keyboard.

*** INFO REMOVED FROM SNEAK PEAK VERSION ***

Reading the Mouse Buttons

There are a few ways to read the mouse buttons.

**** INFO REMOVED FROM SNEAK PEAK VERSION ****

Reading and Writing Text Files

Being able to store and retrieve data within files is very handy for many uses, such as saving high scores, or level data, or for creating your own file format.

To write to a file simply use the following lines of code:

```
** CODE REMOVED FROM SNEAK PEAK VERSION **
```

To read from file, use the following lines of code:

```
** CODE REMOVED FROM SNEAK PEAK VERSION **
```

Loading a Scene at Runtime

To load a level at runtime, we use the Application class, you used this very useful class in the section on Reading & Writing to Text files.

*** INFO REMOVED FROM SNEAK PEAK VERSION ***

Re-using assets, and creating Prefabs

One part of Unity I have found very powerful are prefabs, they allow you to store GameObjects within your Project in a format that can be exported and imported into different Unity projects.

**** INFO REMOVED FROM SNEAK PEAK VERSION ****

Ways to Pause a Game or Application

Within your projects, there may be times when you want to pause or add a delay to your projects execution.

**** INFO REMOVED FROM SNEAK PEAK VERSION ****

Split Screen and Camera Switching

Ok, this is a topic I was keen to learn about, yet I spent sometime searching the forums for answers.

In fact the first part of this section Split screens is covered in the Unity Reference Manual under the Camera documentation, but it's not easy to find, and it's a little unclear.

Also, I did not just want to split the screens, I also want to do things like add picture in picture type setups, imagine a racing car game, and the players view is from within the car, and then you want to add the wing mirror views.

All this is possible with Unity!

**** INFO REMOVED FROM SNEAK PEAK VERSION ****

Finally, what about camera switching, this is easier than you think, when I first tried to do this in Unity I assumed you needed to add as many cameras to the scene as I wanted to switch between.

You can do it that way, and in doing so you can position the cameras exactly where you want in a scene, but there are times when you will want to switch between more dynamic cameras, such as different player views, let's take a driving game, you may want to switch between an in-car view, and a view from behind the car. For this type of scenario you only need one camera, and you just change the camera position via code, simple really.

Below are two snippets of code, one uses a single camera, the other uses two separate cameras to switch views.

**** INFO REMOVED FROM SNEAK PEAK VERSION ****

Runtime GUI Scripting

I have had a number of emails from Unity users asking me how I put together my Runtime Vehicle Editor.

*** INFO REMOVED FROM SNEAK PEAK VERSION ***

Loading Resources

Unity has a very nice class named Resources, this class allows you to dynamically load resources such as textures, prefabs and so on, at runtime within your project.

*** INFO REMOVED FROM SNEAK PEAK VERSION ***

Connecting/linking Scenes together via a Menu System

This is a topic I was keen to learn about, as creating scenes in Unity is all very well, but how do you link them together in your final application.

*** INFO REMOVED FROM SNEAK PEAK VERSION ***

Editor GUI Scripting

Ok, so now we are onto the last topic, and to be honest one of my favourites, as it has enabled me to create my Vehicle Editor, and I also have several other ideas based around my RapidUnity concept that Editor GUI Scripting will allow me to create!

Why extend or create Editor GUI elements? Well the first time I investigated this topic was when I was creating my slot car game, I was using the Unity Editor to create that track layouts manually, each track section prefab needed to be dragged into the scene, and then positioned, and rotated, it was a repetitious task and time consuming.

So I thought there must be an easier way, and I discovered Editor GUI Scripting, basically I was able to create a simple dockable window within the Unity Editor, add some buttons which I could use to load a prefab track section out of my resources folder, and position and rotate it all in code.

This enabled me to build my track layout scenes in no time at all!

**** INFO REMOVED FROM SNEAK PEAK VERSION ****

Getting and Setting the active GameObject

As well as being able to create Editor GUI's, you can also add script code to get the currently selected gameobject within the scene view, or set the currently selected gameobject.

To get the active gameobject, use the following code:

***** CODE REMOVED FROM SNEAK PEAK VERSION *****

To set the active gameobject, like some one clicking it in the Hierarchy panel, use the following code:

***** CODE REMOVED FROM SNEAK PEAK VERSION *****

The first line hilights the gameobject in the Hierarchy panel, and the second line actually sets the active gameobject within the Selection class.

EditorGUI Common Dialog Boxes

Another really nice feature within the Editor GUI Scripting, is the ability to display common dialog boxes, such as an Open file dialog, or a modal Message Box.

Use the following code to display an Open file dialog:

```
** CODE REMOVED FROM SNEAK PEAK VERSION **
```

The value returned in path, will be the full path of the file selected by the user.

To display a modal message box, a modal message box is one that cannot be ignored, it is displayed above all other dialogs, and requires closing before the focus is released.

```
** CODE REMOVED FROM SNEAK PEAK VERSION **
```

Enabling/Disabling GUI Controls

The final item within this Editor GUI Scripting section, covers how to enable and disable GUI controls within your editor GUI's.

When I say disable, I mean to show the control greyed out, so the user can see it, but not click it for instance.

Because the GUI engine is a state engine, you simply use the following code to disable a GUI control.

***** CODE REMOVED FROM SNEAK PEAK VERSION *****