

UnrealScript Tutorial-1 Introduction to UnrealScript

Overview:

UnrealScript is a programming language developed for the specific purpose of creating 3-D game environments. In that sense, UnrealScript is not a general purpose programming language. UnrealScript is very similar to C/C++ and Java. However, it differs widely from these traditional languages in that UnrealScript addresses issues that these other languages do not. The major features of UnrealScript are:

- UnrealScript is *completely* object oriented.
- UnrealScript supports the concepts of *time, state, properties, and networking replication*.
- UnrealScript has built-in AI.
- UnrealScript has tens of thousands of lines of proven code available to the game programmer through the use of a few simple lines of code (called inheritance in Object Oriented Programming - OOP).
- UnrealScript supports development simplicity over that of execution speed.
- UnrealScript script operates at the object and interaction level rather than the bits and pixels level.
- UnrealScript has game specific concepts mapped into the language itself.

There are many other technical features supported by UnrealScript. However, these more advanced technical features will be covered in other tutorials.

Getting Started

In this tutorial you will be shown how to create the simplest possible UnrealScript program. In that sense the program will not do much but it will help you learn many of the important concepts unique to UnrealScript. It is therefore important that you follow each of the following steps. Skipping any one of them could result in your first program not working.

1. Go to the directory: **C:\UDK\UDK-2009-11-2\UTGame\Script** and make sure that none of the files there are write protected. You do this by right-clicking on the file, selecting the **Properties** option from the resulting pop-up menu and making sure that the **Read-only** box is NOT checked. Note that the directory **\UDK-2009-11-2** may be different for your installation. Use the directory where the version of your UDK editor was installed. Note that you will only have to do this only once.
2. Go to the directory: **C:\UDK\UDK-2009-11-2\Development\Src** and create a folder called **MyScripts**. Open that folder and create another folder called **Classes**. Your new directory structure should now look like this:
C:\UDK\UDK-2009-11-2\Development\Src\ MyScripts\Classes This **Classes** folder is the folder where all of your UnrealScripts will be stored.

3. Next we need to tell the UDK system where all of our new UnrealScripts will be stored. To do this go to: **C:\UDK\UDK-2009-11-2\UTGame\Config\DefaultEngine.ini**
The **DefaultEngine.ini** file is nothing more than a simple text file that you can open in notepad. Add the following line of code: **+EditPackages=MyScripts** just *below* the line of code that says: **+EditPackages=UTGameContent**, Note that you do NOT put the **Classes** directory after the **MyScripts** directory. The UnrealEditor will always assume that a **Classes** directory exists for your new scripts.

For these tutorials, you only need to do the above steps once. Your system is now set up for you to create your first UnrealScript program. However you first need to make some observations about your Unreal Editor so that your new script will make sense to you.

You will need to start your Unreal Editor. When you do, a pop-up window will appear stating *Scripts are outdated. Would you like to rebuild now?* Click on **No**. Your Unreal Editor will now come up.

Go to the **View** menu and select **Browser Windows** and then **Actor Classes**. The **Actor Classes** window will pop up.

Look down the list of the **Actor Classes** window for **Note**. Observe that **Note** does not have a **+** sign preceding it indicating that there are no subclasses below it.

Some Important OOP Terms and Concepts (optional stuff)

You have seen the word **Class** used in this tutorial. A **Class** is nothing more than a bunch of programming code that has already been done for you. As an example, the **Note Class** uses all of the lines of code inside the **Actor Class**. We can therefore say that the **Note Class** is a subclass of the **Actor Class**. We say this because the **Note Class** uses all of the coding goodies from the **Actor Class**.

Looking at the Actor Classes window, all of the Classes shown there (such as Brush, CameraActor, ComponentTestActorBase, etc) are subclasses of the Actor Class.

If you right-click on the Brush Class you will get a pop-up menu with two options. If you select the **View Source Code** option you will see the following line within the code:

```
class Brush extends Actor
```

The above line of code tells the editor that the new class Brush is an extension of the existing class Actor. Doing this gives the new class Brush access to all of the coding goodies that exist in the class Actor. The class Brush can now add its own programming goodies as well as opting not to use all of the class Actor programming goodies. What this means is that the tons of programming goodies done by the Unreal developers themselves is now easily available to the Brush class. This was all done with one simple line of code (see above).

If you click on the + sign next to the Brush class you will see a subclass of the class Brush called BrushShape. If you look at the source code for the BrushShape class you will see the following line of code in it:

class BrushShape extends Brush

That above line of code tells the editor that this new class BrushShape will use all of the programming goodies from the class Brush, which, by the way, has all of the programming goodies from the class Actor (because the class Brush is an extension of the class Actor). Now, the new class BrushShape can add its own goodies, plus using the goodies from class Brush and class Actor.

This simple system of passing on tons of tested code in Object Oriented Programming (OOP) is called **inheritance**. Thus you could say that BrushShape inherited the code from class Brush which in turn inherited the code from class Actor. To see all of the code in class Actor, simply right click on it and select to view the source code.

Using this system you could also say that the class Brush is a child of the class Actor and that the class BrushShape is a child of the class Brush. The other way around you could say that the class Brush is the parent of the class BrushShape and that the class Actor is the parent of the class Brush. All of this is OOP speak. But it is important to know because a lot of OOP speak is used in the literature to describe what you are doing here.

Anytime two or more classes are an extension of the same class, they are referred to as **siblings**. For example, BrushShape and Volume are siblings since they both have the Class brush as their parent class (check out the code to see for yourself).

Where Do You Go From Here

In the next tutorial on UnrealScript, you will create your very first UnrealScript program.