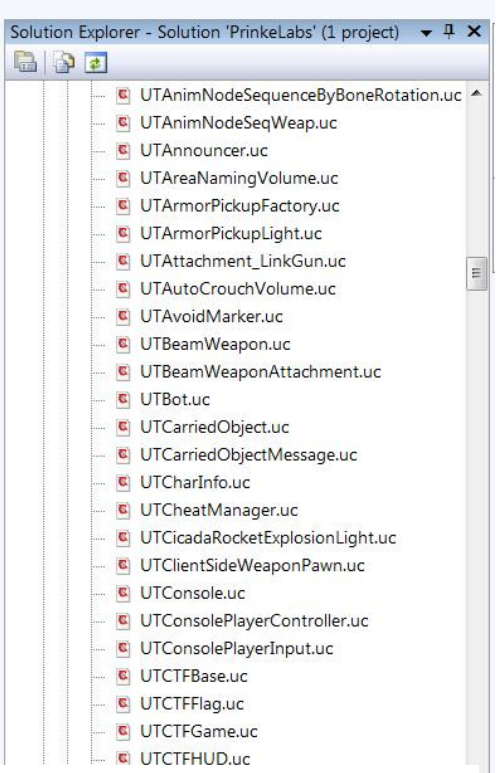
Chapter 1 - Introduction and Setup

Contents

Introduction

The Unreal Development Kit is a powerful tool that can develop everything from mobile games to top-of-the-line AAA titles. With tools like Kismet and Matinee, almost anyone can quickly and easily produce quality levels and content. Unfortunately those tools only take one so far. Kismet has many limitations that aren't immediately evident, such as its inability to run script on more than one object from the same spawner, and if UDK developers don't want to be stuck with Unreal Tournament guns, bots, objects, and gameplay, then it's entirely necessary to start learning Unrealscript, the engine's proprietary scripting and programming language, in order to get the most out of the engine's capabilities.



Where does one asset begin and another one end?

This can be an extraordinarily daunting task. The good news is that Unrealscript itself is as straightforward as any other programming language. If you've ever learned C++ or Flash Actionscript, Unrealscript is very similar, using the same variable sets, the same flow control concepts, and very similar syntax. It also comes loaded with useful scripts that feature incredibly useful functions for developing new game code. The problem? There's hundreds of these scripts and it can be very difficult to tell where to start coding a new game, or even just a new asset. While more documentation and tutorials are always being released, most of them focus on introducing new users to basic programming concepts as opposed to explaining Unrealscript's organization and features.

This documentation is meant to fill that purpose, assuming you already understand basic programming concepts and providing a roadmap for your Unrealscript needs as we go through
building a new game from the ground up.

In this chapter we're going to start by figuring out where Unrealscript is, getting our environment set up for coding, and organizing our file structure. As you can see from the figure above, we've got a lot to figure out.

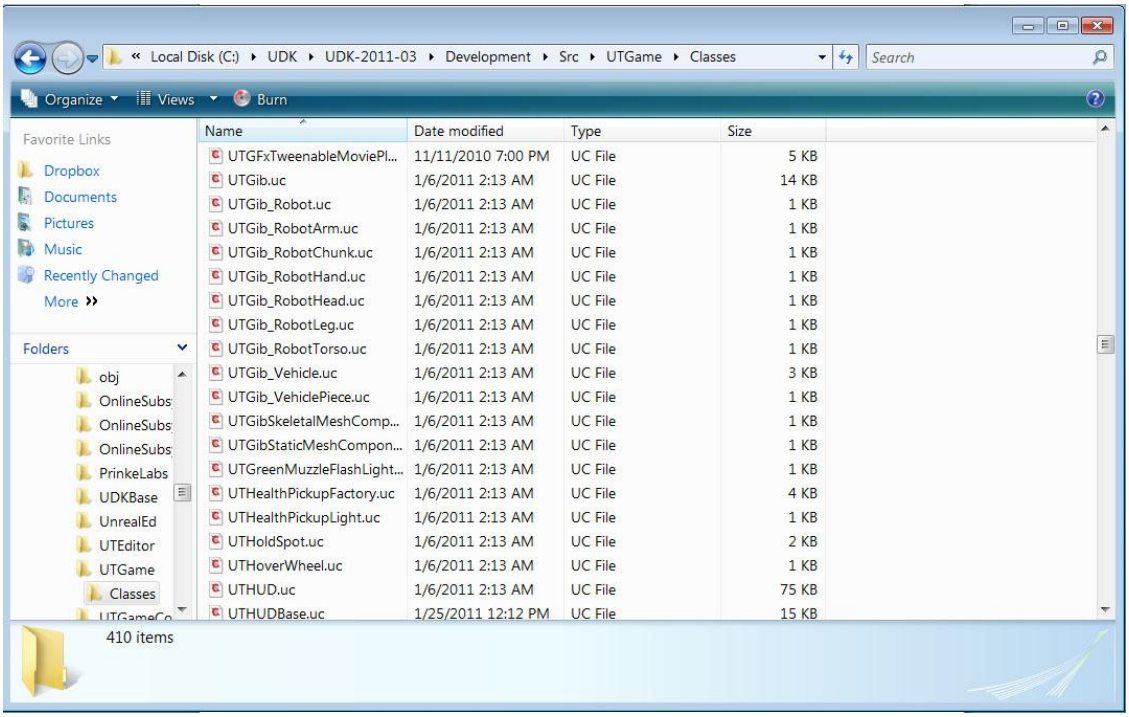
Where is Unrealscript?

A good question was posed by a friend of mine. "Where exactly *is* Unrealscript?" He and many of my classmates had assumed there would be some kind of Unrealscript editor inside the UDK editor, much like how Flash and the Unity Game Engine have their own scripting editors. Unfortunately UDK has no

such feature, so we have to provide our own scripting editor. Before we get into that, though, let's answer his question and figure out exactly where UDK keeps its guts.

The Unrealscript Directory

Open up the directory that houses your UDK installation; in my case it would be C:\UDK\UDK-2011-03. Open up Development\Src. You'll see a whole ton of folders. Open up any of them and you'll see a folder titled "Classes." Open *that* folder up, and voila.



You should see more Unrealscript Class (.UC) files than you can shake a stick at, grouped into a variety of different folders. As the name "src" suggests, this folder contains all of the source code for your UDK project. I shall list and explain each of the folders in the SRC directory here.

Unrealscript Folder Guide

Core

Contains many of the most basic Unrealscript Classes, such as "Objects." These should never be edited.

Engine

Contains basic engine

GameFramework

GFxUI

Contains groundwork for Unreal's Scaleform support, including Scaleform-related Kismet nodes.

GFxUIEditor
Contains Scaleform-related Unreal Editor components; mainly scripts for importing and instantiating Flash SWF files and the like.

IpDrv
Internet Protocol-related scripts.

MobileGame
Basic mobile game support, designed specifically for touchscreen. If you're using UDK mobile you are likely working with these scripts a lot.

MyMod
There's no scripts in here. Just a text file that says "DO NOT DELETE." I would take its advice.

obj
Contains debug output; no Unrealscript files are in this folder.

OnlineSubSystemGameCenter
Unknown.

OnlineSubSystemPC
Unknown

OnlineSubSystemSteamWorks
Unknown

UDKBase
Can be best described as a folder that acts as a middleman between scripters and UDK's most basic functions. Much of Unreal Tournament's original functionality for player pawns, weapons, vehicles, and other common ingame actors has been split off and placed here to provide basic functionality without the overhead of UT-specific game elements.

UnrealEd
Contains scripts pertaining to how the UDK Editor displays information to the user, including content browser option displays, import options for specific types of objects, and much more.

UTEditor
Purpose unknown. Appears to be related to the scripts in the UnrealEd directory.

UTGame
For those of you who *want* UT-specific game elements, Unreal Tournament-specific code has been placed here. Much of what's available is cut down from the full release of Unreal Tournament.

UTGameContent
Contains physical game content for Unreal Tournament; IE, the actual ingame actors.

This sums up all the source code for our UDK Project. We will be exploring much more about the full road map of Unrealscript later on, but for right now that should give you a clear idea of what types of scripts are located where. As their behavior goes they are able to reference one another easily, as if they were all contained in the same folder, so you will often see scripts from UTGame extending scripts from Engine.

How do I Edit It?

Now that we've figured out where to find our UDK project's Unrealscript library it's time to start playing with it. You can edit .UC files in any text editor, including notepad, but there's more than a few text editing programs that are better suited to it than that.

Suggested Unrealscript Editors

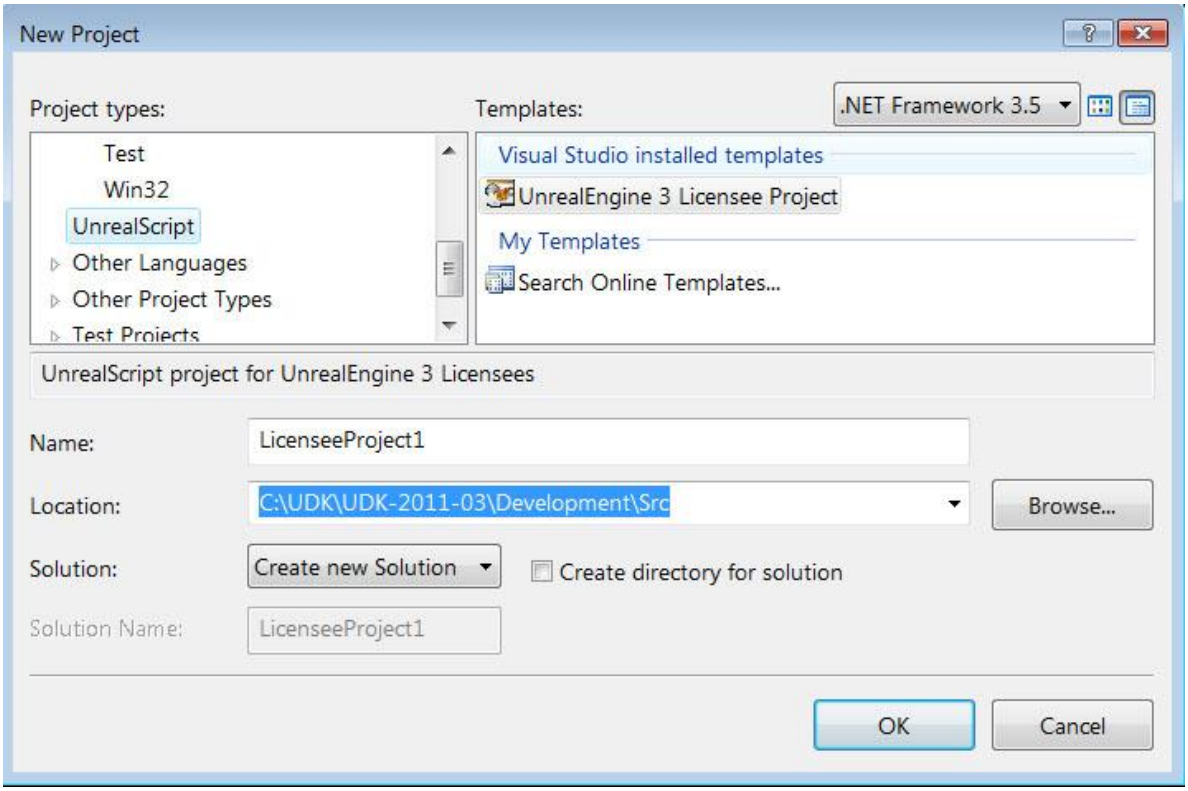
The following are different Unrealscript editors that I've found and links to guides for setting them up.

Microsoft Visual Studio and NFringe

Visual Studio with the NFringe toolset is the Cadillac of Unrealscript tools. For many people it's is prohibitively expensive, but there are many places that will either sell it cheaply or provide it free to students. Microsoft has a web site called DreamSpark that provides such a service. You can visit it here.

NFringe is a third-party Unrealscript plugin for Visual Studio created a group called PixelMine. The download for its installer can be found here, though if you ever want to make money off your Unrealscript projects there's a bit of a licensing fee to go along with the tool. Fortunately it costs nothing for a student or a modder to install it for learning purposes. It is an extremely powerful tool, providing not only Unrealscript syntax highlighting but also auto-complete support, allowing you to put in your syntax and call up variable names quickly and typo-free.

Once you have Visual Studio and PixelMine both installed, setting up Visual Studio to work inside your project is simple. Open Visual Studio. Select File -> New -> Project. Then, from "Project Types," select Unrealscript. From "Templates," select UnrealEngine 3 Licensee Project. Name it whatever you want.



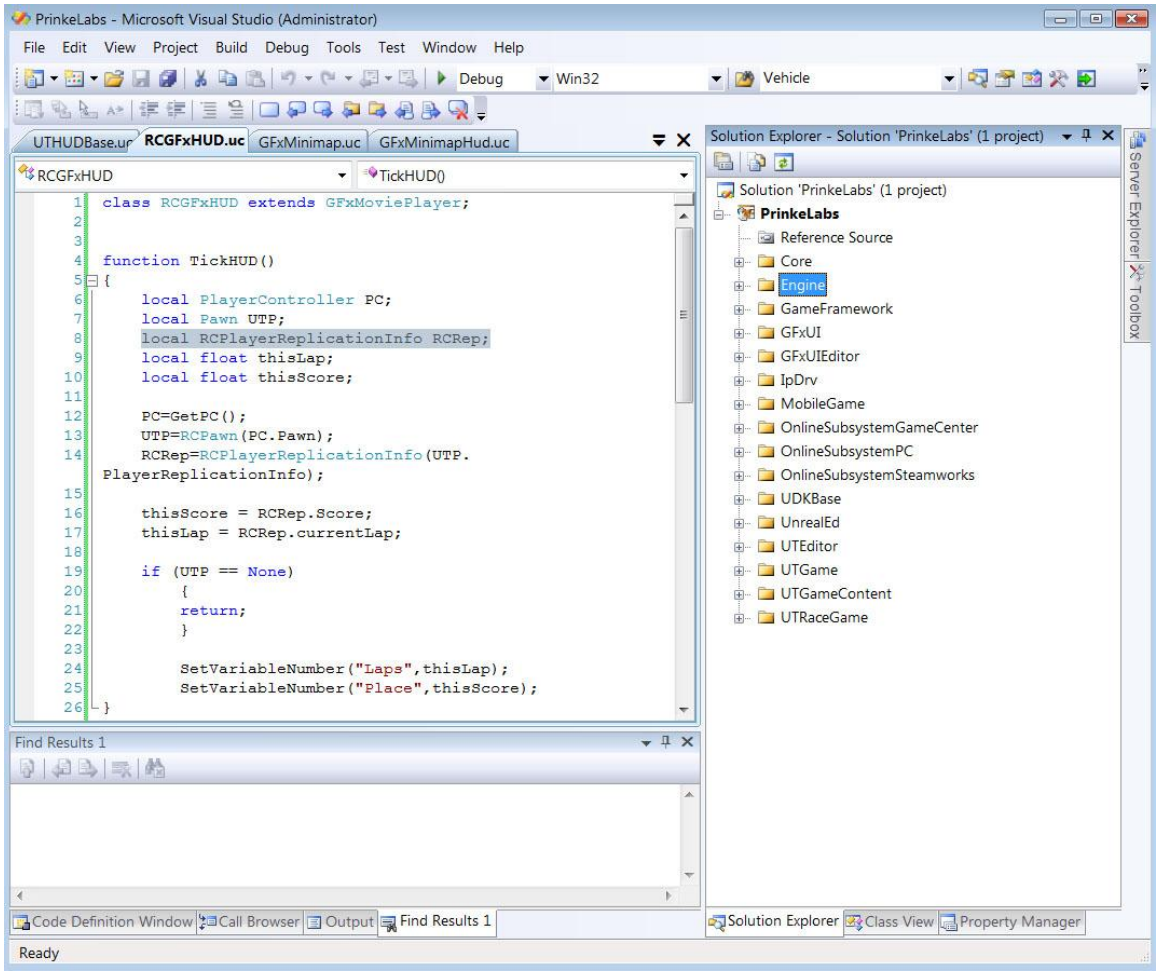
Make sure to uncheck "create directory for solution." Locate the Src folder for your installation of UDK, just as we located it above, and set that as your project location. Now, hit "OK."

We've got the project started in Visual Studio now, but there's a few key things missing. Namely, *all* of our Unrealscript files. Close Visual Studio as soon as it opens, and go to the folder where you set up the project. It will be in a folder in the Src directory named for whatever you named your project. For instance:

C:\UDK\UDK-2011-03\Development\Src\MyProject

Inside there will be three files: an MyProject.sln, MyProject.suo, and MyProject.uproj. If they aren't all there, make sure to show hidden files inside this folder. Grab all three of them, cut them, move up to the Src folder, and paste them.

Now, just open up the SLN file with Visual Studio again and voila--your Visual Studio solution will now have the entire UDK script library at your disposal, displayed on the right in a handy little directory tree for easy, nonlinear browsing.



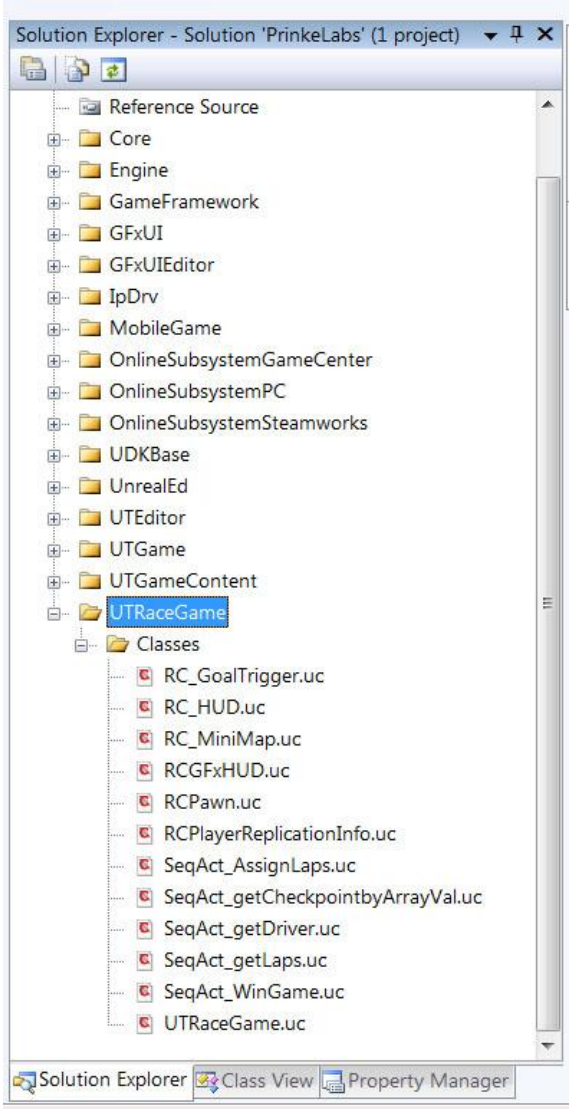
ConTEXT

Notepad++ and ConTEXT are two of many freeware "Notepad Substitute" programs specially developed for coding of all sorts. You may already be using one of these two programs for any number of purposes, including web development.

ConTEXT is the one that I recommend the most, however, as its Unrealscript library installation is easier and its syntax highlighter is less obtrusive. ConTEXT can be downloaded here, and the Unrealscript highlighting file can be found here; you need only copy and paste the code provided into a file, save it out as a .chl file, and put it in the "Highlighters" directory under ConTEXT.

In just a few moments you're ready to roll and have handy Unrealscript highlighting. It isn't as powerful as Visual Studio's AutoFill support, but it does provide all the necessities and also features a handy File Explorer like Visual Studio's.

Organizing your Script Directory



Now that we have our scripting environment set up, it's best that we learn about good organization. You *can* put your new Unrealscript files in the existing directories, but then they'll all get compiled along with the original scripts, which is bad organization for both the engine and you as it can become difficult to keep track of what's new scripts and what's old scripts.

You *can* edit the original scripts to create content as well, but it's not recommended as it can cause instability and force you to do a fresh install. We want to preserve the integrity of our scripting library as much as possible, and that means keeping our scripts separate from the original UDK scripts.

In order to achieve this, we must first create a new folder in the Src directory and name it after your project. Inside that folder, create another new folder called "Classes." This is necessary for UDK to be able to recognize that it should compile and reference the scripts that are inside. As you can see on the left, this is extremely helpful for keeping our custom scripts in a place where we can easily find them.

Editing UDK's Config Files to Compile your Script

But that's not quite enough. UDK *also* needs to be able to find them, and unless it's explicitly told to, it never will. Navigate to the "UDKGame" folder in your UDK installation directory, and then go into the "Config" folder.

Inside are a bunch of .INI files--but we only want one. Specifically: **UDKEngine.ini**. Copy it and make a backup so that we can go back to the original if we ever mess something up. Call it "UDKEngine_Backup.txt," re-naming it as a .txt file so that Unreal doesn't accidentally start reading it. Open it up in Notepad and use CTRL+F to search for the word "EditPackages." It'll be under a heading called [UnrealEd.EditorEngine] -- Like So:

```
[UnrealEd.EditorEngine]

LocalPlayerClassName=UnrealEd.EditorPlayer
bSubtitlesEnabled=True
GridEnabled=True
SnapScaleEnabled=True
ScaleGridSize=5
SnapVertices=False
```

SnapDistance=10.000000
GridSize=(X=16.000000,Y=16.000000,Z=16.000000)
RotGridEnabled=True
RotGridSize=(Pitch=1024,Yaw=1024,Roll=1024)
GameCommandLine=-log
FOVAngle=90.000000
GodMode=True
AutoSaveDir=..\..\UDKGame\Autosaves
InvertwidgetZAxis=True
UseAxisIndicator=True
MatineeCurveDetail=0.1
Client=WinDrv.WindowsClient
CurrentGridSz=4
bUseMayaCameraControls=True
bPrefabsLocked=True
HeightMapExportClassName=TerrainHeightMapExporterTextT3D
EditorOnlyContentPackages=EditorMeshes
EditorOnlyContentPackages=EditorMaterials
EditorOnlyContentPackages=EditorResources
EditPackagesInPath=..\..\Development\Src
EditPackages=Core
EditPackages=Engine
EditPackages=GFxUI
EditPackages=GameFramework
EditPackages=UnrealEd
EditPackages=GFxUIEditor
EditPackages=IpDrv
EditPackages=OnlineSubsystemPC
EditPackages=OnlineSubsystemGameSpy
EditPackages=OnlineSubsystemLive
EditPackages=OnlineSubsystemSteamworks
bBuildReachSpecs=FALSE
bCustomCameraAlignEmitter=TRUE
CustomCameraAlignEmitterDistance=100.0
bDrawSocketsInGMode=FALSE
bSmoothFrameRate=FALSE
MinSmoothedFrameRate=5
MaxSmoothedFrameRate=120
EditPackagesOutPath=..\..\UDKGame\Script
FRScriptOutputPath=..\..\UDKGame\ScriptFinalRelease
EditPackages=UDKBase
EditPackages=UTEditor
InEditorGameURLOptions=?quickstart=1?numplay=1
EditPackages=UTGame
EditPackages=UTGameContent

This is where UDK gets told to take specific directories in your Src directory and convert them into Unreal Packages, which are essentially specialized zip files that UDK games can access easily but that players can't edit without some really specialized tools for unpacking them.

You want to add a line at the bottom of this list naming the custom directory or directories that you set up before--IE: "**ModEditPackages=MyProject,**" or whatever you named that directory. "ModEditPackages" refers to any directory that isn't a default directory, as opposed to "EditPackages." Save the INI and close it.

Now whenever you start up UDK, it will compile your Unrealscript files from your custom folder into its own separate package. However, you'll have to remember to do this for **every new folder that you create** for both this and future projects.

Conclusion

And to think, we haven't even started scripting anything! By now, though, we've started to answer our most basic questions, and we now know both where Unrealscript is kept, how it's organized, and how we can organize our script to play friendly with our UDK Project. In our next chapter we'll go a little bit deeper and start dissecting the basics of Unrealscript Classes and their *own* internal organization and syntax.