The Dusty Tome of UnrealScript? Black Magic

Document Summary: Tips for Unreal Tournament mod authors regarding the mysteries of UnrealScript?.

Document Changelog: Originally created by Brandon "GreenMarine" Reinhart for Epic Games, Inc. on 12/08/99.

Index to Invocations

Introduction

UnrealScript? is a powerful programming language, but one steeped in the shrouded mystery of time. The purpose of this document is to reveal some of the more arcane methods of invoking this language. The tips and guides in this document will be compiled over time, from the author's one experience and from the experience of those who submit entries. For now, entries will be unsorted.

UnrealScript? Spellcraft

Using UWindow for Mutator Options

Author: Jack 'Mek' Porter
Email: jack@epicgames.com

Mutators with configuration options should make their own UWindow subclasses for configuration, and insert themselves into the (usually invisible) Mods menu, instead of using advanced options. Code examples of this can be found here: http://www.planetunreal.com/mutation/tutorials/mekoptions.html.

Using SpawnNotify? to control object creation.

Author: Brandon 'GreenMarine' Reinhart
Email: brandon@epicgames.com

The SpawnNotify? actor can be used to intercept and replace actors as they are instantiated. To do this subclass the SpawnNotify? actor. In the default properties, set the ActorClass? variable to the class of the actor you want to intercept. In the SpawnNotification? method, destroy the actor that is passed in the parameter and spawn your new actor instead. This can be used to implement a custom PlayerReplicationInfo?:

```
class MyPRISpawnNotify expands SpawnNotify;
event Actor SpawnNotification( Actor A )
{
    if (!A.IsA('MyPlayerReplicationInfo'))
    {
        A.Destroy;
        return Spawn(class'MyPlayerReplicationInfo');
    } else
        return A;
}
defaultproperties
{
    ActorClass=class'PlayerReplicatonInfo'
}
```

When you spawn your SpawnNotify? object it will add itself to the level's SpawnNotify? list. Objects in that list are queried with SpawnNotification? whenever an object of their ActorClass? is created. Notice that the example is careful to check the type of the actor. Failing to do this would cause an infinite loop (as our spawn notification destroyed the MyPlayerReplicationInfo? it tried to create).

Using Entry for persistent object creation.

Author: Mongo
Email: mongo@planetunreal.com

Most UnrealScript? objects are garbage collected when the level changes. If you have an object you need to be persistent across a level change, spawn it inside the 'Entry' level. The Unreal engine always has a small level loaded that exists for the lifetime of a game session, the 'Entry' level. Objects that are loaded inside of this level will not be garbage collected when the main level is destroyed. The UWindow system provides a method called GetEntryLevel?() that I use inside the DecalStay? mutator. Be very careful with this, however. If your persistent object references other objects in the main level, those objects will not be garbage collected. This can lead to very subtle and dangerous bugs.

Package flags for package control

Author: Brandon 'GreenMarine' Reinhart
Email: brandon@epicgames.com

Every package has a set of three flags that can be controlled by a mod author. In order to set these flags, you need to be using `ucc make` to compile your package. In the package's classes directory create a file called "MyPackage.upkg" where MyPackage? is the name of the package. Put the following stuff in that file:

```
[Flags]
AllowDownload=False
ClientOptional=False
ServerSideOnly=False
```

When you rebuild your package, the settings you give each of these flags will be saved. AllowDownload? will send your package to clients that connect to a server with that package set as a ServerPackage?. ClientOptional? means that the client doesn't have to have the package in order to connect to the server. A client optional package can be skipped during auto download and the client will still be allowed to connect. ServerSideOnly? indicates whether or not the package should be loaded on the client side when a client joins a server running the package.

Client side mods that add new graphics or sounds will want to turn off ServerSideOnly? and turn off ClientOptional?. Server side mods will want to turn on ServerSideOnly? and turn on ClientOptional?.

Setting Package flags directly: ucc packageflag

Author: Jack 'Mek' Porter
Email: jack@epicgames.com

It is possible to set packageflags on compiled packages using the `ucc packageflag` commandlet. Individual packageflags can be added or removed, type `ucc help packageflag` for information.

Using MS Developer Studio as an UnrealScript? IDE.

Author: Brandon 'GreenMarine' Reinhart
Email: brandon@epicgames.com

With just a little bit of work you can set up Microsoft Developer Studio as an UnrealScript? development environment. The first thing to do is export the UnrealScript? source files. Start UnrealED? and switch the browser bar to "Classes." Hit the Export All button to export the source files to disk.

Now start DevStu? and create a new workspace in your *Unreal Tournament* directory. Add an empty windows DLL project for each source package (Core, Engine, Botpack, etc) and create a folder until the FileView? called Classes. Set the folder's extension type to uc. Finally add each package's source files to their respective folder.

You can compile inside DevStu? by using a custom batch file. Create a MakePackage?.bat inside your UnrealTournament? system directory and add some commands like this:

```
@ECHO OFF
del %1
ucc make
```

This will rebuild the argument (for example, MakePackage? Wookie.u would delete Wookie.u and rebuild it). Now add a custom tool to the tools list in DevStudio?. If you select "Use Output Window" the results of the build will be dumped to the Build window in Dev Studio. Finally, you can attach your custom tool to a custom button and put it on your button bar for fast access.

If you want limited context highlighting, you can run regedit and find the following key:

```
[HKEY_CURRENT_USER\Software\Microsoft\DevStudio\6.0\Text Editor\Tabs/Language Settings\C/C++]
```

Add "uc" to the FileExtensions? list. Make sure DevStudio? is shutdown first, otherwise it'll write over your changes when you exit. Now restart DevStudio? and you've got some limited context highlighting for UnrealScript?!

Using BatchExport?

Author: Nathaniel "^Soul^" Brown
Email: Soul@PlanetUnreal.com

UCC has many commandlets which are interfaces from UCC to Unreal. They are defined in Object->Commandlet. They can be scripted (such as Object->Commandlet->HelloWorldCommandlet) or done in C++, the later of which gives you better access to unreal and its package format. One of the more useful commandlets is BatchExport?. BatchExport? allows you to export all files of a specific type from a .u, .uax, .umx, and .utx. From the command prompt you just type `ucc.exe batchexport` 'Item type' 'File Extension 'path''. The valid types are 'Class', 'Sound', 'Texture', and 'Music'. So say for instance you wanted to export all the class files out of botpack.u without opening UnrealED?, you would go to the command prompt and type `ucc batchexport botpack.u class uc x:\tournament\botpack`.