# UDK to UE4 T3D Tool Documentation

## Table of Contents



## Introduction

The UDK to UE4 T3D tool takes input generated by Unreal engine 3 (UDK) ’s T3D scripting format and reformats and converts the data so that it can be pasted into Unreal Engine 4.
This allows you to port the actors inside of a UE3 level into an UE4 level.

T3D from Unreal 3 if pasted into Unreal 4 simply causes Unreal 4 to crash.

## Supported Actors

Currently the tool supports the following actors & properties:

**(All actors have their Position, Rotation and scale ported over in addition to what's listed below)**

| Actor | Additional Properties | Notes |
|---|---|---|
| **Static meshes** | The Path to static mesh, any and all Actor Material Overrides, Lightmap UV Override, Vertex Colors | See note about Vertex Colors in the Notes section below. |
| **Lights** | Color, Brightness, Attenuation, Spotlight Cone angles | Skylights and Pickup lights are not supported currently.<br>All types (Regular, Moveable toggable, dominant) of all versions (Directional, spot, point) lights are supported. |
| **Interp Actors** | The Path to static mesh, any and all Actor Material Overrides, Lightmap UV Override | Gets turned into a regular static mesh actor set to moveable. |
| **Interactive Foliage Actors** | The Path to static mesh, any and all Actor Material Overrides, Lightmap UV Override | Gets turned into a regular static mesh actor |
| **Kactors** | The Path to static mesh, any and all Actor Material Overrides, Lightmap UV Override | Actors that react to physics are now static mesh actors with settings that enable it. Requires that the static mesh to have collision to work. |
| **Skeletal Meshes** | Path to the skeletal mesh, any and all Actor Material Overrides | |
| **Cameras** | Field of view, Aspect Radio, Constrain Aspect Ratio | |
| **PlayerStarts** | | |
| **Particles** | Path to the Particle | The particles themselves will have to be re-created. |
| **Decals** | Path to the decal material | In UE3, scale was ignored and the size of the decal was determined by the width/height internal properties. In UE4 the scale determines the size of the decal. |
| **Audio** | Path to the Sound Or Cue, Volume Multiplier, Pitch Multiplier, Attenuate, Spatialize, Distance Model, Radius min, Radius Max, Attenuate With LPF, LPFRaius min ,LPFRadius Max, Pitch Min, Pitch Max, Volume Min, Volume Max, High Frequency Gain Multiplier | There is one "AmbientSound" actor in UE4 rather than regular/simple. |
| **Exponential height fog** | Density, Height Falloff, Max Opacity, Start Distance, Opposite light color, Light In scattering Color | "LightInScatteringColor" is now "FogInscatteringColor" and "OppositeLightColor" is now "DirectionalInscatteringColor" |
| **Fractured Static meshes** | The Path to Fractured mesh | Gets turned into the new "Destructible Actor" You'll have to re-create the destructible mesh first in UE4. |
| **Apex Destructible Actors** | The Path to Apex asset | Gets turned into the new "Destructible Actor", The new fractured mesh system is PhysX therefore is Apex. But instead you'll have to re-create the destructible mesh in UE4 rather than the standalone apex tool. |

## Tutorial

1. If you do not have access to the source files themselves, Export all Meshes and Textures out of Unreal Engine 3.
2. Import assets into Unreal Engine 4. It is required that every asset have the same name as it did in UE3. It does not matter the location in the project the file is located.
   In addition to this, you'll need to make sure that every asset has its own unique name.
   It is possible to have two assets in different folders with the same name. If this happens the search matching will end up trying to use the first asset with that name that it finds which may be the wrong asset that it needs.
3. You will have to recreate & assign materials to your meshes. (May be possible in the future to write a converter for materials)
4. Any Particles, Sound Cues, and Destructible mesh assets will have to be re-created in UE4.
5. Start the tool, and click the browse button to browse to the "Content" folder of your Unreal Engine 4 project.
6. Inside of Unreal Engine 3, turn off grouping and "lock prefabs from selection"
7. In UE3, select the actors you want to port over, and Ctrl + C to copy them.
8. Paste the output into the tool and convert it.
9. Copy the converted script to clipboard and paste it into Unreal Engine 4.

## Notes

- The tool will search through your Unreal 4 project and match files by their name. If a match is found it will use that path. If no match is found, or you do not provide a path to the content folder, the tool will fall back to taking the path to the file form Unreal engine 3, and converting it to work in Unreal 4.
  This means it will be looking for that file specifically where it was in UE3. If it is not there you will end up with an empty actor in the scene.
- If the tool converts the path provided by Unreal 3, you can use the "Asset Path" text box to specify a subfolder of where the asset is, ignored if a path from Unreal 4 is found.
- Unreal Engine 4 changed the Unit scaling. One Unreal Unit (UU) is now 1cm, where it was 2cm in UE3. This means that things ported over to UE4 are going to be half the size they should be.
  The tool provides a way to fix this by checking the "Multiply Locations/Scale by 2" options.
  Alternatively you can take every actor ported into UE4 and parent it to something, then changing the scale of the parent to 2.
- **A note on Vertex Colors**: While the tool does support the porting over of Static Mesh vertex colors. There is an issue that may prevent them from working.
  The import/export process of meshes is sometimes not 1 to 1. A mesh that is re-exported out of UDK may not have the same structure as it actually does inside UDK. This means that although the Vertex color data is there the T3D script and is converted, it won't appear up on the UE4 mesh because technically, it's a different mesh now.
  In order to fix this issue, You can try re-importing the mesh back into UDK. This means both engines have the exact same data to work with, then copying the actors.

- Since the “Drawscale” value was removed in UE4 and now there’s only Drawscale3D. The Drawscale value is applied to the Drawscale3D values by multiplying them by the Drawscale. So that they match their correct size.
- Lights in Unreal 3 used their own units for brightness which defaults to 1, and for the radius which defaults to 1024.
  Unreal 4 switched to kelvin units for Brightness which defaults to 5000, and defaults to 1000 for radius.
  In order to persevere the intended effect, the UE3 units are converted so that they match relative to the default values in UE4 (e.g. 0.5 brightness gets converted to 2500).
- Skylights in UE4 have a completely different use than what they did in UE3, no need to convert them. Pickup lights were normal lights but specific to Unreal Tournament pickups.

## Contact

Email: Matt.swanton3d@gmail.com

Website: www.mattswanton.com

Unreal Forum topic:
https://forums.unrealengine.com/showthread.php?3172-Tool-UDK-T3D-to-Unreal-4-T3D-Tool

**Change list:**

Version 1.0 – Initial release that supported only static meshes

Version 1.1 – bug fixes, built-in Scaling assets by 2, Static mesh materials & Override

Version 1.2 – Static Mesh vertex color support

Version 2.0 - Lightmap UVsupport, Lights, Cameras, Decals, Kactors, Skeletal Mesh actors, Interp Actors, Particles, Audio and Exponential height fog actors all supported. Dynamic linking of assets.