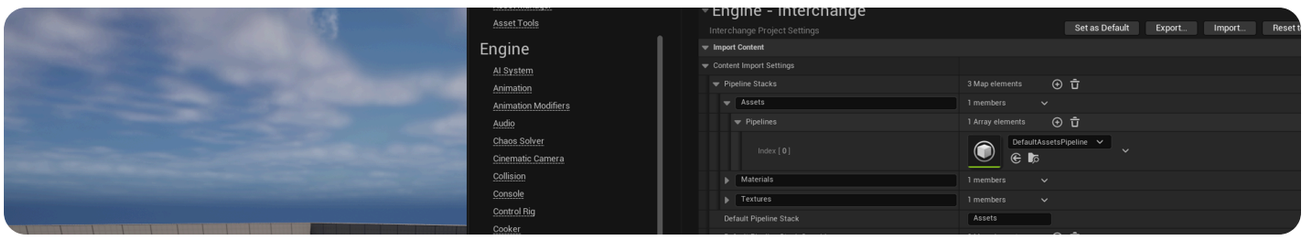
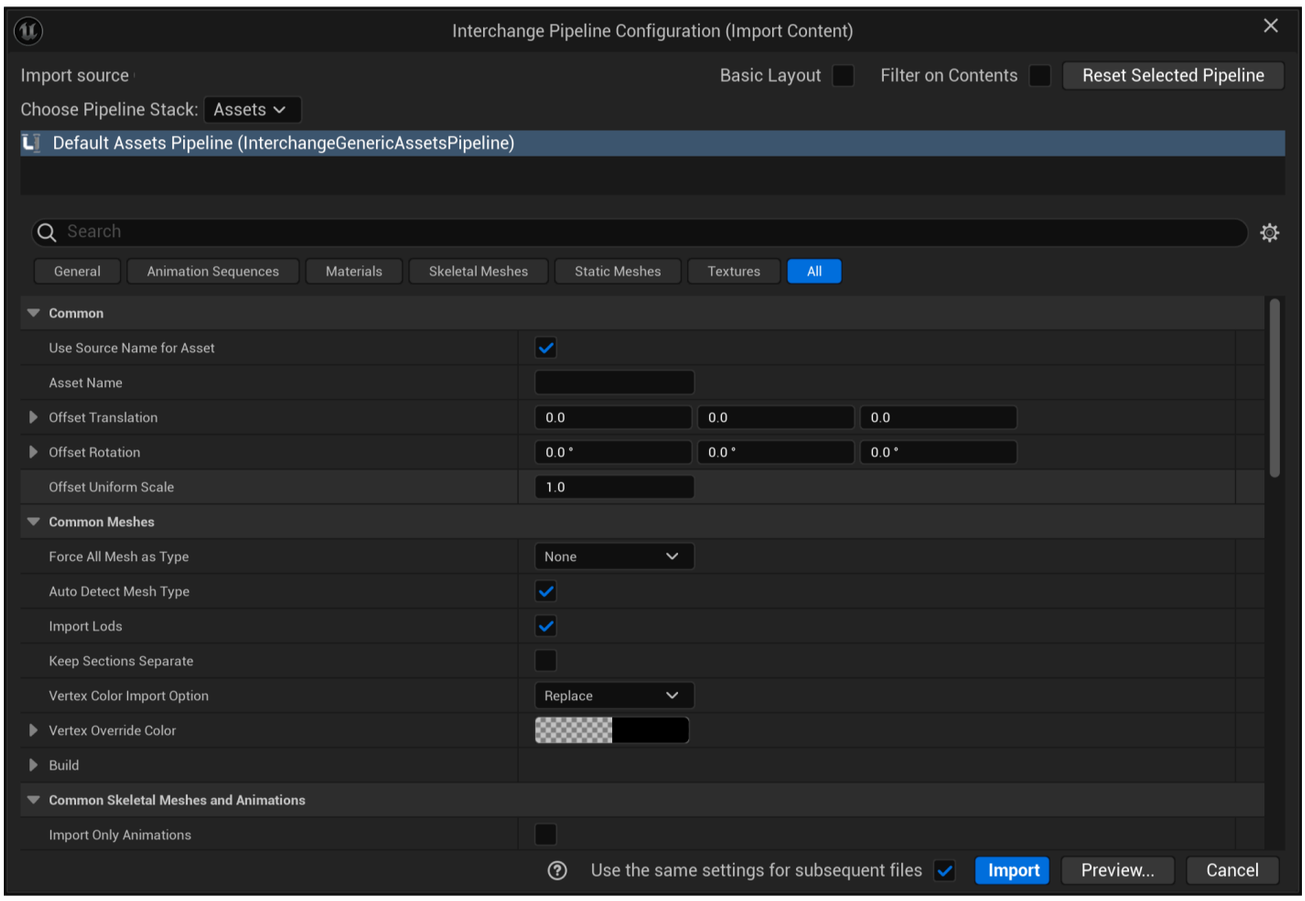
Importing Assets Using Interchange

An overview of the Interchange Framework and how it can be used to customize the import process.



The **Interchange Framework** is Unreal Engine's import and export framework. It is file format agnostic, asynchronous, customizable, and can be used at runtime.



Interchange import interface

Interchange uses a code base that is extensible and provides a customizable pipeline stack. This gives you the freedom to edit the import pipeline using C++, Blueprint, or Python to fit your project's needs.

Important Concepts and Terms

When using Interchange, the following concepts and terms are important:

- **Pipeline**: A collection of operations that process imported data. A pipeline exposes the options used to customize the import process.
- **Pipeline Stack**: An ordered list of pipelines that process an imported file. Pipelines are combined in the stack and assigned to specific file formats. The pipeline stacks are located in **Project Settings > Interchange**.
- **Factory**: An operation that generates the asset from the imported data.

Enable the Interchange Plugins

The Interchange Framework requires the **Interchange Editor** and **Interchange Framework** plugins, which are enabled by default. If these plugins are not enabled in your project, you can enable them in the Project settings for your project.

For more information on enabling plugins, see Working with Plugins.

Import an Asset

Assets are imported into Unreal Engine using one of several different methods.

You can import assets in the Content Drawer or Content Browser, or by selecting **File > Import Into Level**.

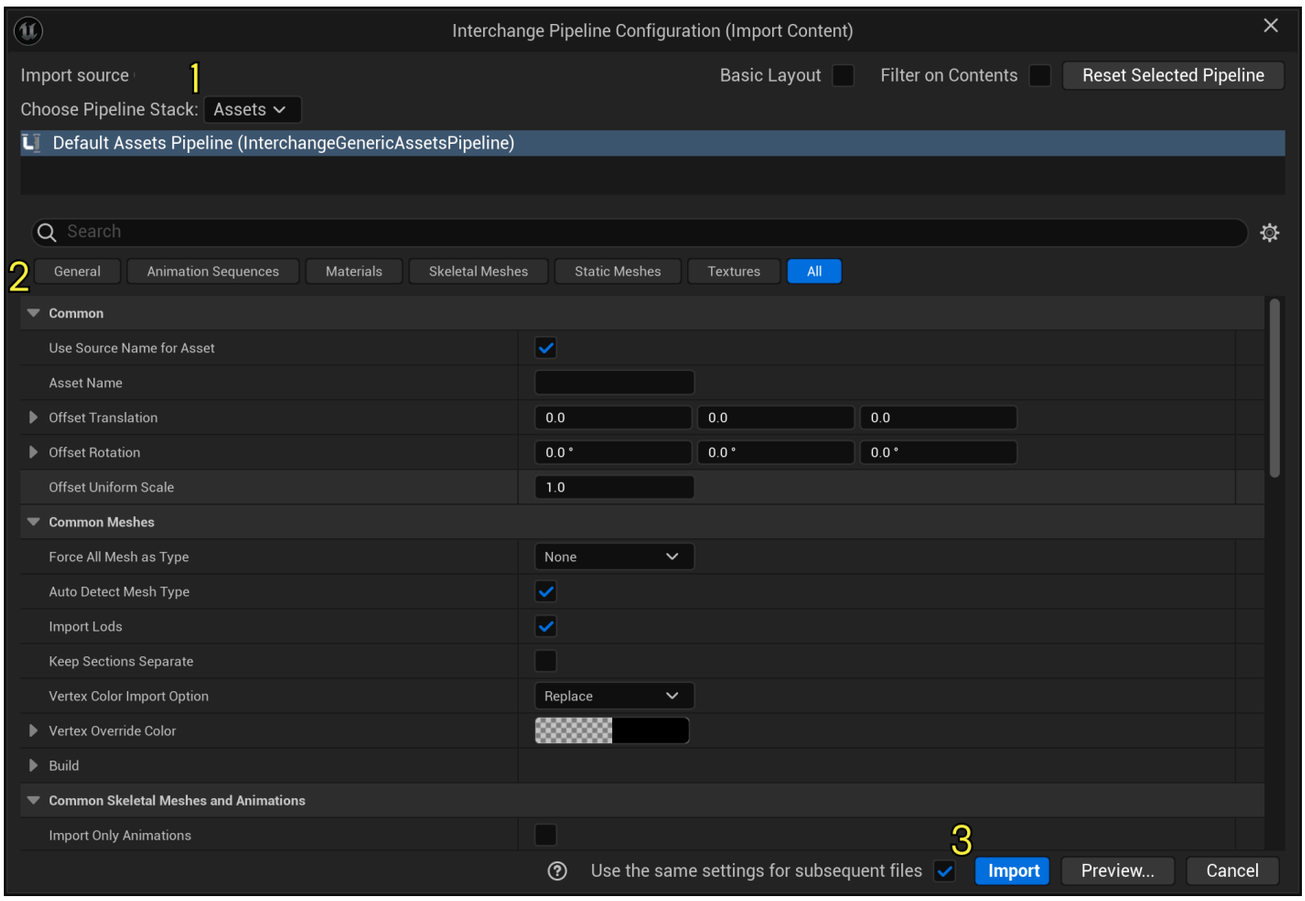
For more information on importing files, see Importing Assets Directly.

> **Import Into Level** currently only works with glTF and **MaterialX** file formats.

Import Process

Begin the import process by using one of the methods listed above.This opens the **Interchange Pipeline Configuration** window:

1. Open the **Choose Pipeline Stack** dropdown menu and select the pipeline stack to use from the list.
2. Configure your settings.
3. Press **Import** to complete the process.



Use the interface to select your import settings and click Import to continue.

With each import, the engine checks whether the file format is supported by the Interchange Framework. If the file is supported, Interchange uses the appropriate import pipeline stack for your format.

Interchange then goes through the following process:

1. Interchange converts the imported data into an intermediary node structure in Unreal Engine.
2. Interchange data processes through the pipeline stack, and follows the instructions for the import.
3. Uses factories to generate the asset from the result.

If the file format is not supported by Interchange, Unreal Engine uses the legacy framework to import the file.

The Interchange Pipeline Configuration window has the following options:

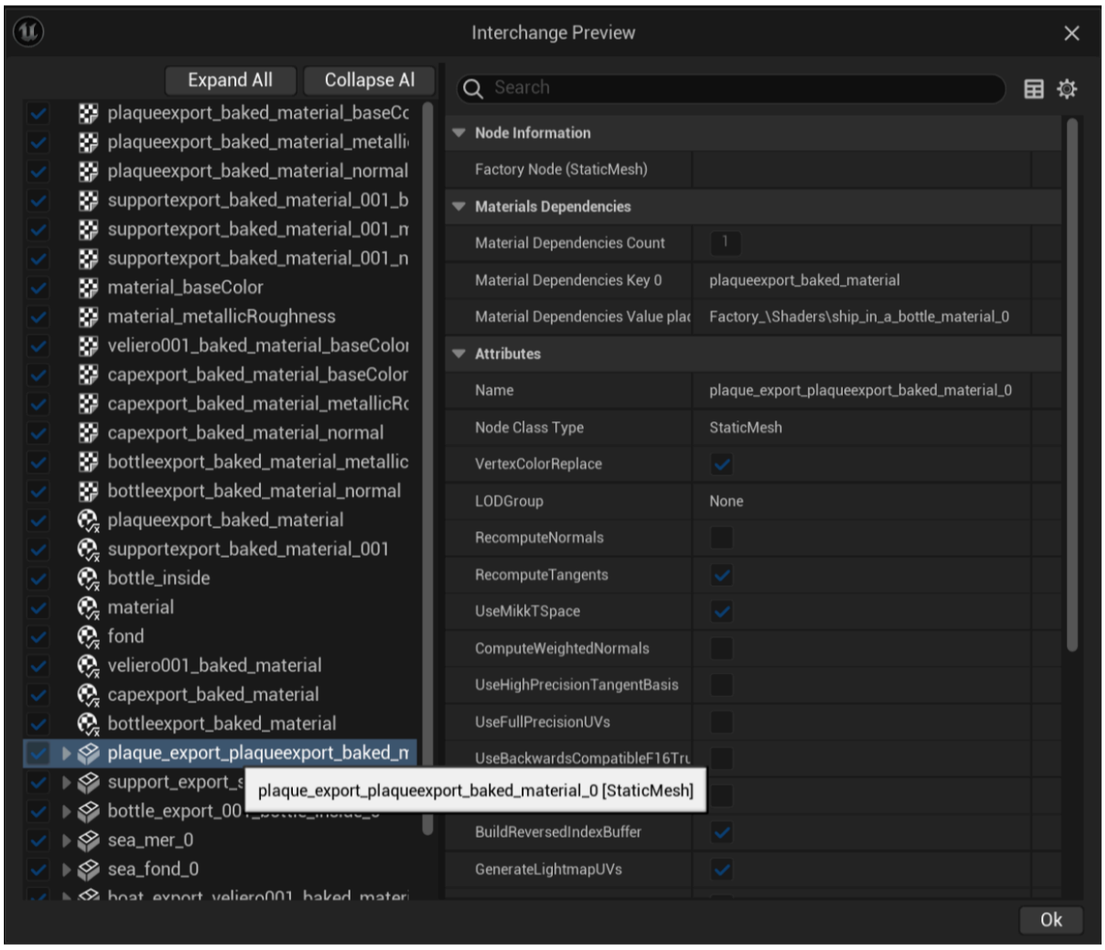
Option	Description
Basic Layout	Filters the import pipeline options down to the basic pipeline properties.
Filter on Contents	Filters the import pipeline options based on the data found in the source file.
Choose Pipeline Stack	Selects that pipeline stack that is used for this import.

Support for the FBX file format is currently Experimental. To enable FBX import using Interchange, use the following console commands:

Console Command	Description
Interchange.FeatureFlags.Import.FBX	Toggles experimental support for FBX import using Interchange.
Interchange.FeatureFlags.Import.FBX.ToLevel	Toggles experimental support for FBX Import Into Level.

Interchange Preview

When you click the **Preview** button in the Interchange Pipeline Configuration window, the Interchange Preview editor window opens:



The Interchange Preview window shows a list of assets that will be created.

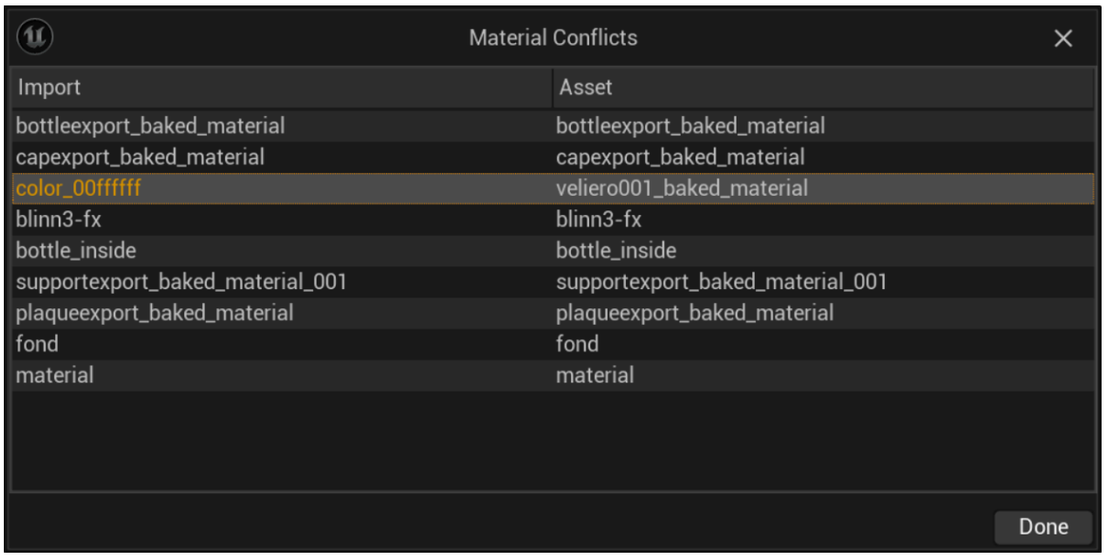
In this window, you can see:

- A list of assets that will be created.
- Their types as icons or in the tooltip text (materials, static mesh, texture2D).
- Their properties are set up by the pre-import step of the pipeline.

Conflicts Preview

If the import process detects changes in the material or skeleton structure of a reimported asset, it highlights the affected pipeline. When you click **Show Conflict**, the Conflicts Preview

window opens:



The Interchange Conflicts Preview window shows changes to the material or skeleton structure on reimport.

This window highlights each conflict to inform you what changes when the asset is reimported.

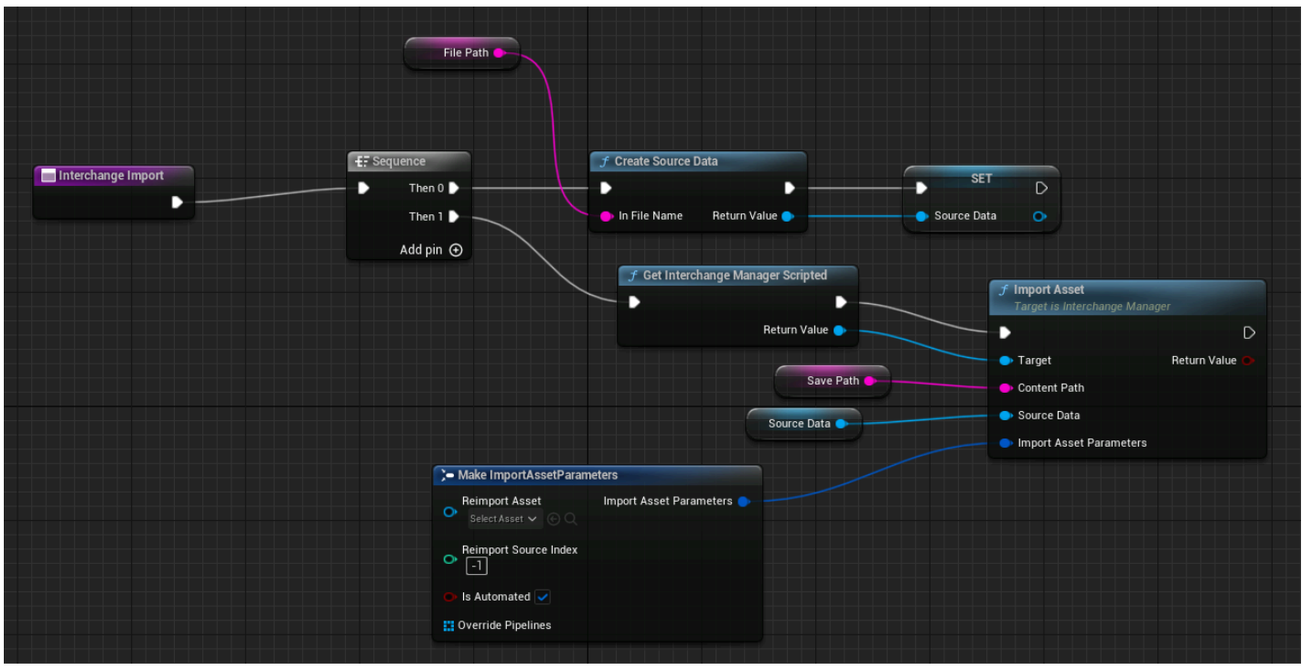
> In previous versions, you could choose to preserve the original material assignment or replace it from the conflict window. You can no longer do this using Interchange. To change the assigned material of an asset, you must make the correction in the source file or use the Static Mesh Editor. For more information on using the Static Mesh Editor, see Applying a Material via the Static Mesh Editor.

Reimporting Assets using Interchange

When you reimport an asset that was previously imported using Interchange, Unreal Engine remembers the pipeline stack and options that were used, and displays those options.

Import an Asset Using Blueprint

You can use Blueprint to import assets into Unreal Engine through the Interchange Framework.



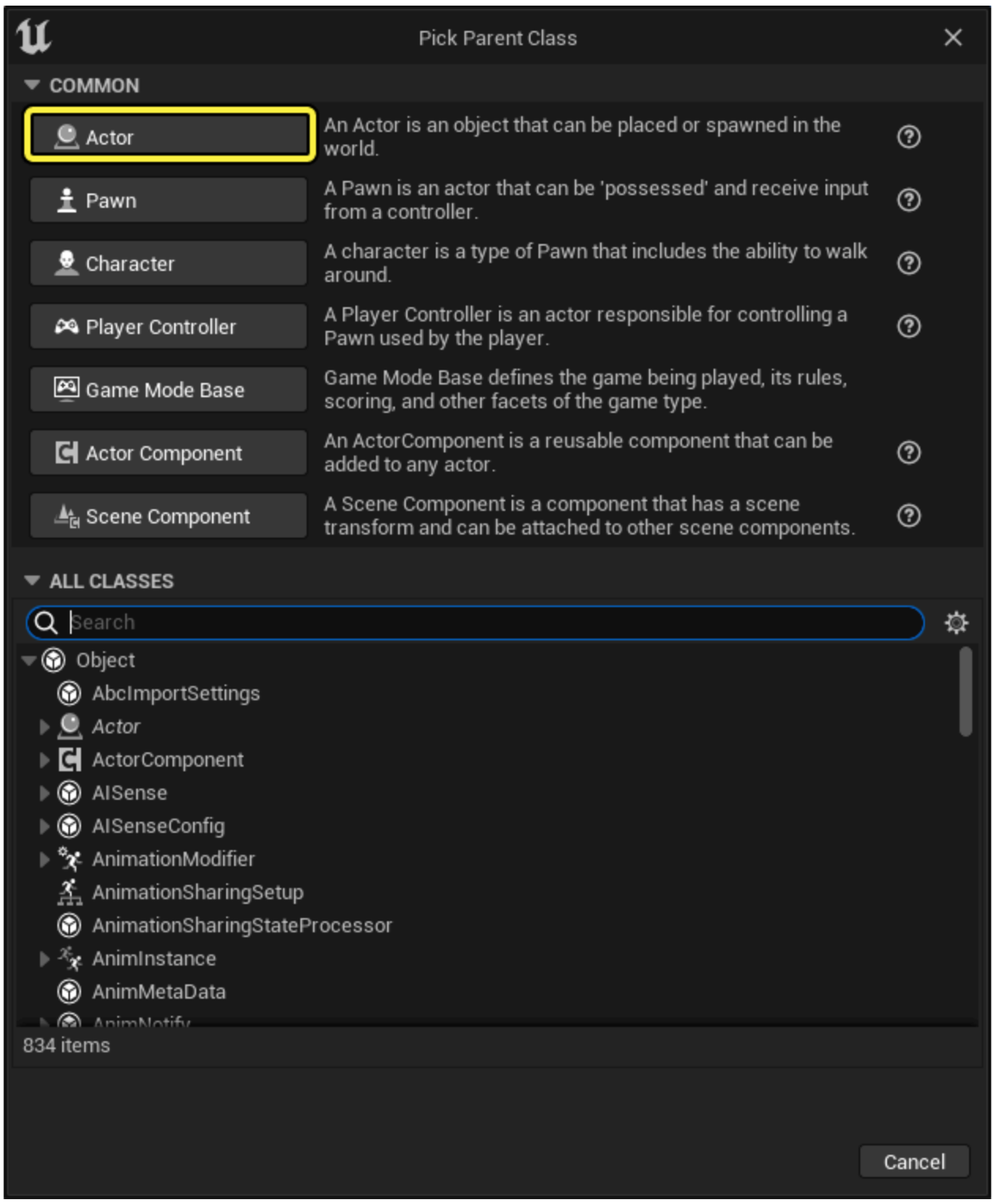
The Blueprint example creates an object that imports files at runtime using Interchange.

For example, you can use Blueprint to import files using Interchange at runtime in an Unreal Engine-based application. The example above creates a function that imports a texture file to a specified file location using the default texture pipeline stack. This method of import currently does not support Skeletal Mesh or Animation data.

Create a new Blueprint Class

To recreate the example, follow the steps below:

1. In your project, create a new Actor Blueprint Class to contain the Function. To create the Actor Blueprint, right-click in the **Content Browser**, navigate to the context menu, and select **Blueprint Class**.
2. In the **Pick Parent Class** window, select **Actor** and name the new Blueprint class **InterchangeActor**.

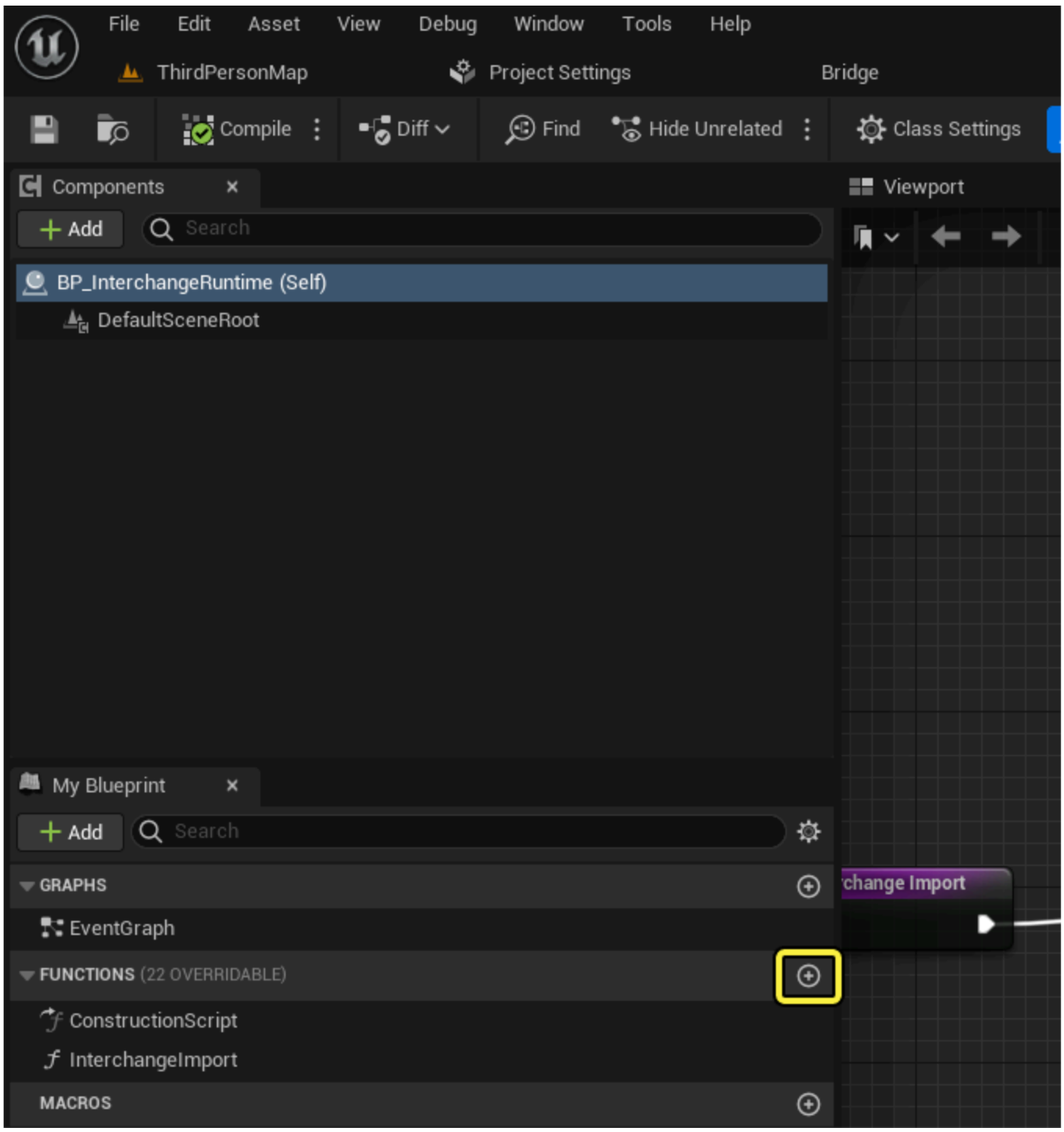


Pick the Parent Class of your new Blueprint.

Add a Function

To add a Function:

1. Double-click the new Blueprint to open the editor.
2. In the **My Blueprint** panel, go to the Functions setting, click the **+** button, and name the new function **InterchangeImport**.



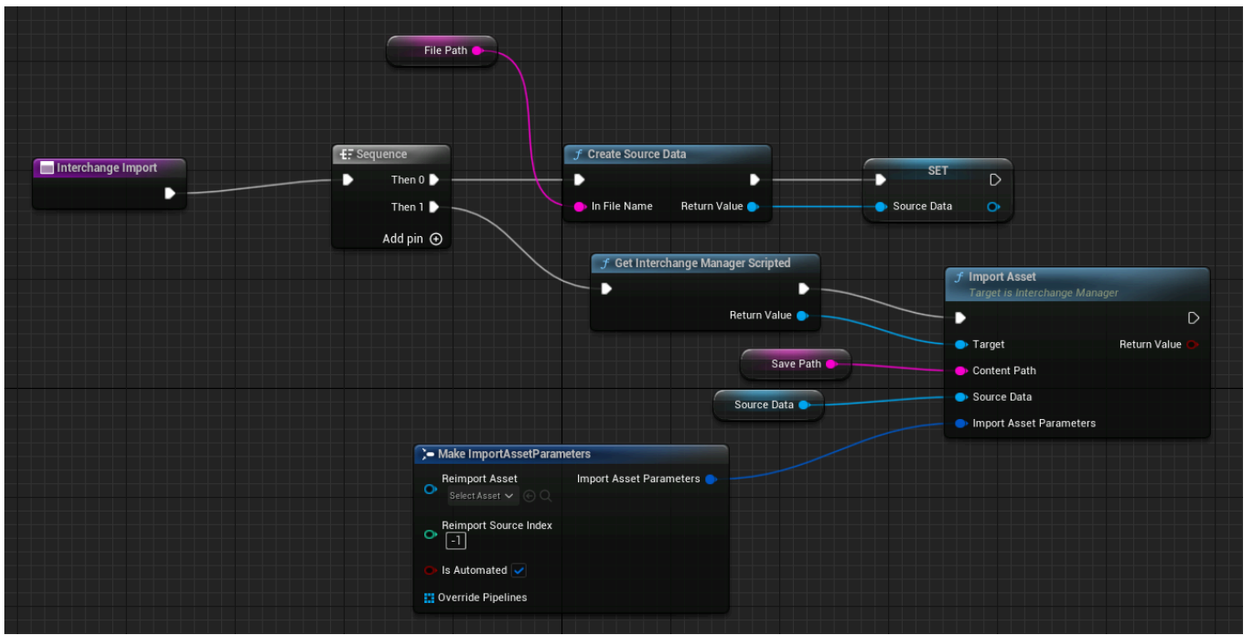
Create a new Function

Add and Connect the Nodes

To add and connect the nodes:

1. Add a **Sequence** node and connect it to the output of the function.
2. Connect the **Then 0** output and create a **Create Source Data** node to reference the existing file that will be imported.
3. Connect the **In File Name** input on **Create Source Data** and, from the context menu, select **Promote to Variable**.
4. Name the new String variable **FilePath**. This holds the location of the file that will be imported.
5. In the blueprint, select the new variable, and click the checkbox for **Instance Editable**.
6. Promote the output of the **Create Source Data** node to a new variable named **SourceData**.
7. Drag from the **Then 1** output on the Sequence and create a **Get Interchange Manager Scripted** node. This creates a pointer to the Interchange Manager that is used in the next step.

8. Drag from the **Get Interchange Manager Scripted** output and create an **Import Asset** node. Connect the Return Value from **Get Interchange Manager Scripted** to the **Target input** on **Import Asset**.
9. Drag off from the **Content Path** input and promote it to a new variable named **SavePath**. This holds the location of the newly imported file.
10. In the blueprint, select the new variable and select the checkbox for **Instance Editable**.
11. Get a reference to the **Source Data** variable and connect it to the Source Data input on **Import Asset**.
12. Drag off from the **Import Asset Parameters** input and create a **Make Input Asset Parameters** node.



Click image for full size.

Make the Function Available at Runtime

To make the function available at runtime:

1. In **My Blueprints**, click on the **InterchangeImport** function, and click the checkbox next to **Call In Editor** in the **Details** panel. This option makes the function available at runtime in the **Details** of the **InterchangeActor** object.
2. **Save** and **Compile** your Blueprint.

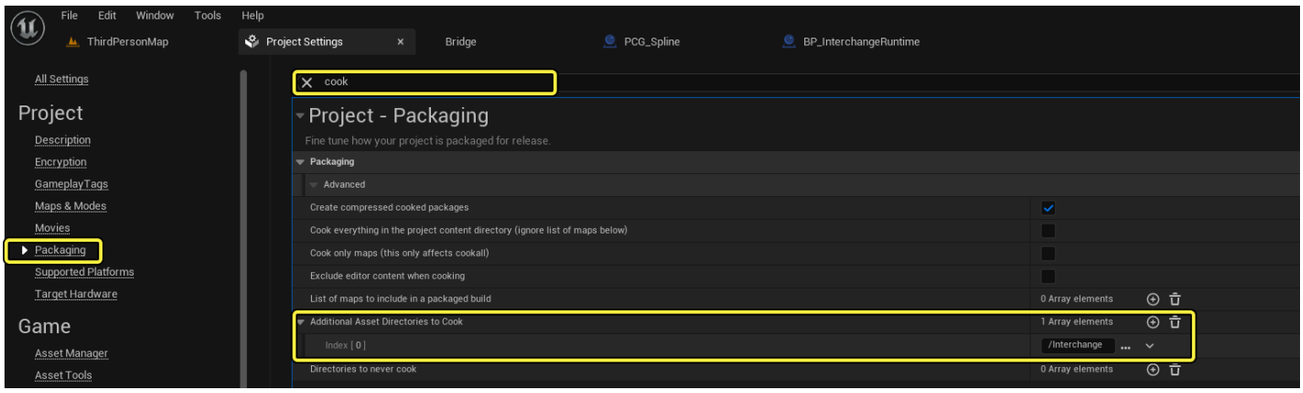
Use Your New Blueprint

1. Drag a copy of the InterchangeActor blueprint into the level.
2. Click **Play**.
3. In the **Outliner**, select the **InterchangeActor**.
4. In the **Details** panel, fill in the **FilePath** and the **SavePath**.
5. Click the **Interchange Import** button to import your file.

If you use an **Import Scene** node with the Blueprint example above, the asset spawns directly into the scene.

Using Interchange in a Cooked Application

If you plan to use the Interchange Framework at runtime in a cooked application, go to the **Project Settings > Project - Packating > Additional Asset Directories to Cook**, and add the **Interchange** folder.



Click image for full size.

Import an Asset Using Python

You can use Python script to import assets into Unreal Engine through the Interchange Framework.

```python
import unreal

import_path = "C:/Users/foo/Downloads/Fbx/SkeletalMesh/Animations/Equilibre.f

import_extension = unreal.Paths.get_extension(import_path, False)

is_gltf = import_extension == 'glb' or import_extension == 'gltf'

is_fbx = import_extension == 'fbx'

#if you want to import fbx file make sure interchange fbx import is enable
if is_fbx:
level_editor_subsystem = unreal.get_editor_subsystem(unreal.LevelEditorSubsys
unreal.SystemLibrary.execute_console_command(level_editor_subsystem.get_worlc
true')

editor_asset_subsystem = unreal.get_editor_subsystem(unreal.EditorAssetSubsys

transient_path = "/Interchange/Pipelines/Transient/"
transient_pipeline_path = transient_path + "MyAutomationPipeline"

editor_asset_subsystem.delete_directory(transient_path)
```

```
22
23 #Duplicate the default interchange asset content pipeline, gltf have a specia
24 if is_gltf:
25 pipeline = editor_asset_subsystem.duplicate_asset("/Interchange/Pipelines/Def
   transient_pipeline_path)
26 else:
27 pipeline = editor_asset_subsystem.duplicate_asset("/Interchange/Pipelines/Def
   transient_pipeline_path)
28
29 #Set any pipelines properties you need for your asset import here
30
31 #force static mesh import
32 pipeline.common_meshes_properties.force_all_mesh_as_type = unreal.Interchange
33 #combine static mesh
34 pipeline.mesh_pipeline.combine_static_meshes = True
35 #Prevent Material import
36 pipeline.material_pipeline.import_materials = False
37 #Prevent Texture import
38 pipeline.material_pipeline.texture_pipeline.import_textures = False
39
40 #Create a source data from the filename
41 source_data = unreal.InterchangeManager.create_source_data(import_path)
42 #create the parameters for the interchange import
43 import_asset_parameters = unreal.ImportAssetParameters()
44 #Script is normaly an automated import
45 import_asset_parameters.is_automated = True
46
47 #Add the configured pipeline to the import arguments
48 import_asset_parameters.override_pipelines.append(unreal.SoftObjectPath(trans
   ".MyAutomationPipeline"))
49 #gltf importer use 2 pipeline add the second one
50 if is_gltf:
51 import_asset_parameters.override_pipelines.append(unreal.SoftObjectPath("/Int
52
53 interchange_manager = unreal.InterchangeManager.get_interchange_manager_scrip
54 #import the asset
55 interchange_manager.import_asset("/game/AA0A/testpython/",source_data,import_
56
57 editor_asset_subsystem.delete_directory(transient_path)
```

Copy full snippet

In the above example, a Python script is used to import the `Equilibre.fbx` file. The script checks to see if the file format is `.gltf` or `.fbx` and then assigns the correct pipeline.

Edit the Pipeline Stack

One of the advantages of the Interchange Framework is the ability to select and customize the pipeline stack, a customizable stack of processes that process the asset data is processed. You can add pipelines to the default pipeline stack to add behaviors during the import process.

Unreal Engine ships with the following default pipelines:

- Default Assets Pipeline
- Default Material Pipeline
- Default Texture Pipeline
- Default Scene Assets Pipeline
- Default Scene Level Pipeline
- Default Graph Inspector Pipeline

Each default pipeline contains the most common options used for that type of import. You can customize these pipelines to meet the specific needs of your project.

Edit an Existing Pipeline

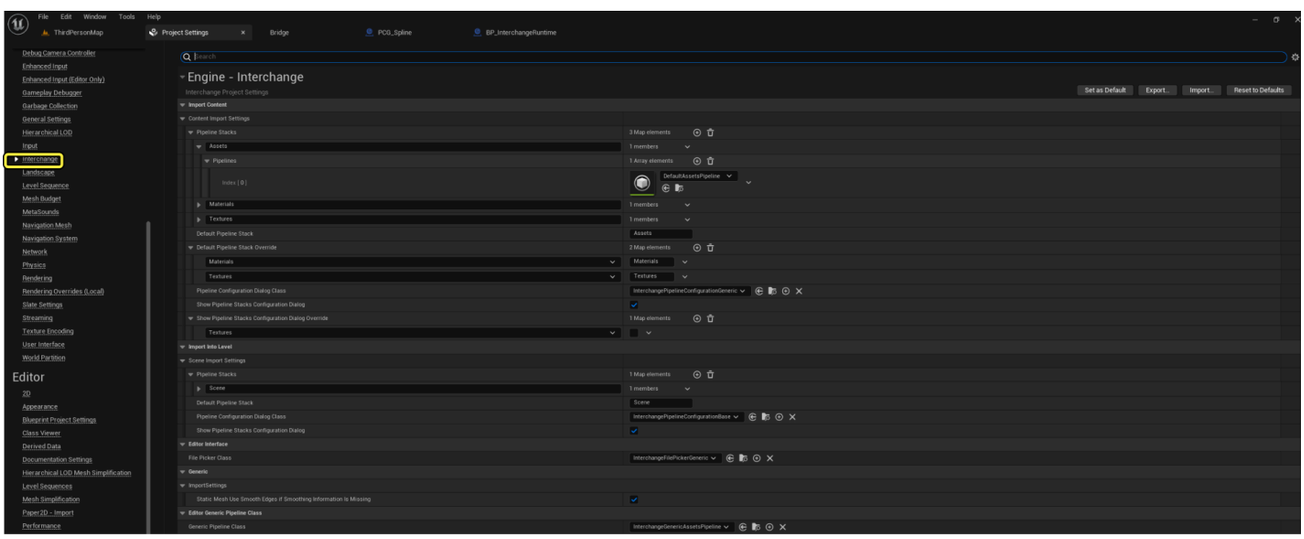
Each of the default pipelines is customizable to fit the needs of your project and team.

The following are methods for customizing the import options for your project:

- Add, remove, or reorder the existing pipeline stack in your **Project Settings**.
- Change which pipelines are used by default.
- Modify the existing default pipelines.
- Create a custom pipeline.

Edit the Project Settings

You can find the pipeline stack in the **Project Settings** under **Engine > Interchange**:



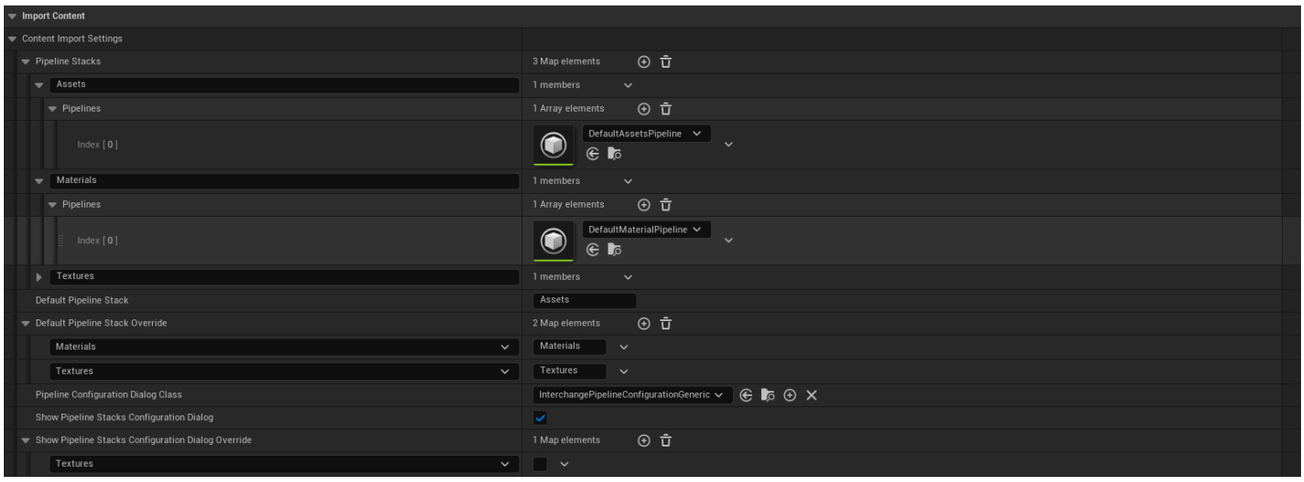
The pipeline stack in Project Settings.

The pipeline stack contains the default settings for:

- Import Content
- Import Into Level
- Editor Interface
- Generic
- Editor Generic Pipeline Class

Import Content

Unreal Engine uses these settings to import content into the **Content Drawer** or **Content Browser**.

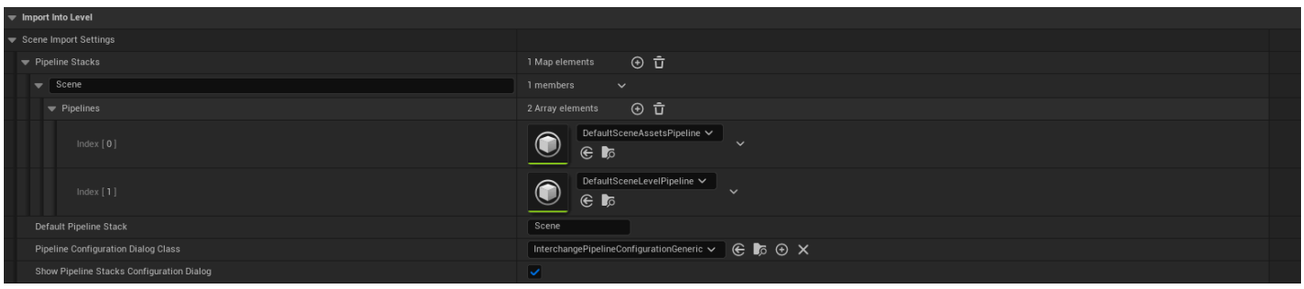


Click image for full size.

You can alter the settings for each type of content listed. You can also add additional headings if needed. For example, the default configuration contains **Assets**, **Materials**, and **Textures**. You could add an additional section to the pipeline stack for Animations and then be able to add one or more custom pipelines to handle incoming animation files.

Import Into Level

In the Editor window, you can find the **File > Import Into Level**. By default, this function uses two pipelines that work together. These pipelines import the Actor data from a file and then spawn an Actor in the level. The import function uses the following settings:



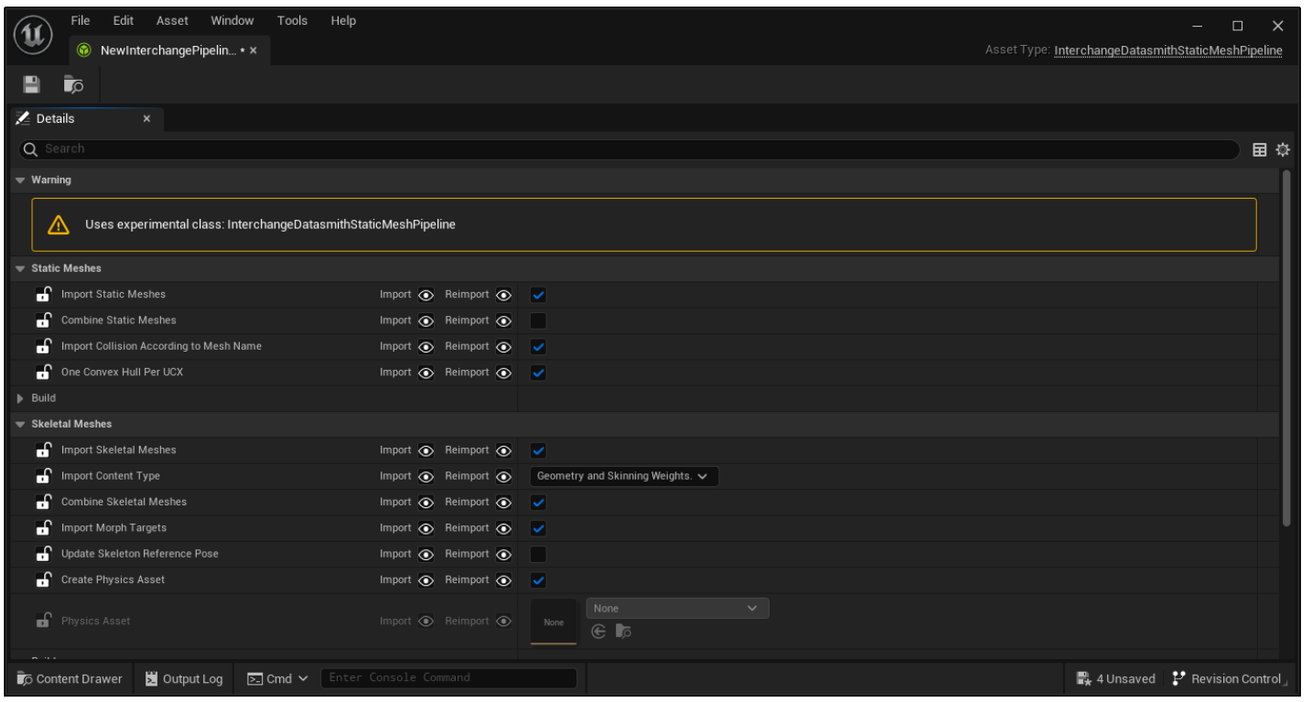
Click image for full size.

- **DefaultSceneAssetPipeline** is based on the same class as the DefaultAssetPipeline and is designed for scene import.
- **DefaultSceneLevelPipeline** generates the Actor in the world after the data passes through the DefaultSceneAssetPipeline.

Modify the Existing Default Pipelines

You can modify the properties of the default Interchange Pipelines to change the following:

- Default Values
- Visibility
- Read-only status



The Interchange Pipeline Details panel.

To change the settings of the default Interchange Pipelines, follow the steps below:

1. Locate the default pipelines in the Content Drawer or Content Browser and double click one to open it. The pipelines are located in the **Engine > Plugins >Interchange Framework Content > Pipelines** folder. If you are unable to see the Engine folder, click on **Settings** in the top right corner of the **Content Drawer** or **Content Browser** and select the checkbox for **Show Engine Content**.
2. Edit the following as needed:
    a. Visibility during the import and reimport process.
    b. Default setting.
    c. Whether the property is read only during the import process.
3. Save and close the window.

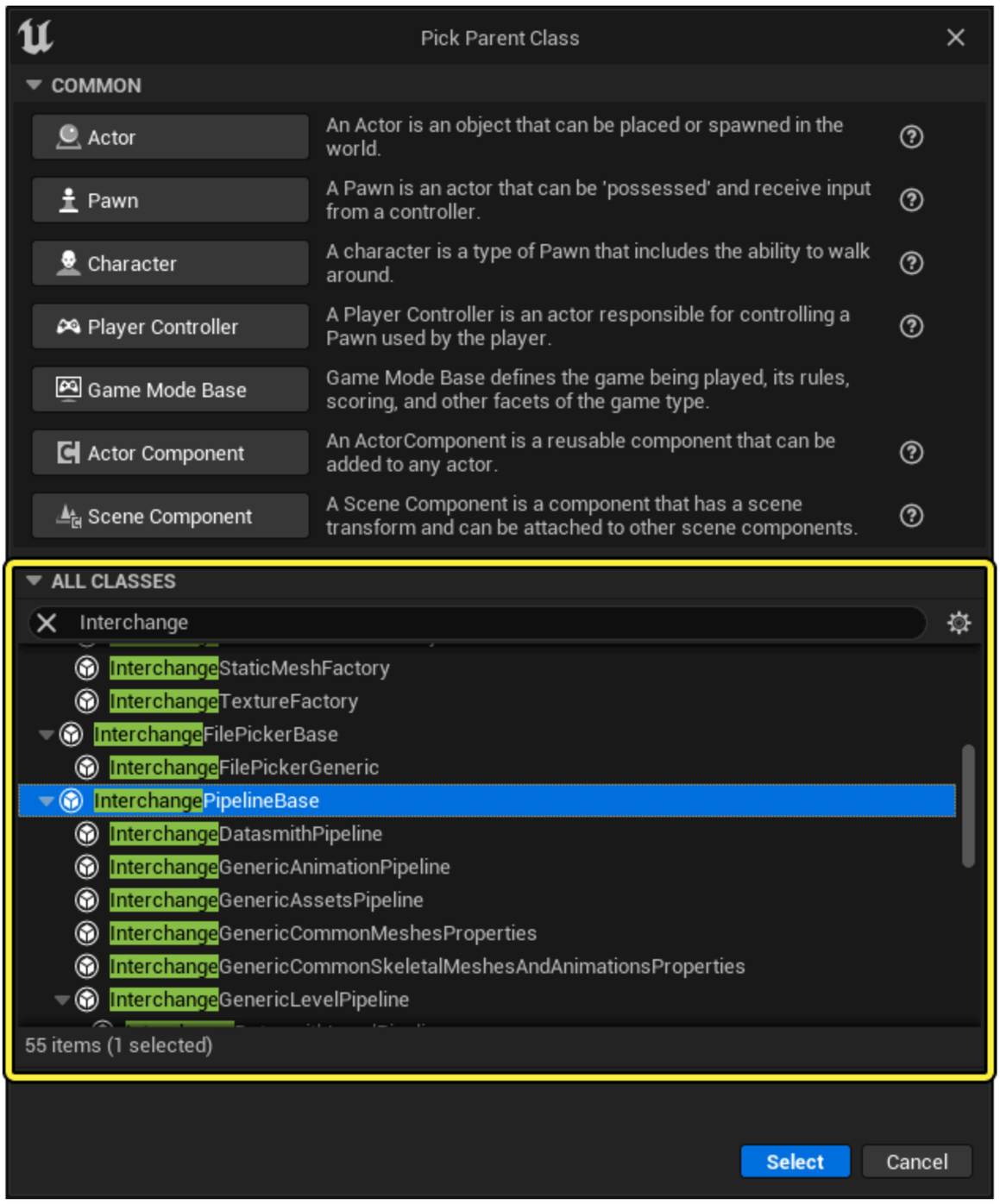
Create a Custom Pipeline

You can create new Interchange Pipelines to further customize the import process using Blueprints, C++, or Python.

Create a Custom Pipeline Using Blueprints

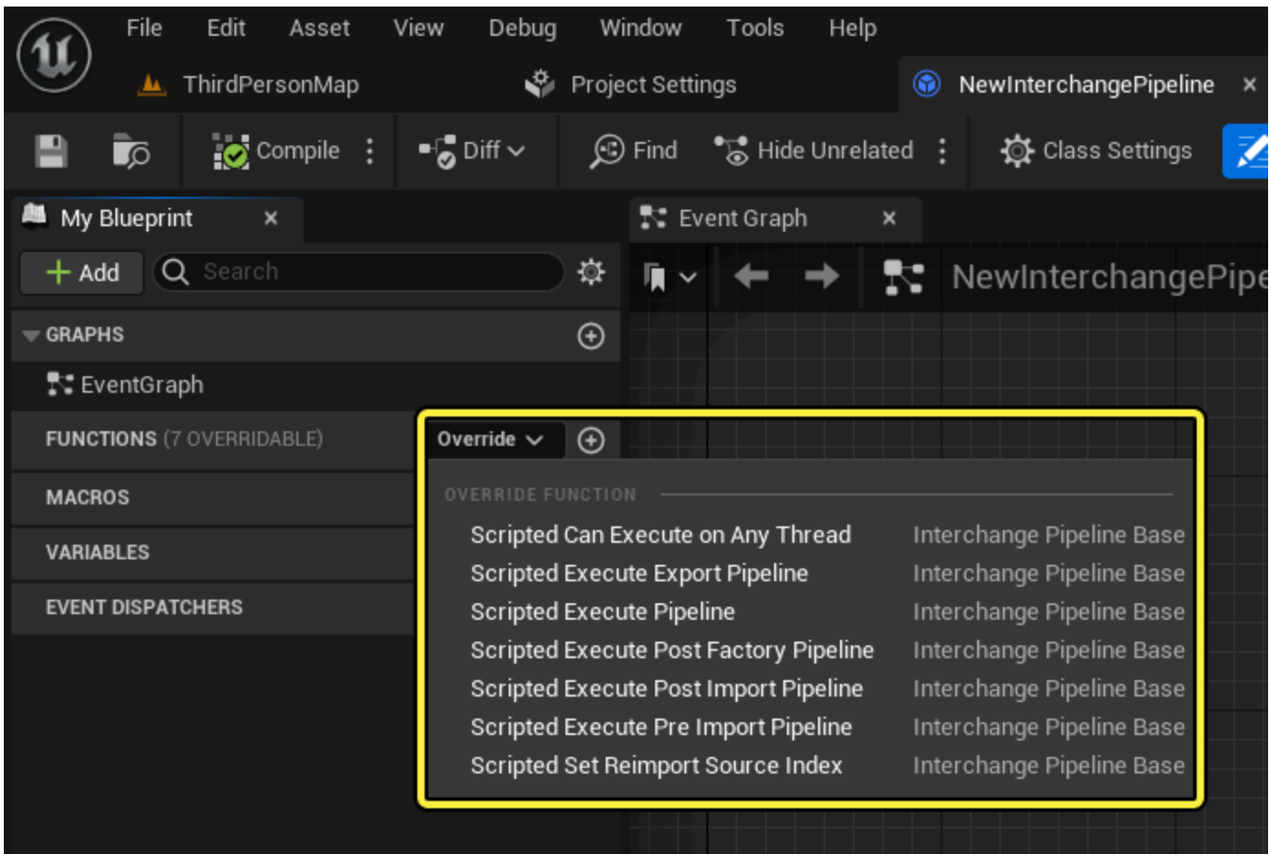
To create a new Interchange Pipeline using Blueprints, follow the steps below:

1. In the **Content Drawer** or **Content Browser**, right-click and select **Create Blueprint Class**.
2. In the P**ick Parent Class window**, expand the **All Classes** category, and select **InterchangePipelineBase** as its **Parent Class**.
3. Double-click the new Blueprint to open the **Blueprint Editor**.



Select InterchangePipelineBase as the Parent Class.

A custom pipeline created using Blueprints has the following functions that can be overridden to add custom behavior.



Interchange Blueprint Override Functions.

Override Function	Description
Scripted Can Execute on Any Thread	Communicates to the Interchange Manager that this pipeline can be executed in async mode.
Scripted Execute Export Pipeline	Executes during the export process (feature is currently nonfunctional).
Scripted Execute Pipeline	Executes after file translation. Creates the factory needed to generate Assets.
Scripted Execute Post Factory Pipeline	Executes after the factory creates an asset, but before the PostEditChange function is called.
Scripted Execute Post Import Pipeline	Executes after the Asset is completely imported and the PostEditChange function is called.
Scripted Set Reimport Source Index	Executes and tells the pipeline which source index to reimport. Use this function when reimporting an Asset that can have more than one source. For example, a Skeletal Mesh that has one source file for geometry and another for skinning information.

Create a Custom Pipeline Using C++

To create a new Interchange pipeline using C++, Create a header file that contains the following:

```h
#pragma once

#include "CoreMinimal.h"

#include "InterchangePipelineBase.h"
#include "InterchangeSourceData.h"
#include "Nodes/InterchangeBaseNodeContainer.h"

#include "InterchangeMyPipeline.generated.h"

UCLASS(BlueprintType)
class INTERCHANGEPIPELINES_API UInterchangeMyPipeline : public UInterchangePipelineBase
{
GENERATED_BODY()


protected:
virtual void ExecutePipeline(UInterchangeBaseNodeContainer* BaseNodeContainer, const TArray<UInterchangeSourceData*>& SourceDatas) override;

virtual bool CanExecuteOnAnyThread(EInterchangePipelineTask PipelineTask) override
{

return true;
}
};

```

Copy full snippet

Next, create a source file that contains the following:

```cpp
#include "InterchangeMyPipeline.h"


void UInterchangeMyPipeline::ExecutePipeline(UInterchangeBaseNodeContainer* NodeContainer, const TArray<UInterchangeSourceData*>& InSourceDatas)
{
Super::ExecutePipeline(NodeContainer, InSourceDatas);

// Put the logic you need on either translated nodes or factory nodes
}

```

Copy full snippet

For more information on working with C++ in Unreal Engine, see Programming with C++.

Create a Custom Pipeline Using Python

To create a new Interchange Pipeline using Python script, create a new Python script and use the Project settings to add it to the Startup Scripts. For more information on working with Python scripting in Unreal Engine, see Scripting the Unreal Editor Using Python.

In the example script below, Python script is used to create a basic asset import pipeline.

```python
import unreal

@unreal.uclass()
class PythonPipelineTest(unreal.InterchangePythonPipelineBase):

import_static_meshes = unreal.uproperty(bool,meta=dict(Category="StaticMesh"))
import_skeletal_meshes = unreal.uproperty(bool,meta=dict(Category="SkeletalMesh"))

def cast(self, object_to_cast, object_class):
try:
return object_class.cast(object_to_cast)
except:
return None

def recursive_set_node_properties(self, base_node_container, node_unique_id):
node = base_node_container.get_node(node_unique_id)
# Set the static mesh factory node enable state
static_mesh_node = self.cast(node, unreal.InterchangeStaticMeshFactoryNode)
if static_mesh_node:
static_mesh_node.set_enabled(self.import_static_meshes)
# Set the skeletal mesh factory node enable state
skeletal_mesh_node = self.cast(node, unreal.InterchangeSkeletalMeshFactoryNode)
if skeletal_mesh_node:
skeletal_mesh_node.set_enabled(self.import_static_meshes)
# Iterate children
childrens = base_node_container.get_node_children_uids(node.get_unique_id())
for child_uid in childrens:
self.recursive_set_node_properties(base_node_container, child_uid)

@unreal.ufunction(override=True)
def scripted_execute_pipeline(self, base_node_container, SourceDatas):
root_nodes = base_node_container.get_roots()
for node_unique_id in root_nodes:
self.recursive_set_node_properties(base_node_container, node_unique_id)
return True
```

Copy full snippet