Python Scripting in Sequencer

Learn common Python scripting commands and features used with Sequencer.

Python scripting can be used to automate and control various parts of **Sequencer**. This document provides an overview of the main ways to use Python with Sequencer, and examples of general Sequencer scripting.

Prerequisites

- You have some experience with Python scripting in Unreal Engine.
- You have an understanding of how to use Sequencer.

Sequencer Python Terms

Sequencer uses the following terminology:

Sequencer Python Terms	Description
World	An object that represents a map in which actors and components can exist and be rendered (also referred to as a

Sequencer Python Terms	Description
	Level).
LevelSequence	An asset that is a container for cinematics scenes (also referred to as a **Sequence**). Level Sequences contain data and tracks that can be bound to different objects to animate them.
MovieSceneBindingProxy	A struct defining the actor or component that is bound by a Level Sequence (also referred to as a **Binding**).
Possessable	A type of binding describing an actor or component that exists in a Level, in which the Level Sequence can own any animatable properties.
Spawnable	A type of binding describing an actor or component that exists only while the Sequence is playing.
MovieSceneTrack	An object located under a **Binding** (MovieSceneBindingProxy), which contains all sections of edits for a specific typed property. For example, **MovieScene3DTransformTrack > Actor / Component Transform**.
MovieSceneSection	An object located under a Track (MovieSceneTrack), which contains all channels, length, and parameters for a specific typed property. For example, **MovieScene3DTransformSection > Pre / Post Roll, When Finished State, Active/Muted, Additive**.
MovieSceneScriptingChannel	An object located under a Section (MovieSceneSection) which contains all keyframes that animate a specific typed property or sub-property. For example, **MovieSceneScriptingFloatChannel > Location.X**.
MovieSceneScriptingKey	An object that represents a keyframe in the specifically typed channel. For example, **MovieSceneScriptingFloatKey**.

Sequencer Python Terms	Description
FrameNumber	A struct that represents a frame.
FrameRate	A struct that represents a fraction of 2 integers defining frames and seconds. For example, 30 frames per second is written as: **30/1**.

Accessing a Level Sequence

The first step when Python scripting in Sequencer is to access the **LevelSequence**, which is the main object you'll be interacting with. There are several ways to do that depending on your situation.

Simple Access

To access a Level Sequence that exists in the **Content Browser**, you can use the following example script. The sequence does not have to be currently opened or exist in your current Level. This script assumes the Level Sequence Asset is located in the root content folder.

```python
import unreal

# Get a level sequence asset

level_sequence = unreal.load_asset("/Game/LevelSequenceName")

# Then open it in Sequencer

unreal.LevelSequenceEditorBlueprintLibrary.open_level_sequence(level_sequence

```

Copy full snippet

Access Current Level Sequence

You can also access a currently-opened Level Sequence using the following script:

```
import unreal

# Get the currently opened Level Sequence

level_sequence = unreal.LevelSequenceEditorBlueprintLibrary.get_current_level_sequence()

```

Copy full snippet

Create and Open Level Sequence

You can use the following script to create a new Level Sequence Asset and open it:

```
import unreal

# Get asset tools

asset_tools = unreal.AssetToolsHelpers.get_asset_tools()

# Create a Level Sequence with name LevelSequenceName in root content folder

level_sequence = unreal.AssetTools.create_asset(asset_tools, asset_name = "LevelSequenceName", package_path = "/Game/", asset_class = unreal.LevelSequence, factory = unreal.LevelSequenceFactoryNew())

```

Copy full snippet

Get and Set the Currently Viewed Level Sequences

You can use the following scripts to get or set the level sequence of focus in Sequencer:

```
# Get the current level sequence of focus

```

```
focused_level_seqeunce =
unreal.LevelSequenceEditorBlueprintLibrary.get_focused_level_sequence()

```

Copy full snippet

You can also focus on a specific sub sequence by providing the sub sequence section to focus on:

```
# Get the first sub sequence track and get the first section from the level s

sub_sequence_track = level_sequence.find_tracks_by_type(unreal.MovieSceneSubT

sub_sequence_section = sub_sequence_track.get_sections()[0]

# Set the current level sequence of focus

unreal.LevelSequenceEditorBlueprintLibrary.focus_level_sequence(sub_sequence_

```

Copy full snippet

To focus back on a parent sequence, all you would need to do is run this command:

```
unreal.LevelSequenceEditorBlueprintLibrary.focus_parent_sequence()

```

Copy full snippet

Querying and Editing a Level Sequence

Once you have access to a Level Sequence in Python, you can perform changes to it. There are a variety of ways to affect your sequence, and some examples will be provided below.

Change Frame Rate

By default, Level Sequences play at a rate of 30 frames per second (fps). To change this playback rate, you can use the following commands:

```
# Create a frame rate object and set to the desired fps number

frame_rate = unreal.FrameRate(numerator = 60, denominator = 1)

# Set the display rate

level_sequence.set_display_rate(frame_rate)

```

Copy full snippet

Change Start and End Times

By default, A sequence's playback range is set to start at frame 0 and end at frame 150 (Assuming the frame rate is 30fps). You can adjust both the start and end frames with the following commands:

```
# Set the playback range to 20-200

level_sequence.set_playback_start(20)

level_sequence.set_playback_end(200)

```

Copy full snippet

Adding Actors

To add an actor from your current Level for Sequencer to possess, use the following commands:

```
# Get the Actor and Level Sequence Editor subsystems

actor_system = unreal.get_editor_subsystem(unreal.EditorActorSubsystem)
```

```

ls_system = unreal.get_editor_subsystem(unreal.LevelSequenceEditorSubsystem)

# Add selected actors to current level sequence as possessables

actors = actor_system.get_selected_level_actors()

bindings = ls_system.add_actors(actors)

```

Copy full snippet

To add a Camera object through python scripting, mimicking the process through the Sequencer UI, use the following command :

```
# Get the Level Sequence Editor subsystem

ls_system = unreal.get_editor_subsystem(unreal.LevelSequenceEditorSubsystem)

# Add a spawnable camera actor binding, with a camera cut track

camera = ls_system.create_camera(spawnable = True)

```

Copy full snippet

Instead of possessing an actor present in your Level, you can spawn actors using python you can use for the duration of the sequence. Using the previously mentioned **add_actors** in the **LevelSequenceEditorSubsystem**, you can first use a command to make the actor possessables and then convert it to a spawnable object. Use the following commands to add a spawnable to your sequence:

```
# Get the Actor and Level Sequence Editor subsystems

actor_system = unreal.get_editor_subsystem(unreal.EditorActorSubsystem)

ls_system = unreal.get_editor_subsystem(unreal.LevelSequenceEditorSubsystem)

# Add selected actors to current level sequence as possessables

```

```
actors = actor_system.get_selected_level_actors()

bindings = ls_system.add_actors(actors)

# Loop through all the added bindings

```

Copy full snippet

For binding in bindings:

```
# Convert to spawnable

ls_system.convert_to_spawnable(binding)

```

Copy full snippet

If necessary, you can additionally convert spawnables back to possesables with the following command using the **LevelSequenceEditorSubsystem**:

```
ls_system = unreal.get_editor_subsystem(unreal.LevelSequenceEditorSubsystem)

selected_bindings = 
unreal.LevelSequenceEditorBlueprintLibrary.get_selected_bindings()

```

Copy full snippet

For binding in selected_bindings:

```
ls_system.convert_to_possessable(binding)

```

Copy full snippet

Creating Tracks and Sections

You can also add Tracks and Sections through Python scripting, with each track type informing the section type. For example:

- Transform Tracks are defined as `unreal.MovieScene3DTransformTrack` and their sections use `unreal.MovieScene3DTransformSection`.
- Skeletal Mesh Animation Tracks are defined as `unreal.MovieSceneSkeletalAnimationTrack` and their sections use `unreal.MovieSceneSkeletalAnimationSection`.

To add Tracks and Sections, use the following commands:

```python
# Use the binding to add tracks into sequencer - specified by track type

transform_track = actor_binding.add_track(unreal.MovieScene3DTransformTrack)

anim_track = actor_binding.add_track(unreal.MovieSceneSkeletalAnimationTrack)

# Add section to track to be able to manipulate range, parameters, or properties

transform_section = transform_track.add_section()

anim_section = anim_track.add_section()

# Get level sequence start and end frame

start_frame = level_sequence.get_playback_start()

end_frame = level_sequence.get_playback_end()

# Set section range to level sequence start and end frame

transform_section.set_range(start_frame, end_frame)

anim_section.set_range(start_frame, end_frame)

# Refresh to visually see the new tracks and sections added

unreal.LevelSequenceEditorBlueprintLibrary.refresh_current_level_sequence()

```

Copy full snippet

Some sections may require properties to be defined in order to be used. In the case of an Animation Track section, an animation asset must be defined. To do this, use the following commands:

```
# Get the animation sequence asset

anim_seq = unreal.load_asset("/Game/Mannequin/Animations/ThirdPersonWalk")

# Get the section, get the parameters, set animation to anim sequence asset

anim_section.params.animation = anim_seq

```

Copy full snippet

Copy and Paste Commands

Additionally, Objects, Tracks, sections and folders can all be organized and managed using python scripting, using the following copy and paste functions:

Folders

```
ls_system = unreal.get_editor_subsystem(unreal.LevelSequenceEditorSubsystem)

selected_folders =
unreal.LevelSequenceEditorBlueprintLibrary.get_selected_folders()

level_sequence =
unreal.LevelSequenceEditorBlueprintLibrary.get_current_level_sequence()

# Gets added to clipboard, returns text that can be inputted to paste

ls_system.copy_folders(selected_folders)

# Create parameters to determine where this will paste to

# In this case, this will paste to the same level sequence where it was copied from

paste_params = unreal.MovieScenePasteFoldersParams()

paste_params.sequence = level_sequence

paste_params.parent_folder = None

# Will look at clipboard if string is empty, however you can input the return text from copy_bindings

ls_system.paste_folders("", paste_params)

```

Copy full snippet

Bindings

```
ls_system = unreal.get_editor_subsystem(unreal.LevelSequenceEditorSubsystem)

selected_bindings = unreal.LevelSequenceEditorBlueprintLibrary.get_selected_bindings()

# Gets added to clipboard, returns text that can be inputted to paste

ls_system.copy_bindings(selected_bindings)

# Create parameters to determine where this will paste to

# In this case, this will paste to the same level sequence where it was copied from

# Since the properties require arrays of specific type, we will have to pass an empty array

paste_params = unreal.MovieScenePasteBindingsParams()

paste_params.bindings = []

paste_params.folders = []

paste_params.parent_folder = []

# Will look at clipboard if string is empty, however you can input the return text from copy_bindings

ls_system.paste_bindings("", paste_params)

```

Copy full snippet

Tracks

```
ls_system = unreal.get_editor_subsystem(unreal.LevelSequenceEditorSubsystem)

level_sequence = unreal.LevelSequenceEditorBlueprintLibrary.get_current_level_sequence()

tracks_to_copy_from = unreal.LevelSequenceEditorBlueprintLibrary.get_selected_tracks()

# The first selected binding will be the one from the selected track

bindings_to_paste_to = unreal.LevelSequenceEditorBlueprintLibrary.get_selected_bindings()[1:]

# Gets added to clipboard, returns text that can be inputed to paste

ls_system.copy_tracks(tracks_to_copy_from)

# Create parameters to determine where this will paste to

# In this case, this will paste to selected tracks to the selected bindings

# Since the properties require arrays of specific type, we will have to pass an empty array

paste_params = unreal.MovieScenePasteTracksParams()

```

```

paste_params.bindings = bindings_to_paste_to

paste_params.folders = []

paste_params.parent_folder = None

paste_params.sequence = level_sequence

# Will look at clipboard if string is empty, however you can input the
return text from copy_bindings

ls_system.paste_tracks("", paste_params)

```

Copy full snippet

Sections

```
ls_system = unreal.get_editor_subsystem(unreal.LevelSequenceEditorSubsystem)

level_sequence =
unreal.LevelSequenceEditorBlueprintLibrary.get_current_level_sequence()

# Get the sections from the first selected track

sections_to_copy_from =
unreal.LevelSequenceEditorBlueprintLibrary.get_selected_tracks()
[0].get_sections()

# Get any tracks after the first selected track.

tracks_to_paste_to =
unreal.LevelSequenceEditorBlueprintLibrary.get_selected_tracks()[1:]

# Gets added to clipboard, returns text that can be inputed to paste

ls_system.copy_sections(sections_to_copy_from)

# Create parameters to determine where this will paste to

# In this case, this will paste to selected section in same level sequence where it was copied from at the sequence's start time

# Since the properties require arrays of specific type, we will have to pass an empty array

paste_params = unreal.MovieScenePasteSectionsParams()

paste_params.time = unreal.FrameTime()

paste_params.track_row_indices = []

paste_params.tracks = tracks_to_paste_to

# Will look at clipboard if string is empty, however you can input the return text from copy_bindings

ls_system.paste_sections("", paste_params)


```

Copy full snippet

Track Filtering

Track filtering commands can also be used:

```
# Get track filter names and print them

track_filter_names = unreal.LevelSequenceEditorBlueprintLibrary.get_track_filter_names()

for track_filter_name in track_filter_names:

print(track_filter_name)

# Set the track filter for Skeletal Mesh and Selected Control Rig Controls

```

```
unreal.LevelSequenceEditorBlueprintLibrary.set_track_filter_enabled("Skeletal
Mesh", True)

unreal.LevelSequenceEditorBlueprintLibrary.set_track_filter_enabled("Selectec
Control Rig Controls", True)

# See the filter enabled status per track

print(unreal.LevelSequenceEditorBlueprintLibrary.is_track_filter_enabled("Eve

print(unreal.LevelSequenceEditorBlueprintLibrary.is_track_filter_enabled("Ske
Mesh"))

```

Copy full snippet

Additional Sequencer Scripting Resources

For more resources on general Sequencer Python scripting, refer to the Sequencer scripting examples located in your local engine path:

`…\Engine\Plugins\MovieScene\SequencerScripting\Content\Python`