

- Developer
- / Documentation
- / Unreal Engine ▾
- / Unreal Engine 5.4 Documentation
- / Animating Characters and Objects
- / Cinematics and Sequencer
- / Movie Render Queue
- / Using Command Line Rendering with MRQ

Using Command Line Rendering with MRQ

An overview of using Command Line Rendering with Movie Render Queue.



If your project contains existing C++ code that references `MoviePipelineMasterConfig.h`, you must replace these references with `MovePipelinePrimaryConfig.h`.

When rendering [Sequencer](#) projects using [Movie Render Queue \(MRQ\)](#) in **Unreal Engine**, you can use a combination of MRQ asset configuration, streamlined [Command Line Rendering](#) arguments, and Python Executors to customize your rendering workflow.

Unlike traditional offline render farms, which often process a single frame on a machine, the smallest unit of distributable work in Unreal Engine is a single camera cut, or a Level Sequence Shot. This is because Unreal Engine's real-time features rely on information from the previous frame and this data cannot be shared between multiple machines.

Movie Render Queue configuration files can be complex compared to the [legacy Render Movie feature](#). MRQ moves away from the Command Line arguments used by the legacy system and uses a new system for invoking renders from the command line. The list of supported command line arguments has been reduced to only a few basic arguments, but

Unreal Engine supports customizing your rendering process using Python. This means that you have a great deal more control over how command line renders are executed which provides increased flexibility compared to the legacy Render Movie system.

At a high level, the three modes you can run are as follows:

1. Specifying a single Level Sequence and config asset.
2. Specifying an entire Queue to be rendered.
3. Using the custom Python Executor to have full control over rendering.

The following document contains a description of the three argument modes supported, as well as a custom python executor that you can use and customize in your projects rendering using MRQ.

Level Sequence Arguments

The `-LevelSequence` argument can be used to specify the path to a specific Level Sequence asset to render using MRQ. If you use this route then you must also specify a `-MoviePipelineConfig` argument which points to a **UMoviePipelinePrimaryConfig** preset asset so the proper settings are assigned to the asset with which to render.

This method is the most basic use of the command line to initiate an MRQ render. Your command line might look something like the following, where `subwaySequencer_P` is the map to load, `SubwaySequencerMASTER` is the Level Sequence asset, and the `SmallTestPreset` is the **UMoviePipelinePrimaryConfig** preset asset:

```
1
2 UnrealEditor-Cmd.exe "E:\SubwaySequencer\SubwaySequencer.uproject" subwaySeque
3 -LevelSequence="/Game/Sequencer/SubwaySequencerMASTER.SubwaySequencerMASTER"
4 -
   MoviePipelineConfig="/Game/Cinematics/MoviePipeline/Presets/SmallTestPreset.Sm
   -windowed -resx=1280 -resy=720 -log -notexturestreaming
5
```


 Copy full snippet

This example method only supports rendering a single job on a single map, and will render all shots contained by that Level Sequence.

Movie Pipeline Config Argument

Building off of the Level Sequence argument, is to specify a `-MoviePipelineConfig` argument that operates as the path to a **UMoviePipelineQueue** asset (saved via the MRQ UI) instead of a **Primary Config** and Level Sequence asset. This will automatically process each job in the queue on the map specified by their entry in the queue, and with the configuration for each job stored in the queue. For this argument, your command line might look something like the following:

```
1
2 UnrealEditor-Cmd.exe "E:\SubwaySequencer\SubwaySequencer.uproject" subwaySeque
   game
3 -
   MoviePipelineConfig="/Game/Cinematics/MoviePipeline/Presets/BigTestQueue.BigTe
   -windowed -resx=1280 -resy=720 -log -notexturestreaming
4
```

 Copy full snippet

Custom Pipeline Functionality using Python

You can also build your own pipeline functionality using a custom Executor in Python, which can be used to read arguments from the command line, configure or create MRQ jobs, and more. To do this, you must use the `-`

`MoviePipelineLocalExecutorClass=/Script/MovieRenderPipelineCore.MoviePipelinePythonHostExecutor` and then specify your Python executor class with `-ExecutorPythonClass=...`.

When running a custom python executor, your command line might look something like the following:

```
1
2 UnrealEditor-Cmd.exe "E:\SubwaySequencer\SubwaySequencer.uproject" subwaySeque
3 -
   MoviePipelineLocalExecutorClass=/Script/MovieRenderPipelineCore.MoviePipelineP
```

```
4 -ExecutorPythonClass=/Engine/PythonTypes.MoviePipelineExampleRuntimeExecutor -  
   resx=1280 -resy=720 -log -notexturestreaming  
5
```

 Copy full snippet

With this command line the engine will boot up and then create an instance of your Python-based executor (`MoviePipelineExampleRuntimeExecutor` in this case). Once your executor is created you can read command line arguments, run MRQ jobs, communicate with external servers to determine job information, and more.

If you wish to build your own Python-based executor you will need to create a Python-based UClass which inherits from `unreal.MoviePipelinePythonHostExecutor`, and then you will need to override the `execute_delayed` function. You will need to create a "init_unreal.py" class in your `/Content/Python` folder which imports your custom module. This `init_unreal.py` file is required for Unreal to parse your python file and to re-write it into an engine supported UClass which can then be specified on the command line. Please see the example script for an example on how to do this.

You can view a sample Python executor in

`\Plugins\MovieScene\MovieRenderPipeline\Content\Python\MoviePipelineExampleRuntimeExecutor.py` which shows how to read from the command line, make **HTTP** requests and attempt local socket connections. The exact arguments for this sample are slightly more than what is pictured above so please read the information at the top of the example file when attempting to test it.