RenderStream

1 April 2026

DISGUISE

Contents

RenderStream Overview

RenderStream is Disguise's scalable architecture to extend and augment the platform's visual capabilities, plugging in both partner applications such as 3D realtime render engines and other visual functionality extensions.

A range of RenderStream plugins have been developed enabling the sequencing of content from a variety of sources including AI, AR, and graphics processing.

RenderStream is available for use in remote rendering and local rendering mode, and enabled with the same RenderStream license.

Remote Rendering

With RenderStream remote mode, the rendering is performed on one or more dedicated render nodes which are distinct from the media servers that are compositing and outputting the content. These are typically Disguise RX nodes. Render nodes can be used to scale content endlessly without being limited by the capabilities of a single GPU, and are connected together via a high-speed network configuration.

> **Note**
>
> Remote mode is also referred to simply as RenderStream.

Local Rendering

With RenderStream Local mode, the rendering is performed directly on the media server that is responsible for compositing and outputting the content, which minimises interaction delay by removing the need to transport frames through a high-speed network.

RenderStream Remote vs RenderStream Local

Criteria	RenderStream remote mode	RenderStream local mode
Flexibility	Provides maximum flexibility for rendering without being tied to the media server's physical output.	The workload must render what is required by the physical output of the media server.
Performance	Non-rendering operations cannot impact rendering performance.	Since rendering, compositing, and other operations are executed on the same media server, it may impact the performance.
Simplicity	Requires more complex set up and additional network configuration.	Provides simplified set up in Designer and does not require additional network administartion above a regular Designer session.
Network Latency	Includes additional network latency for a stream.	Does not include additional network latency since rendering and physical output are performed on the same server.
Rack Space	Requires additional render nodes and Fabric switch.	Requires less rack space.

For more information about these modes, check out RenderStream) and RenderStream Local.

RenderStream Assets

RenderStream assets are the files that are used to create the content that is rendered by the render nodes. These assets can be created in Unreal Engine, Unity, Notch, TouchDesigner, and other third-party render engines integrated via the RenderStream SDK.

Assets are stored in a specific location on the render node. By default this is the `D:\Renderstream Projects` folder on Disguise hardware products, and `C:\Users\USERNAME\Documents\Renderstream Projects` otherwise. This path is defined by a registry key, `Computer\HKEY_CURRENT_USER\Software\d3 Technologies\d3 Production Suite\RenderStream Projects Folder`. The path can be changed by modifying the registry key. The change will be automatically detected and the new path will be scanned for assets.

Assets are discovered automatically by d3service, which watches the RenderStream projects folder path for changes, and are shared via the network. Any Designer instance, connected to the same network will be able to select the asset.

Engine Support

Disguise provides first-party support for Unreal Engine, Unity and Notch with RenderStream. In order to communicate with Disguise, both Unreal Engine and Unity require the installation of a plugin on the render node. Visit the **Disguise GitHub** for the latest plugins.

Other engines, such as TouchDesigner and **Volinga**, are maintained by their developers. RenderStream is open source for developers to implement their own support. Take a look at Extending RenderStream to learn about making your own implementation of RenderStream for custom engines.

Further Reading

Read the **Disguise Whitepaper** to learn more about how the Disguise architecture works with RenderStream.

RenderStream Resources

RenderStream versions, downloads, and release notes.

RenderStream is a protocol for controlling third-party render engines from Designer.

We recommend that in all instances you use the latest plugin versions, which can be found on GitHub.

RenderStream plugins support both layers, the RenderStream layer, and the RenderStream Local layer.

Unreal Engine

GitHub https://github.com/disguise-one/RenderStream-UE/releases

For release notes and full information, see GitHub.

Versions

Unreal Engine 5.7

Plugin	Compatibility
RS UE 5.7 plugin r2.0	Compatible with r25 & higher.

Unreal Engine 5.6

Plugin	Compatibility
RS UE 5.6 plugin r2.0	Compatible with r25 & higher.

Unreal Engine 5.5

Plugin	Compatibility
RS UE 5.5 plugin r2.0	Compatible with r25 & higher.

> **Note**
>
> The settings UI for this version is slightly different. Instead of the Scene Selector setting being in a separate section, it is now in the same section as the rest of the settings.

> **⚠ Warning**
>
> Due to a bug in Unreal Engine 5.5.0, DX11 does not work with RenderStream. This issue was fixed in version 5.5.1.

Unreal Engine 5.4

Plugin	Compatibility
RS UE 5.4 plugin r2.0	Compatible with r25 & higher.

Unreal Engine 5.3

Plugin	Compatibility
RS UE 5.3 plugin r2.0	Compatible with r25 & higher.

Unreal Engine 5.2

Plugin	Compatibility
RS UE 5.2 plugin r2.0	Compatible with r25 & higher.
RS UE 5.2 plugin r1.30	Compatible with r22 & higher.

Unreal Engine 5.1

Plugin	Compatibility
RS UE 5.1 plugin r1.30	Compatible with r22 & higher.

Unreal Engine 5

Plugin	Compatibility
RS UE 5.0 plugin r1.30	Compatible with r22 & higher.
RS UE 5.0 plugin r1.29	Compatible with r21.

Unreal Engine 4.27

Plugin	Compatibility
RS UE 4.27 plugin r1.30	Compatible with r22 & higher.
RS UE 4.27 plugin r1.29	Compatible with r21.
RS UE 4.27 plugin r1.28	Compatible with r20.

Plugin	Compatibility
RS UE 4.27 plugin r1.27	Compatible with r19.

Unreal Engine 4.26

Plugin	Compatibility
RS UE 4.26 plugin r1.28	Compatible with r20.
RS UE 4.26 plugin r1.27	Compatible with r19.
RS UE 4.26 plugin r1.25	Compatible with r18.

Unreal Engine RenderStream Sample projects

Sample projects include:

- RenderStream plugin
- backplate camera (cam_backPlate)
- front plate camera (cam_frontPlate)
- an auto-generated .JSON file

Each camera has a **RenderStream Channel Definition component** attached, ready to connect to a **RenderStream layer**.

Each project has a rotating Disguise logo blueprint added to the scene. A float variable parameter called “logoRotationSpeed” has been exposed to control the rotation speed of the logo via the level blueprint. This demonstrates how an exposed variable is setup in the level blueprint. This exposed parameter is set to editable and so can be controlled directly from the **RenderStream layer** editor.

Sample projects

Designer r25 and above [RS2.0]

UE Version	RS Version	Link
RS UE 5.7	RS2.0 v0 plugin	max_UE5_7_clean_26_01_27.zip
RS UE 5.6	RS2.0 v0 plugin	max_UE5_6_clean_24_12_20.zip
RS UE 5.5	RS2.0 v0 plugin	max_UE5_5_clean_24_12_20.zip
RS UE 5.4	RS2.0 v1.1 plugin	max_UE5_4_clean_24_08_28.zip
RS UE 5.3	RS2.0 v2 plugin	max_UE5_3_RS2_0-24_04_09.zip
RS UE 5.2	RS2.0 v0 plugin	max_UE5_2_rs2-0_24-01-24.zip

UE Version	RS Version	Link
RS UE 5.1.1	RS2.0 v0 plugin	max_UE5_1_Clean_240401.zip
RS UE 5.0.3	RS2.0 v0 plugin	max_UE5_0_rs2-0_24-01-24.zip
RS UE 4.27	RS2.0 v0 plugin	max_UE4_27_Clean_240401.zip

Designer r22 and above [RS1.30]

UE Version	RS Version	Link
RS UE 5.2	RS1.30 v0 plugin	max_UE5_2_rs1-30_24-01-24.zip
RS UE 5.1.1	RS1.30 v0 plugin	max_UE5_1_rs1-30_24-01-24.zip
RS UE 5.0.3	RS1.30 v0 plugin	max_UE5_0_rs1-30_24-01-24.zip
RS UE 4.27	RS1.30 v0 plugin	max_UE4_27_rs1-30_24-01-24.zip

Designer r21 and below

UE Version	RS Version	Link
RS UE 5.0.3	RS1.29 v9 plugin	max_UE5_0_rs1-29_r21_24-01-24.zip
RS UE 4.27	RS1.29 v6 plugin	max_UE4_27_rs1-29_21_24-01-24.zip

Unity

GitHub https://github.com/Disguise-one/RenderStream-Unity

For release notes and full information, see GitHub.

For more information, see RenderStream and Unity.

Notch

The Notch plugin is bundled with Designer.

For the Demo Notch Builder Project, visit download.Disguise.one/#resources.

For more information on the Notch integration, visit RenderStream and Notch.

TouchDesigner

For more information on the RenderStream and TouchDesigner integration, visit RenderStream and TouchDesigner.

Volinga

For more information on the RenderStream and Volinga integration with Unreal Engine, visit Volinga.

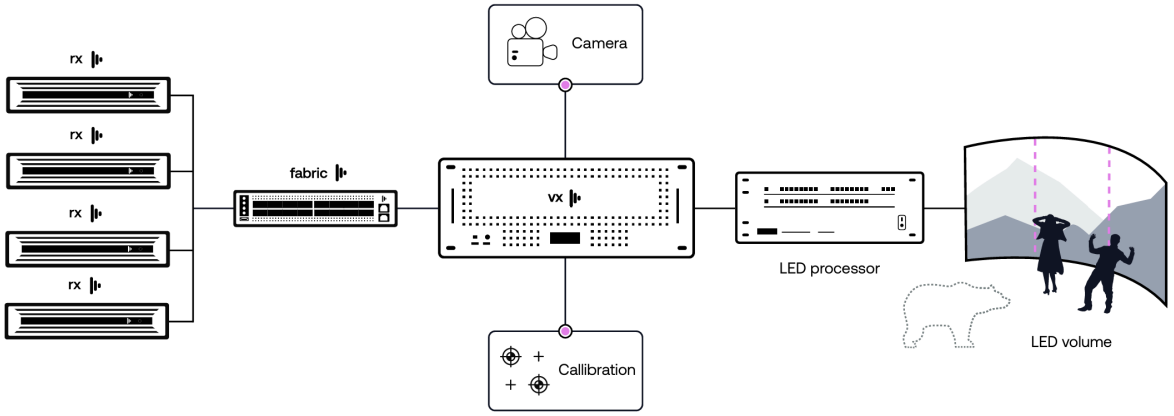
RenderStream Integration SDK

The RenderStream Integration SDK is available to those wishing to develop their own integrations.

The GitHub repository is for API definitions, documentation and examples. For more information, see the GitHub RenderStream Integration SDK page.

RenderStream

RenderStream is compatible with Unreal Engine, Unity, TouchDesigner and Notch with the rendering performed on render nodes. Sequencing of the third-party render engine can be accomplished via the **RenderStream Layer** in Designer.



Uncompressed vs. Compressed

RenderStream Uncompressed requires the use of a Mellanox network interface to stream uncompressed, 10bit video data. To test RenderStream Uncompressed appropriately, access to multiple machines and the networking equipment specified is required. Testing RenderStream Uncompressed also requires both a RenderStream Send License and a RenderStream Uncompressed License.

RenderStream Compressed provides consistent content quality with advanced H.265 compression. You can now choose high frequency compression, and ensure that even the finest details of your content are shown via our superior, reliable networking.

Cluster Rendering

> **Warning**
>
> For cluster rendering it's recommended to use render nodes from the same Disguise product range, e.g. all RX series machines. Mixing of machines from different product ranges is not recommended and is unsupported. It is acceptable to mix RX and RX II types however.

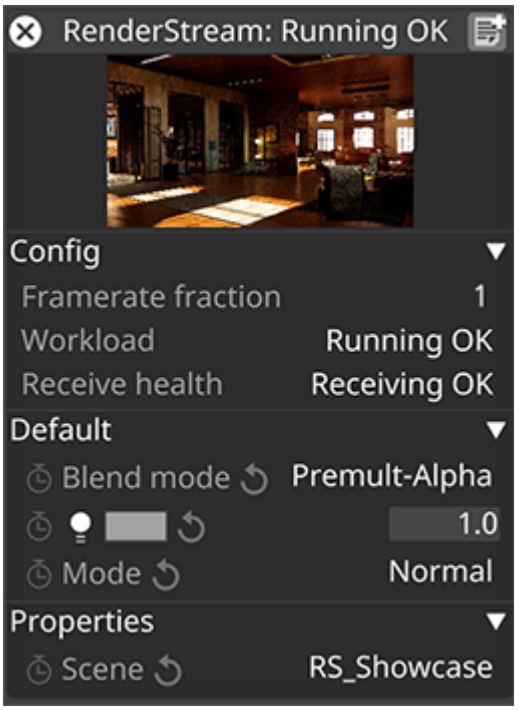
Here are just some of the benefits of using :

1. Cluster Rendering allows you to span your render engine content over more than one Disguise server by scaling out real-time content up to an unlimited capacity.
2. Each machine will let you render a fragment of your final content frame to increase the render power and get your content onto your displays at your desired quality.
3. Use Cluster Rendering to render real-time content of the highest quality, detail and framerate without worrying about GPU power.

Cluster Rendering is configured within Disguise Designer using the **RenderStream Layer**.

RenderStream Layer

The RenderStream Layer is used to control third-party render engines running externally to Designer.

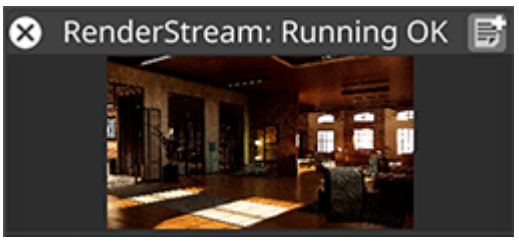


There are four main sections within the RenderStream Layer:

- Media Thumbnail
- Config
- Default
- Properties

Media Thumbnail

This area will display an image of the active stream being received. Note that in a multi-channel environment, this will only show the Camera in UE that has the RS component attached to it.



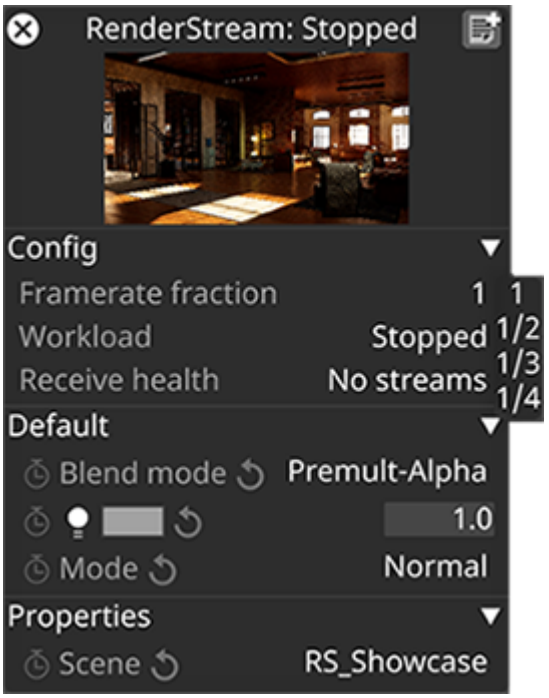
Config

This section is where you will configure properties such as **Asset** and **Cluster Pool**, as well as monitor workload and engine health in real time.

Framerate Fraction

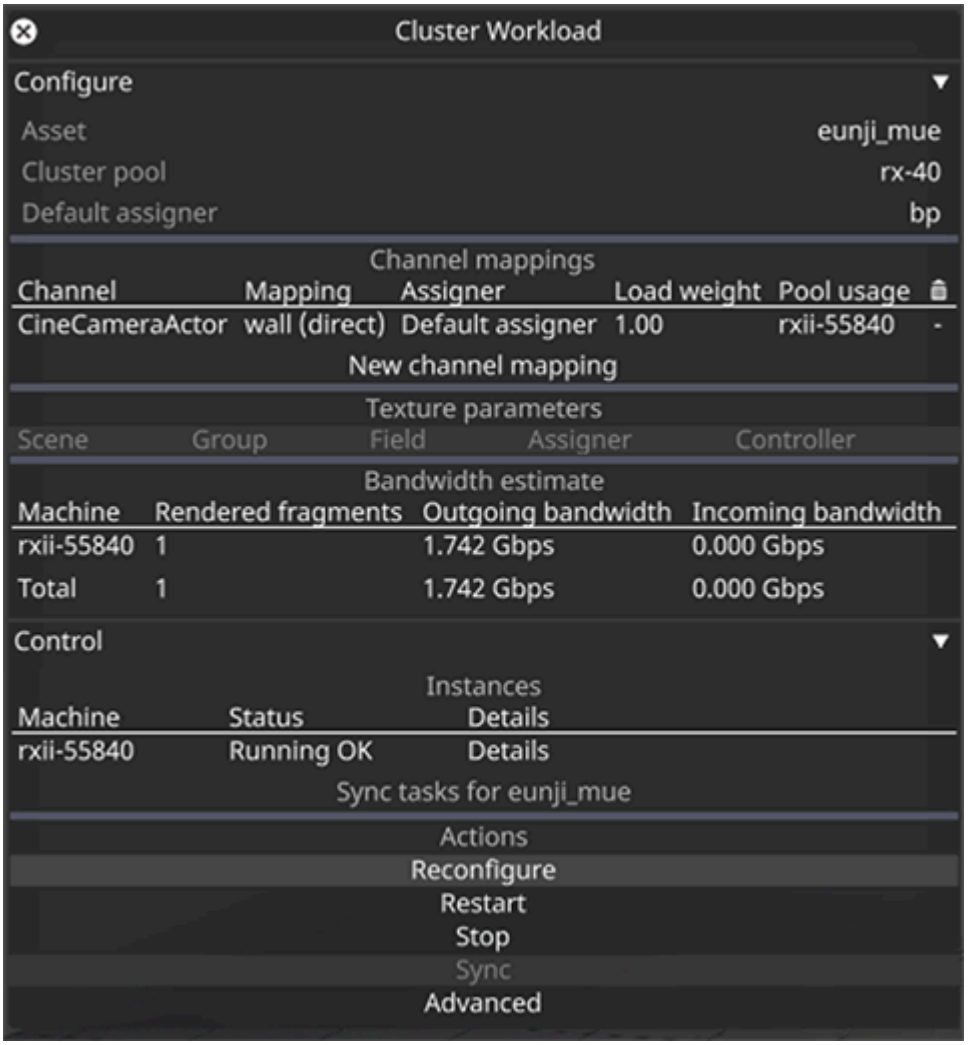
The framerate of each stream is determined by the global refresh rate set within d3. The Framerate Fraction setting indicates the fraction of the framerate this layer renders at.

The options are 1, 1/2, 1/3 and 1/4.



Cluster Workload

The Cluster Workload window contains the elements required for configuring a Cluster.



It contains the following sections:

`Configure`

The Configure section contains these elements of a Cluster:

```
- Asset
- Cluster Pool
- Cluster Assigner
- Channel Mapping
- Bandwidth Estimate
```

`Control`

Asset

This is the Render Engine project or Notch block that will be controlled by the layer. In order for the asset to be visible here, it must be present in the RenderStream projects folder.

Right-click on the asset name to open the **Asset** properties editor:

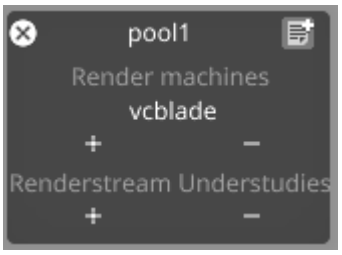
`Asset`

Properties include:

- **Asset Discovery Name**: Name of UE folder/project.
- **Target Engine**: Name of the Render Engine.
- **Engine Settings**: Engine version.
- **Colour shift**: Apply colour correction to entire asset.
- **Content Source Machine**: The render node that is running the render engine project. This is the machine that the project will be synced from.
- **Available**: Status indicator displaying if the Asset is available. Not a tick box.
- **Channels**: displays active channels defined by the CameraActors in your UE project.

Cluster Pool

The **Cluster Pool** will automatically detect all available machines on the same network. Once detected, you can assign as many machines to the **Cluster Pool** as needed and inspect each machine in the pool to see network status, IP address, current Streams and available Assets.



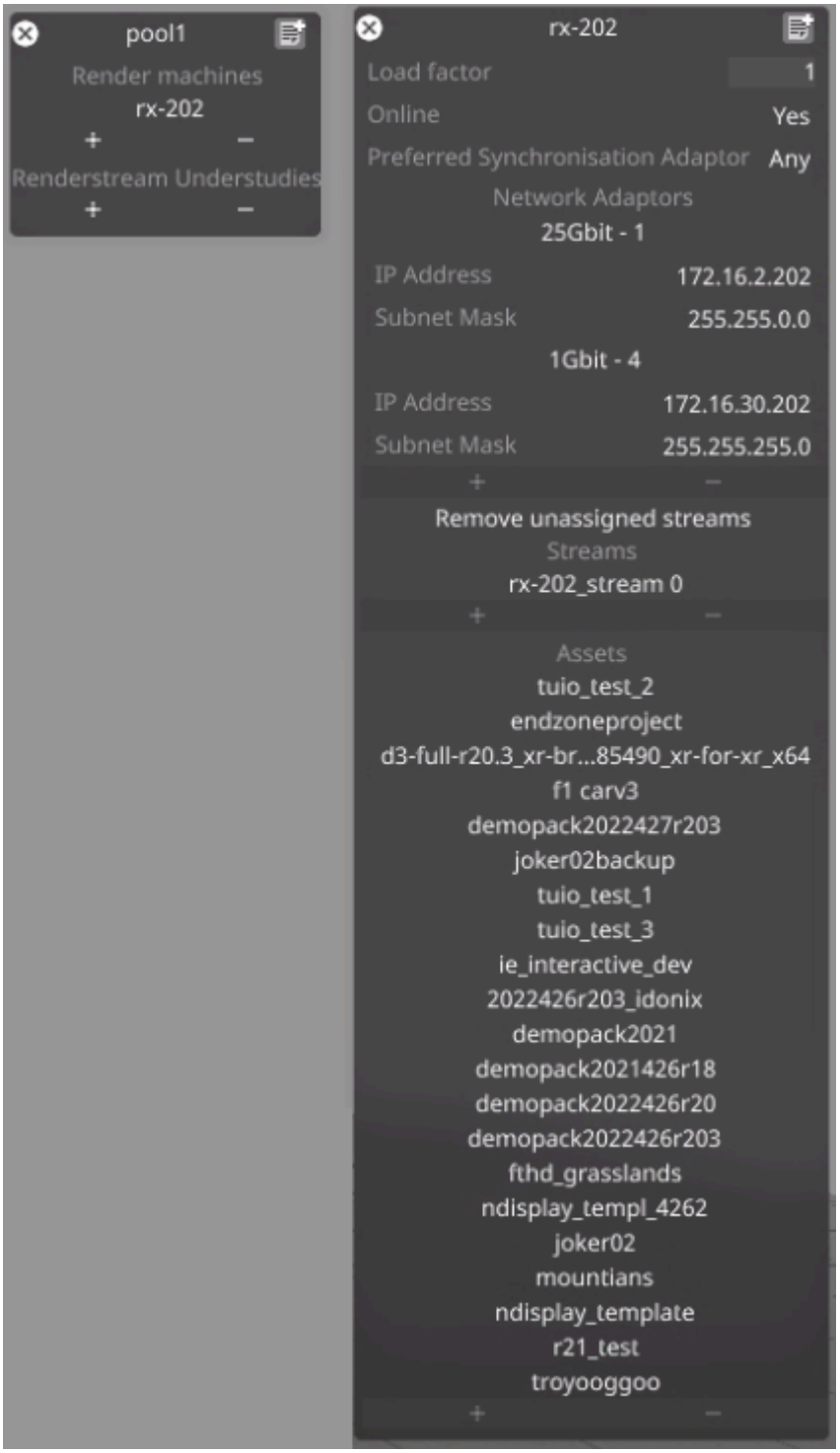
`Cluster Pool`

Properties include:

- **Render Machines**: The list of render nodes that the content will be distributed across.
- **Understudies**: Backup nodes in case of failover.

Cluster Pool Machine Properties

Right-click on the name of a machine in the **Cluster Pool** to edit additional properties related to that machine:



`Cluster Pool Machine`

Properties include:

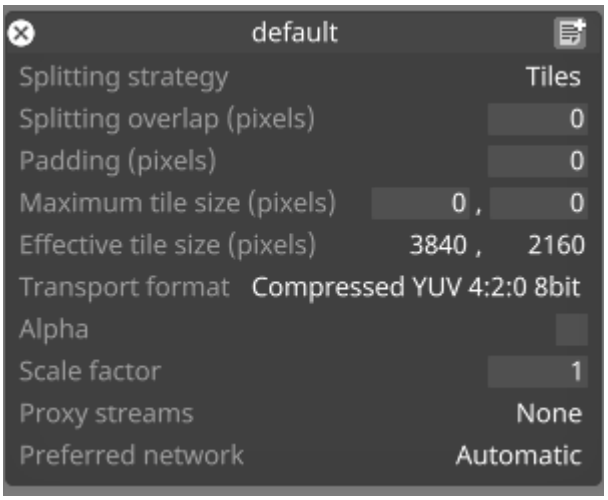
- **Load Factor**: Relative load weight this machine can handle. Weight is relative to other nodes in pool. The workload can be spread evenly across the cluster by the user using a load factor. This allows you to take into account each machine's spec and assign loads accordingly.
- **Online status**

- **Preferred Synchronisation Adapter**: This setting allows you to pick a subnet to use for sync. You should pick something fast - preferably capable of 100Gbps or more. Only used to control cluster communication and content syncing; it has no impact on how the streams are communicated to Disguise.
- **Network Adapters and settings**: Non-editable list of network adapters available on the node. To configure the system for Cluster Rendering, all machines within a Cluster Pool must be on the same subnet. If a machine is reporting an unexpected IP address (e.g. loopback address), make sure to select the specific network adapter within the 'Network' tab of d3Manager.
- **Streams**: non-editable list of RS output from the node.
- **Assets**: non-editable list of assets resident on the node.

Cluster Assigner

The **Cluster Assigner** is where we control the settings for the distribution of the content across all the machines in the **Cluster Pool**.

Cluster Assigners are used to create definitions as to how channels are delivered from the nodes. You can create as many Cluster Assigners as needed.



`Cluster Assigner`

Properties include:

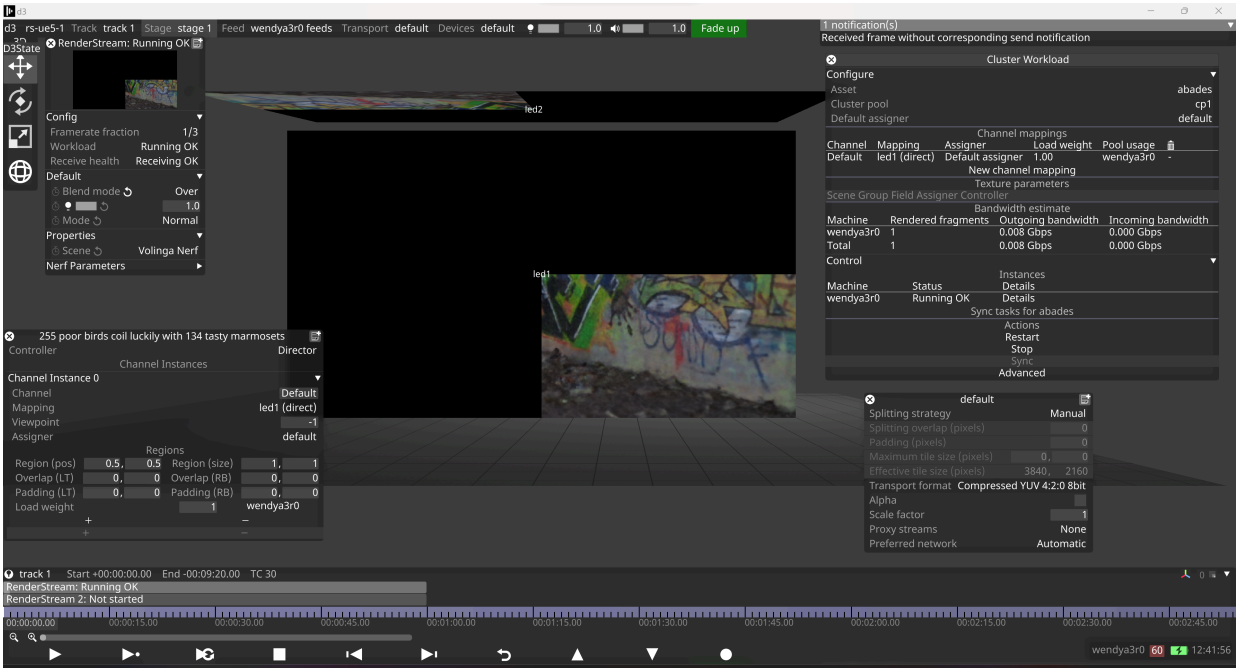
- **Splitting Strategies**

These are used to generate the sub-regions needed for distributing content across a set of render nodes.

There are 3 types of splitting strategies:

1. **Tiles** - Lets Designer split up the content as it sees fit to the mapped channels.
2. **Full frame** - Sends the entire stream to the mapped channel.

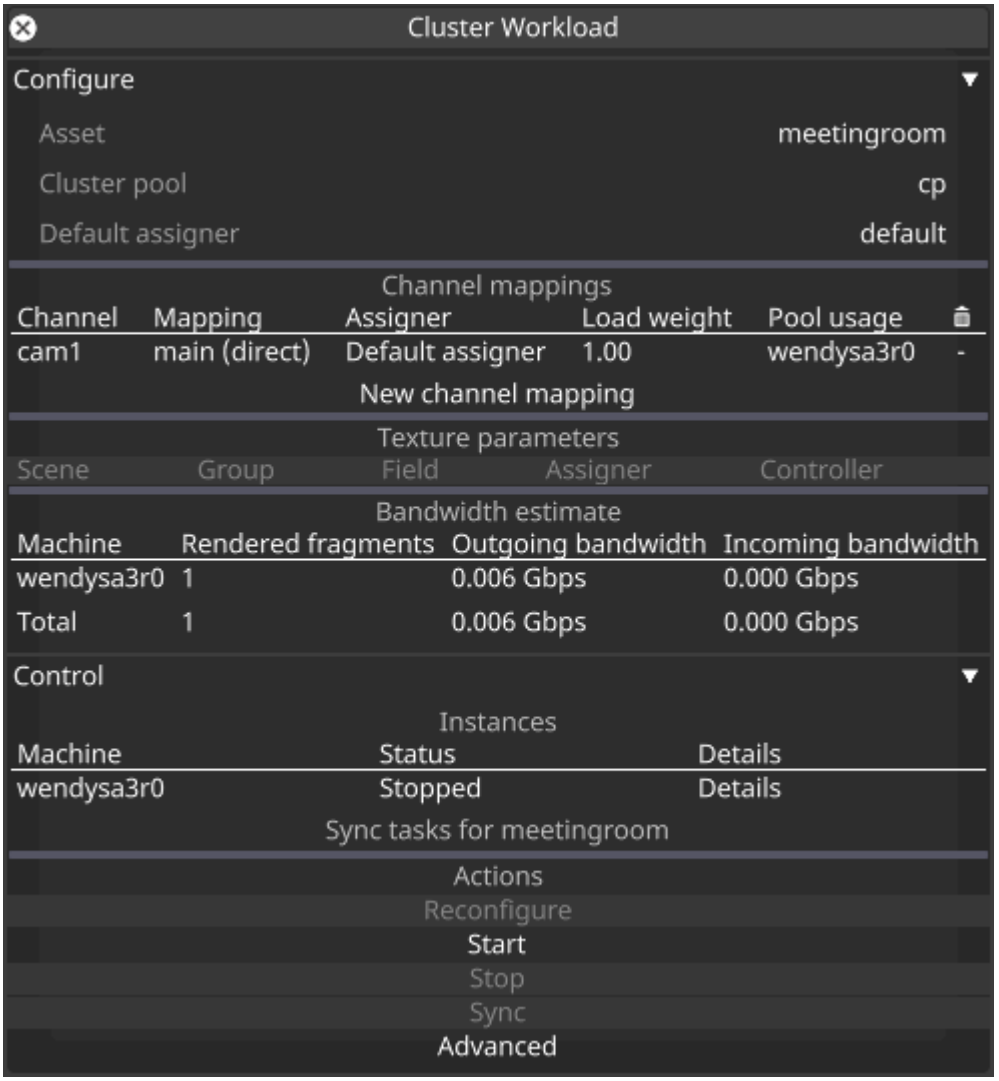
3. **Manual** - Splits each Viewport according to user specified instructions. When using the 'Manual' splitting strategy, the options for specifying how to split frames can be found within the Advanced workload settings.



In Advanced workflow settings, note that region pos. has a range between 0 and 1. The screenshot shows both regions (pos) set to 0.5. Other property values in this editor also range from 0 to 1.

- **Splitting Overlap**: The Splitting Overlap option can be used to define a blend region at each split that hides imperfections at the seams.
- **Padding (Pixels)**:The Padding option enlarges the rendered are at each split, but does not stream the padded area, cutting off edge artifacts.
- **Load Weight**:'Load Weight' allows you to set a priority for the Cluster Assigner. The higher the weight of the assigner, the more machines will be assigned to render the content and thus more splits will be made.
- **Transport format**: The options to pick between 'Compressed' and 'Uncompressed' , 'RGB 4:4:4', 'YUV 4:2:2', and 'YUV 4:2:0', and 8-bit, 10-bit and 12-bit transport bit-depths are available.
- **Alpha**: The option to enable or disable the Alpha channel on the content is available.
- **Scale factor**: Scale factor for the resolution.
- **Proxy Streams**: A compressed stream at a lower quality or resolution used for providing a preview on ma chines where receiving the full-quality stream is undesirable or impossible.
- **Preferred network**: Select the adapter to use for video transport on any channel the assigner is associated with.

Once the cluster has been configured, the stream's playback is managed via the Cluster Workload controls.



The options for managing the workload are:

- **Instances**: There can be only one workload instance running at a time in the same cluster pool.

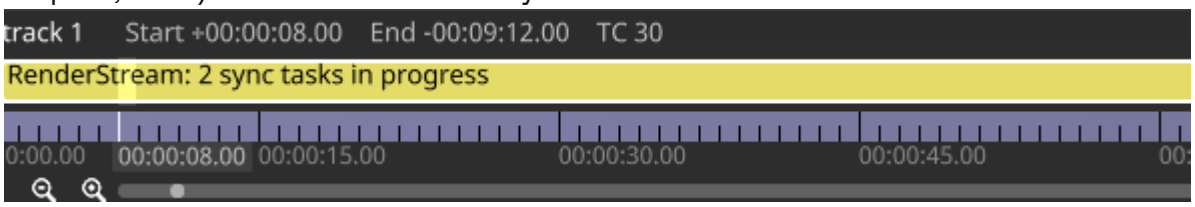
  When started, the Instances table will be populated with the machines that have been sent the start signal.

- **Reconfigure**: Applies updated workload settings directly to the running stream without requiring a full restart.
- **Start:** Sends signal to all machines in the pool to launch the Asset according to config settings.
- **Stop:** Sends signal to cease outputting a stream and quit the process sending it.

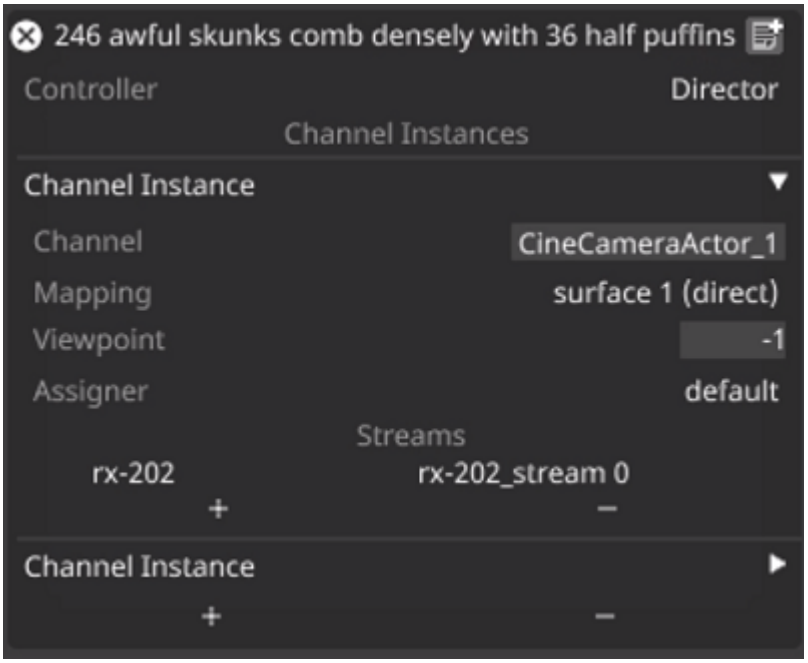
  The workload log can be opened by left-clicking on the status within the Instances table.

- **Sync:** begins the process of content sync. Content is synced from the source machine to all other machines available within the cluster pool; copies only the necessary directories and files needed.

  With r31.0.3, a new workload status has been added to show sync progress (in progress, complete, failed) on the RenderStream layer.



- **Advanced:** right-click to open the workload's advanced options:

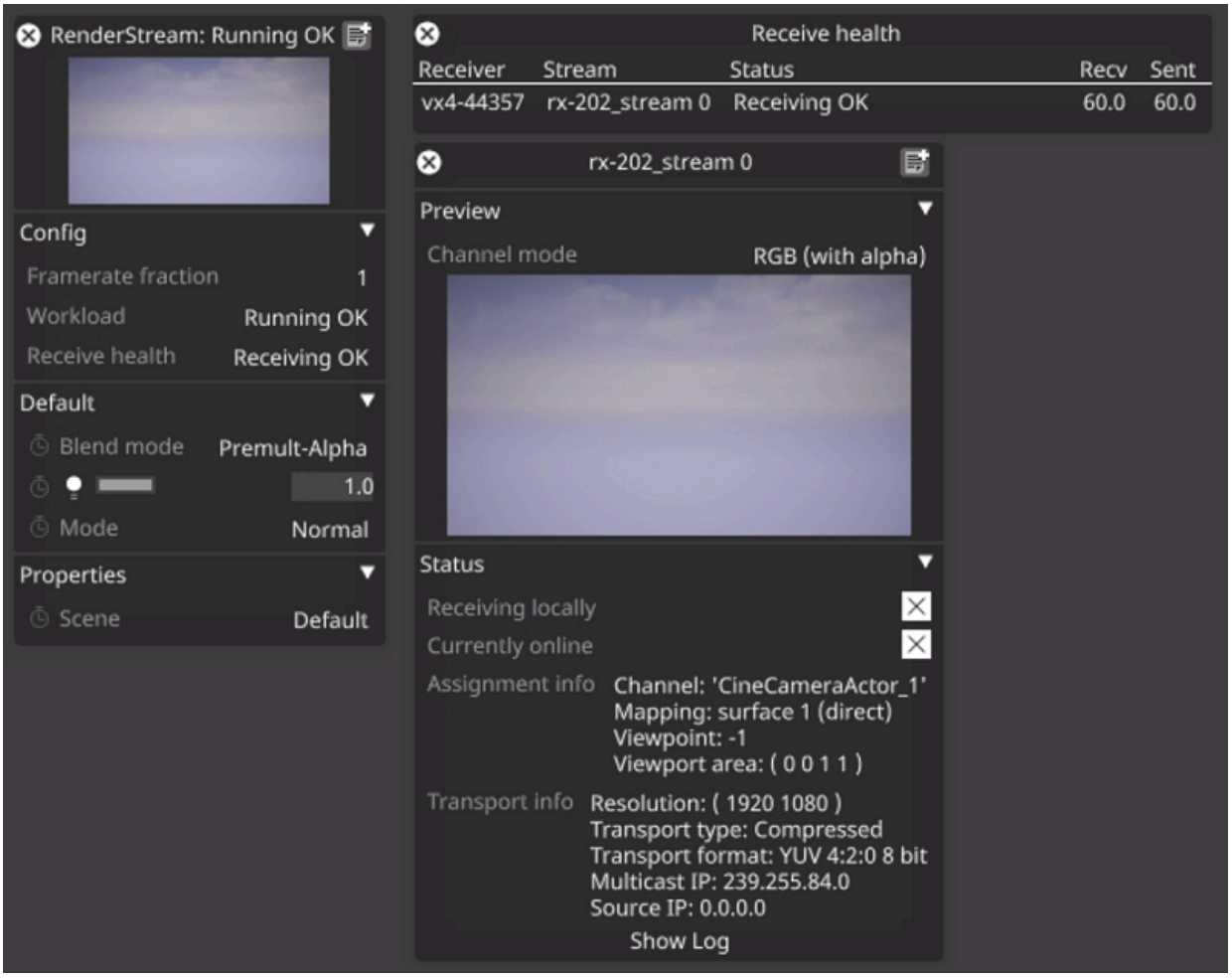


> **Note**
>
> **Please note**: Any changes to the Assigner require a restart of the workload.

Receive Health

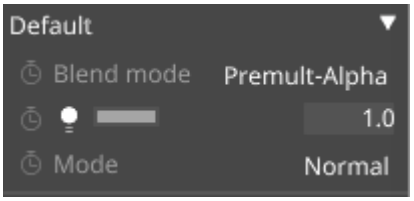
Status of entire cluster receiving streams, dynamically reported while stream is running.

Right-click on **Receive health** to open machine specific window:



Default

In the Default section of the RenderStream Layer, there are four Common Layer Properties:



- **Blend Mode**: Premult-Alpha is needed for content that contains alpha data.
- **Brightness**: Layer intensity.
- **Mode**: This is used to determine the timeline's playback behaviour as the layer is playing.

Properties Section

The last section of the RenderStream layer is asset specific and will display all exposed parameters from the asset as well as their keyframe editors

Health Monitoring

Receive Health

The Receive health field in the RenderStream Layer is used to monitor the health of all streams belonging to the workload in this layer. As the name suggests, receive health monitors whether machines that are subscribed to streams receive all the frames that are being sent from the render nodes as part of each stream.

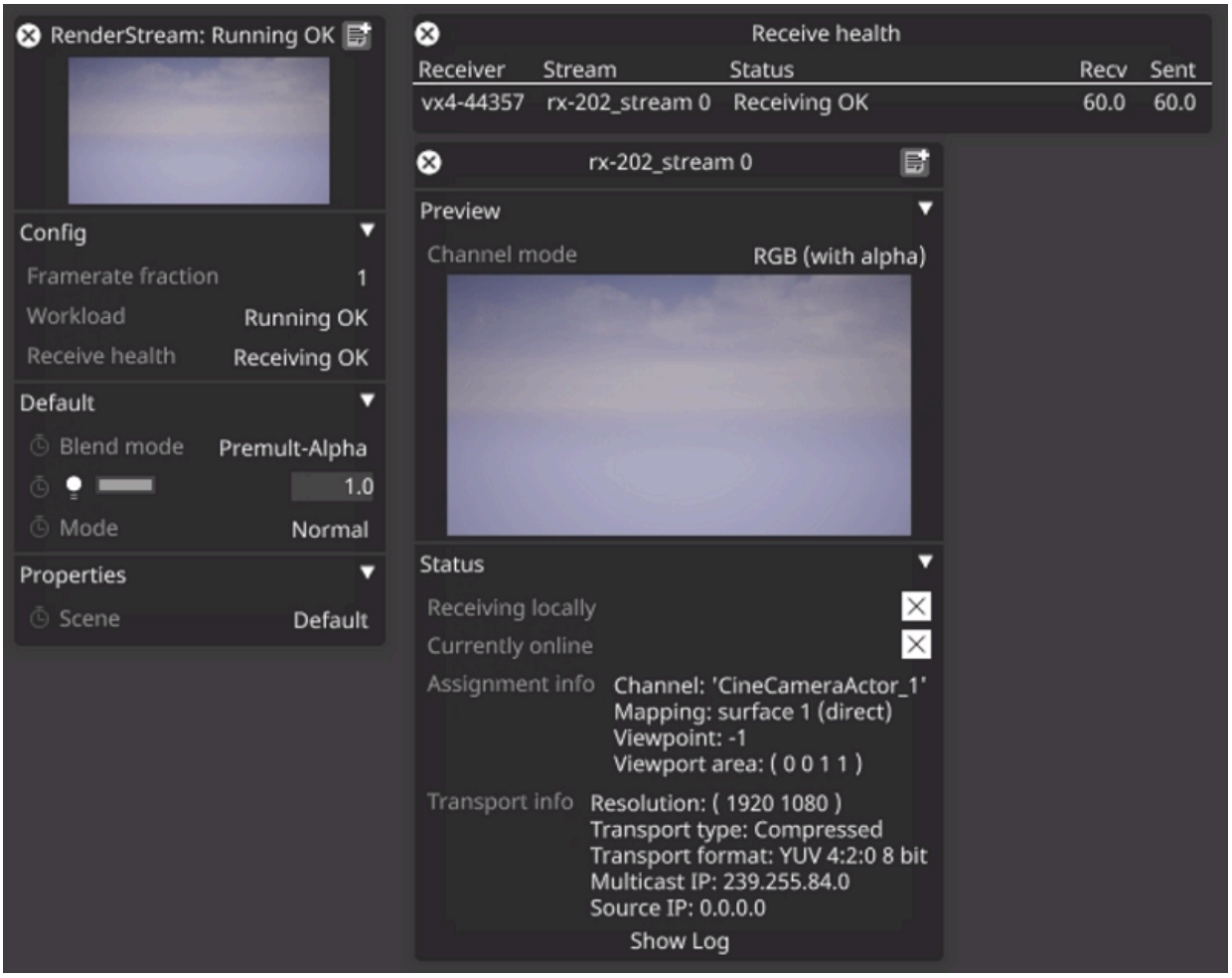
For concerns about the health of the workload render instances, check the **Workload Health** section.

The value displayed in this field is an aggregated status of all streams in the workload.

Possible values are:

- **No subscription requested**: A stream exist, but the receiver of the stream has not subscribed to it.
- **Receiving OK**: The stream is being received correctly.
- **No frames received in s**: No frames have been received in the specified amount of seconds.
- **No frames received in > 10s**: No frames have been received in over 10 seconds.
- **Bad stream ()**: There was an error with the stream, and it includes the specific error message.
- **Frames dropped recently**: A frame was dropped within the last second.
- **No frames received yet**: No frames have been received yet, and the receiver has not reported any dropped packets.
- **No frames received (dropping packets)**: No frames have been received yet because the incoming frames were dropped. This status signifies a network issue.
- **Dropping packets**: Incoming frames are reported as dropped by the receiver.
- **Subscribe unsuccessful**: A stream exists and the receiver tried to subscribe, but failed.

In the aggregated ‘Receive health’ field, the aggregated status reflects the highest priority status of all streams in the workload. The status list above has been sorted from lowest priority to highest priority status.



Receive Health Widget

Right-click on Receive health field to open the Receive Health widget.

The Receive Health widget provides a detailed view of the health of each stream in the workload. It includes information such as:

- **Receiver**: Name of the machine subscribed to the stream.
- **Stream**: Unique identifier of the stream.
- **Status**: Current status of the stream. See Receive Health section for possible values.
- **Recv**: Average receive rate of the stream in frames per second. This is based on how many frames the receiver machine receives.
- **Sent**: Average send rate of the stream in frames per second. This is based on how many frames the sender machine reports as being sent.

Workload Health

Workload health is used to monitor the health of individual render instances (an instance of a render engine running on a machine) in the workload. The status shown in the “Workload” field in the RenderStream Layer is an aggregated status of all render instances running in the

workload. For detailed information per render instance, right-click on the “Workload” field to open the Workload widget while the workload is running, and look at the “Instances” section.

Possible values are:

- **Unknown**: Error getting the status.
- **Crashed**: Remote engine is not responding.
- **Stopping**: The engine has acknowledged the request to shut down and is in the process of responding to that request.
- **StopRequested**: The engine has been asked to shut down.
- **Stopped**: Remote engine is not expected to be running.
- **Error**: An error occurred in the remote engine and is in a bad state. The engine is not stopped either since the process is alive.
- **Launching**: Workload has been requested to start on the remote machine and no response has been received yet.
- **Starting**: The remote engine is sending status updates but is currently not in a state where it will send frames.
- **Offline**: The RenderStream plugin in the remote engine is not sending responses, but the process hasn’t crashed.
- **Ignoring requests**: The remote engine is in a state where it can send frames but is not currently responding to any of the frame requests sent by the workload controller machine.
- **Awaiting requests**: The remote engine is in a state where it can send frames but it has not yet received any requests from the workload controller machine.
- **Warning**: Non-critical issue exist and the workload is still running.
- **Running OK**: The remote engine is processing all incoming requests and sending frames as expected.
- **Discarding requests**: The remote engine is processing some of the incoming requests, but is not fast enough to keep up with the current frame request rate.
- **Dropping input frames**: The remote engine did not receive some of the remote textures that it needed.

Workflow

Set up your environment including installation of the Render Engine and required plugin.

Use the RenderStream Layer to control the third party render engine.

- **Notch**
  - Plugin - included in Designer.
  - Documentation
- **Touch Designer**
  - Plugin
  - Documentation

- **Unity**
  - Plugin
  - Documentation
- **Unreal Engine**
  - Plugin
  - Documentation

> **Note**
>
> The preferred sync adapter should not be left set to 'any', it should be set to the network of the highest bandwidth that is available to all machines in the workload, plus the Disguise machines. This network will then be the one that is used to do the project sync and for all workload comms.

RenderStream Schema

When using the RenderStream plugin in a project, a JSON file is automatically generated.

The schema of the RenderStream JSON file is detailed below.

Top-Level Fields

Field	Description
`majorVersion`	The major version of the JSON schema.
`minorVersion`	The minor version of the JSON schema.
`engineName`	Name of the engine generating the file (e.g., “Unreal Engine”).
`engineVersion`	The version of the engine used (e.g., “5.4.0”).
`pluginVersion`	Version of the RenderStream plugin.
`info`	Reserved for additional metadata (currently empty in your example).
`channels`	A list of RenderStream channels (typically camera names or IDs).
`schema`	An array of scene-specific parameter definitions (explained below).

Schema → Parameters Structure

Each item in the schema array defines a scene and its controllable parameters. In your example:

Schema Object

```
{
    "name": "SampleScene",
    "parameters": [ ... ]
}
```

- name: The name of the RenderStream scene.
- parameters: An array of adjustable parameters in that scene.

Example Parameter Object


```
{
    "group": "Default",
    "displayName": "Message",
    "key": "Message",
    "type": 0,
    "min": 0.0,
    "max": 1.0,
    "step": 1.0,
    "defaultValue": {
        "index": 0,
        "data": 1.0
    },
    "options": ["Off", "On"],
    "dmxOffset": -1,
    "dmxType": 2,
    "noSequence": false,
    "readOnly": false
}
```


Parameter Field Breakdown

Field	Description
`group`	Logical grouping for UI display (e.g., “Default”).
`displayName`	Name shown in the RenderStream controller interface.
`key`	Internal key identifier for the parameter.
`type`	Data type (e.g., 0 often represents enum/switch-style).
`min`, `max`	Range of values allowed.
`step`	Incremental step between values.
`defaultValue`	Contains both `index` and actual `data` value.
`options`	For enumerated types, a list of string values (e.g., “Off”, “On”).
`dmxOffset`	DMX control offset (if used in lighting/control contexts).
`dmxType`	Type of DMX value.
`noSequence`	If true, disables sequence recording.

Field	Description
`readOnly`	If true, the parameter is read-only.

Channels

```
"channels": ["rscam1", "rscam2", "rscam3"]
```

RenderStream Failover

Active backups are crucial for live events. With the RenderStream Failover functionality, you can set up rules and switch inputs to allow your servers to switch to an Understudy if a stream isn't working.

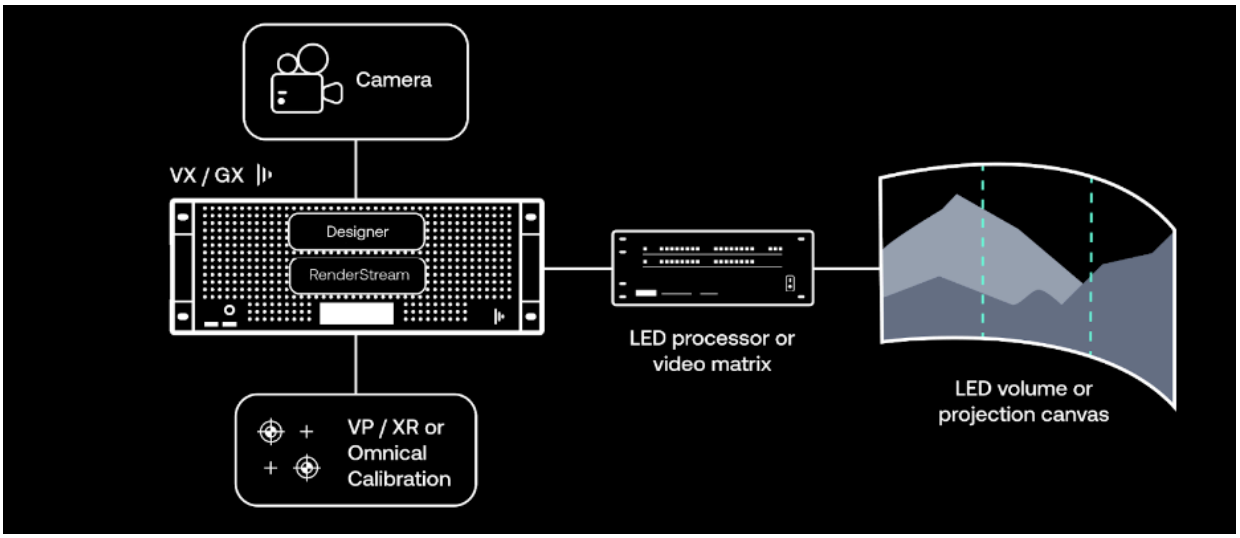
Follow these steps to configure RenderStream for failover:

1. Create a RenderStream Layer and follow the steps on the RenderStream Layer page to configure a Cluster Workload.
2. Start the workload; all render machines and Understudy will launch their instances of the project
3. Use the d3net Network Session widget to monitor and control the render machines and Understudy; switch to the Understudy within the Network Session widget if needed using 'Mark as Failed" on the stream that has failed and Disguise will automatically switch to the Understudy.

RenderStream Local

RenderStream Local has been designed to provide flexible real-time control without the hardware overhead and networking complexity. The setup has been simplified with minimal physical infrastructure requirements. By hosting rendering and output on the same machine, RenderStream Local achieves lower output latency.

RenderStream Local is compatible with Unreal Engine, Unity, TouchDesigner, and Notch. Sequencing of the third-party render engine can be accomplished via the RenderStreamLocal Layer in Designer.



Features

RenderStream Local enables RenderStream workloads to run locally on VX (with the exception of the VX 4 and VX 4+) or GX range media servers with any render engine which has a RenderStream plugin. Unreal Engine, Unity, TouchDesigner, and Notch are all supported.

The VX 4 and VX 4+ do not currently support RenderStream Local as further development work is needed for AMD GPU support.

Limitations

Running a render engine on the same media server as Designer is inherently less performant. This will depend on the type of render engine and the nature of the scene. It is recommended to always test the content in RenderStream Local configuration in order to determine if the performance is satisfactory.

Compute for rendering is shared with compute for compositing and other operations, careful testing of any set up is required.

Server Support

The VX 4 and VX 4+ do not currently support RenderStream Local.

EX range machines **only** support Notch and TD render engines.

Mapping

Mesh mapping is not supported on servers with **AMD** graphics cards.

Mesh mapping is only supported on servers with **Nvidia** graphics cards after r32.1.1.

Licensing

RenderStream Local Layers will show watermarked content by default unless the correct license is applied to the machine. Licensing is applied at the machine-level so every machine in your session that is expected to render RenderStream Local content must have the correct license. Which license you need depends on the machine type you have.

EX Range Machines

EX range machines **only** support Notch and TouchDesigner as render engines with RenderStream Local and only if a **Realtime Connector License** is applied.

VX and GX Range Machines

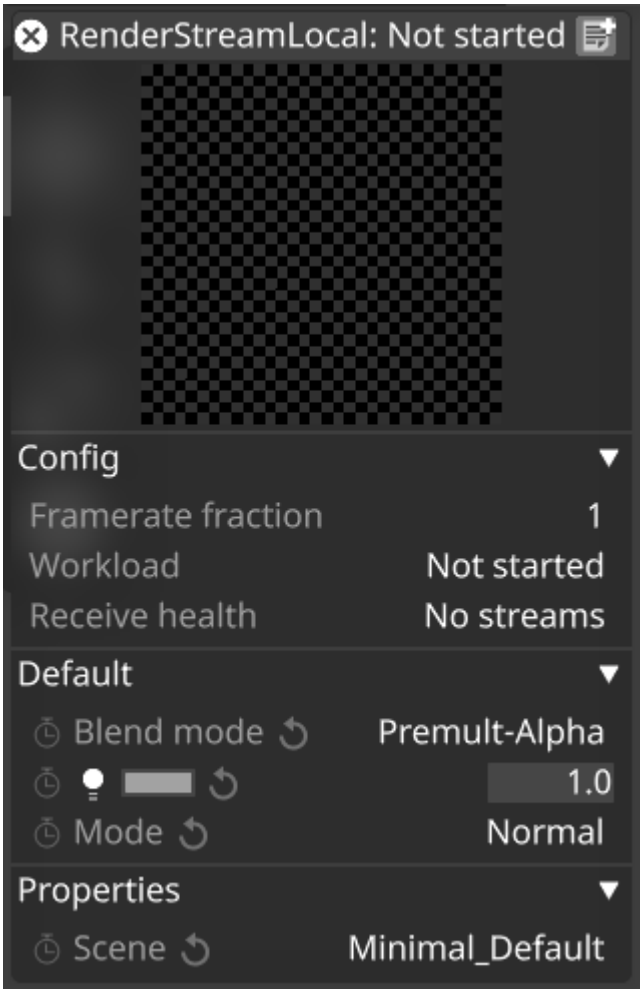
VX and GX range machines support all render engines and require a full RenderStream license applied to the VX or GX machine rendering the layer. This is the same license as a normal RenderStream license on an RX machine.

RenderStream Local Layer

The RenderStream Local Layer is used to control third-party render engines running externally to Designer locally on a VX or GX server.

At this time, the VX 4 and VX 4+ do not currently support RenderStream Local, as further development work is needed for AMD GPU support.

The RenderStreamLocal layer is a local RenderStream mode used to sequence instances of Unreal Engine, Unity, and TouchDesigner using the same RenderStream plugins as RenderStream in remote mode. Support for Notch is built into Designer.



There are four main sections within the RenderStreamLocal Layer:

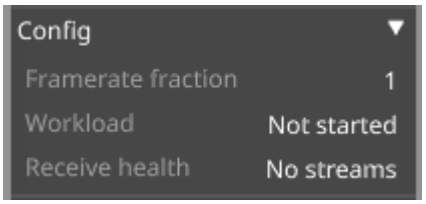
- Media Thumbnail
- Config
- Default
- Properties

Media Thumbnail

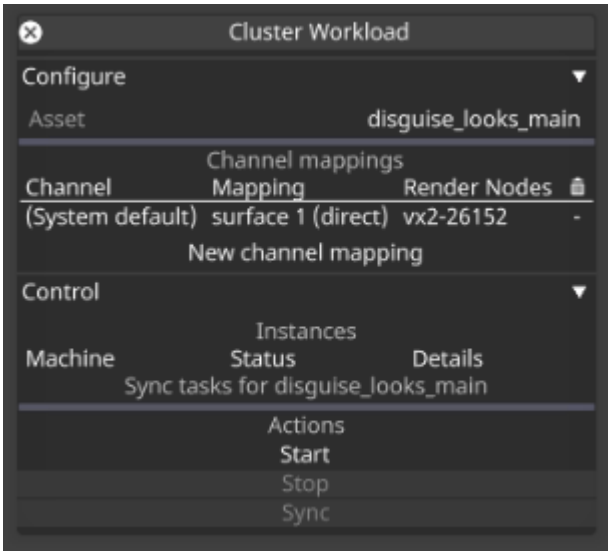
This area will display an image of the active stream being received. Note that in a multi-channel environment, this will only show the Camera in UE that has the RS component attached to it.

Configuration

This section describes the configuration properties, real-time workload monitoring and engine health monitoring.



Cluster Workload



The Cluster Workload window contains the elements required for configuring a workload. It contains **Configure** and **Control** sections.

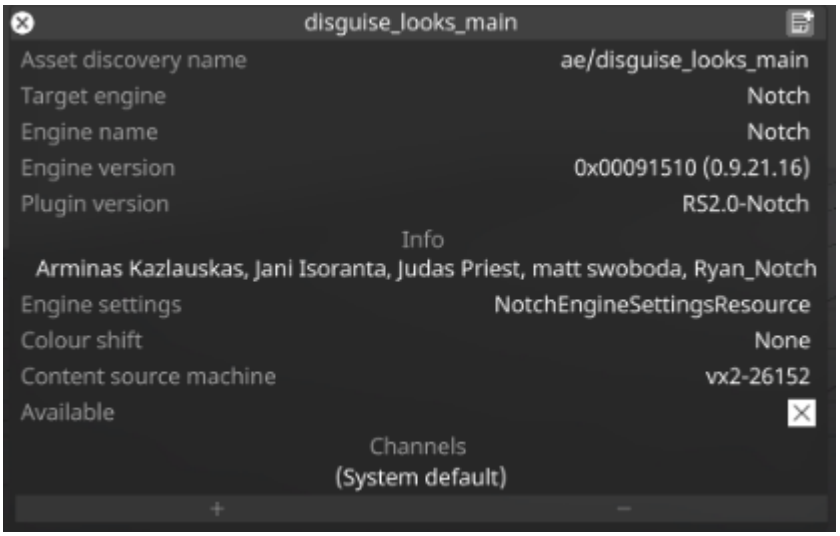
Configure

The Configure section contains elements to set up a workload.

Asset

This is the Render Engine project or Notch block that will be controlled by the layer. In order for the asset to be visible here, it must be present in the RenderStream projects folder of the render machines.

Right-click on the asset name to open the **Asset** properties editor:



Properties include:

- **Asset discovery name**: Name of the folder/project.
- **Target engine**: Name of the Render Engine.
- **Engine version**: Engine version.
- **Plugin version**: Plugin version.
- **Engine settings**: Engine-specific settings.
- **Colour shift**: Apply colour correction to the entire asset.
- **Content source machine**: The render machine that is running the render engine project. This is the machine that the project will be synced from.
- **Available**: Status indicator displaying if the Asset is available. Not a tick box.
- **Channels**: Active channels defined in the render engine project.

Channel Mappings

Channel Mappings include properties such as:

- **Channel:** active channel, editable field.
- **Mapping:** active mapping, editable field.

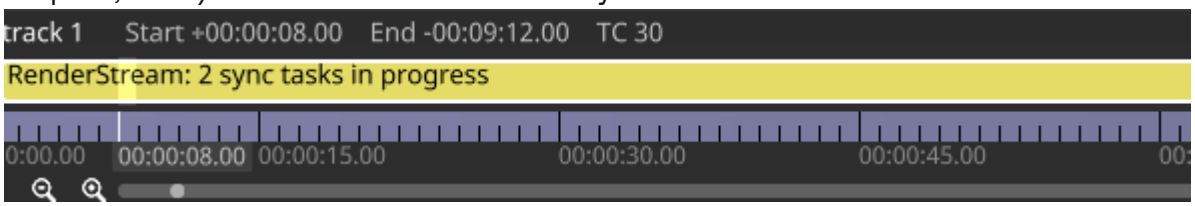
- **Render Nodes:** list of render machines.
  - This property has a **Manual machine selection** option that is disabled by default. In this case machines will be automatically added to the list when the content is mapped to the machine's output display head in the Feed Editor.
  - If **Manual machine selection** is enabled, the machine list field becomes editable. It will override the auto-generated list created by the content mapping in the Feed Editor.

Control

The Control section allows for managing the workload.

- **Instances:** When started, the Instances table will be populated with the machines that have been sent the start signal.
- **Start:** sends a signal to all machines in the Render Nodes list to launch the Asset according to config settings.
- **Stop:** Sends a signal to cease outputting a stream and quit the process sending it. The workload log can be opened by left-clicking on the status within the Instances table
- **Sync:** begins the process of content sync. Content is synced from the source machine to all other available Render Nodes; it copies only the necessary directories and files needed.

  With r31.0.3, a new workload status has been added to show sync progress (in progress, complete, failed) on the RenderStream Local layer.

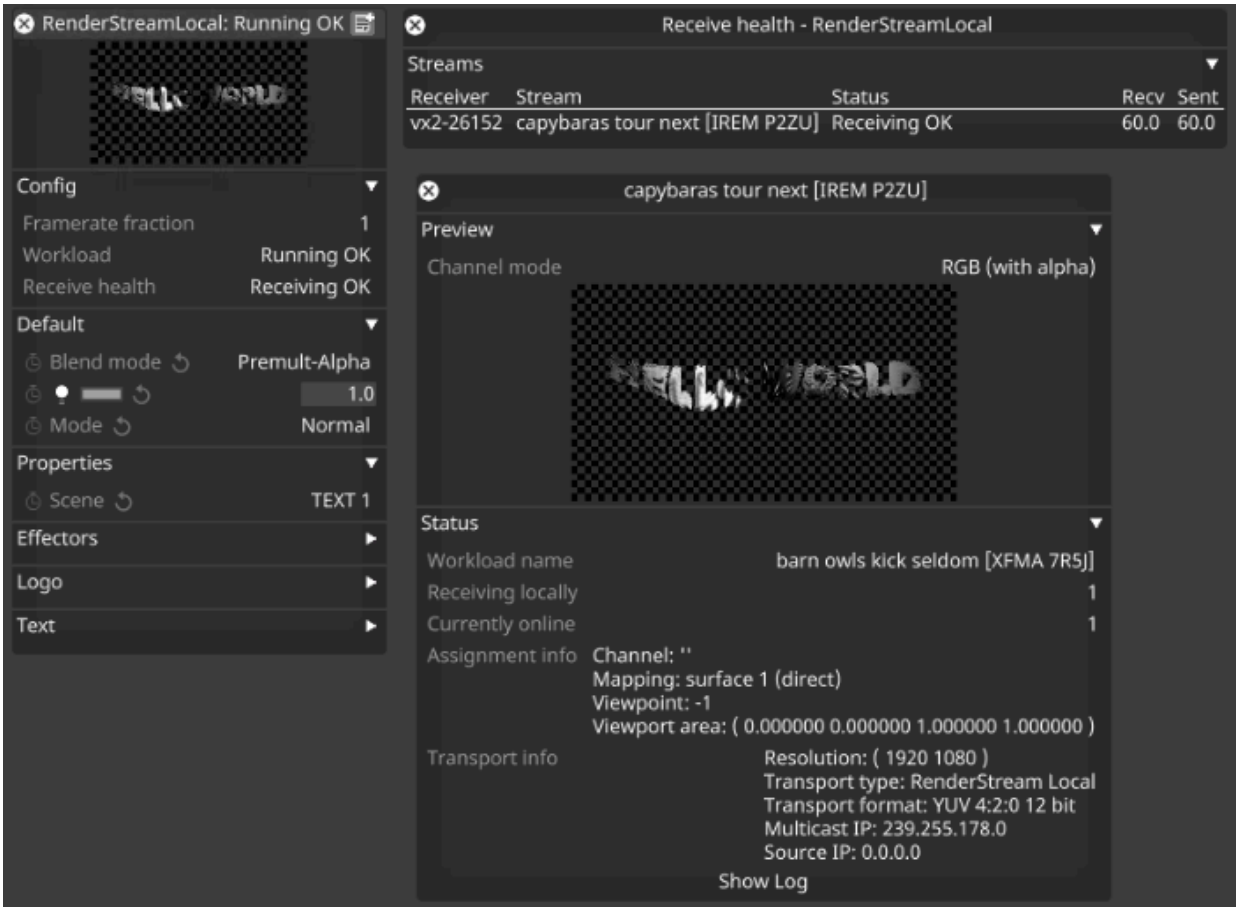


> **Note**
>
> Any changes to the settings require a restart of the workload.

Receive Health

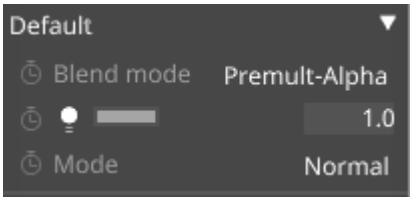
The statuses of all render machines are dynamically reported while the stream is running.

Right-click on **Receive health** to open the machine-specific window:



Default

In the Default section of the RenderStreamLocal Layer, there are four common layer properties:



- **Blend Mode**: Premult-Alpha is needed for content that contains alpha data.
- **Brightness**: Layer intensity.
- **Mode**: This is used to determine the timeline's playback behaviour as the layer is playing.

Properties Section

The last section of the RenderStreamLocal layer is asset-specific and will display all exposed parameters from the asset, as well as their keyframe editors

> **Note**
>
> The current release version does not support remote texture sharing.

Transport Format Differences

RenderStream Local operates exclusively at the highest transport quality supported by Designer. Unlike remote RenderStream workflows, the transport format is fixed and cannot be adjusted.

Key characteristics are as follows:

- YUV 12-bit is always used and is the only supported transport format.
- 8-bit and 10-bit transport formats are not supported in RenderStream Local.
- Alpha is enabled by default and is always included in the stream.

Render Engine Workflows

Set up your environment, including installation of the Render Engine and required plugin.

Use the RenderStreamLocal Layer to control the third-party render engine.

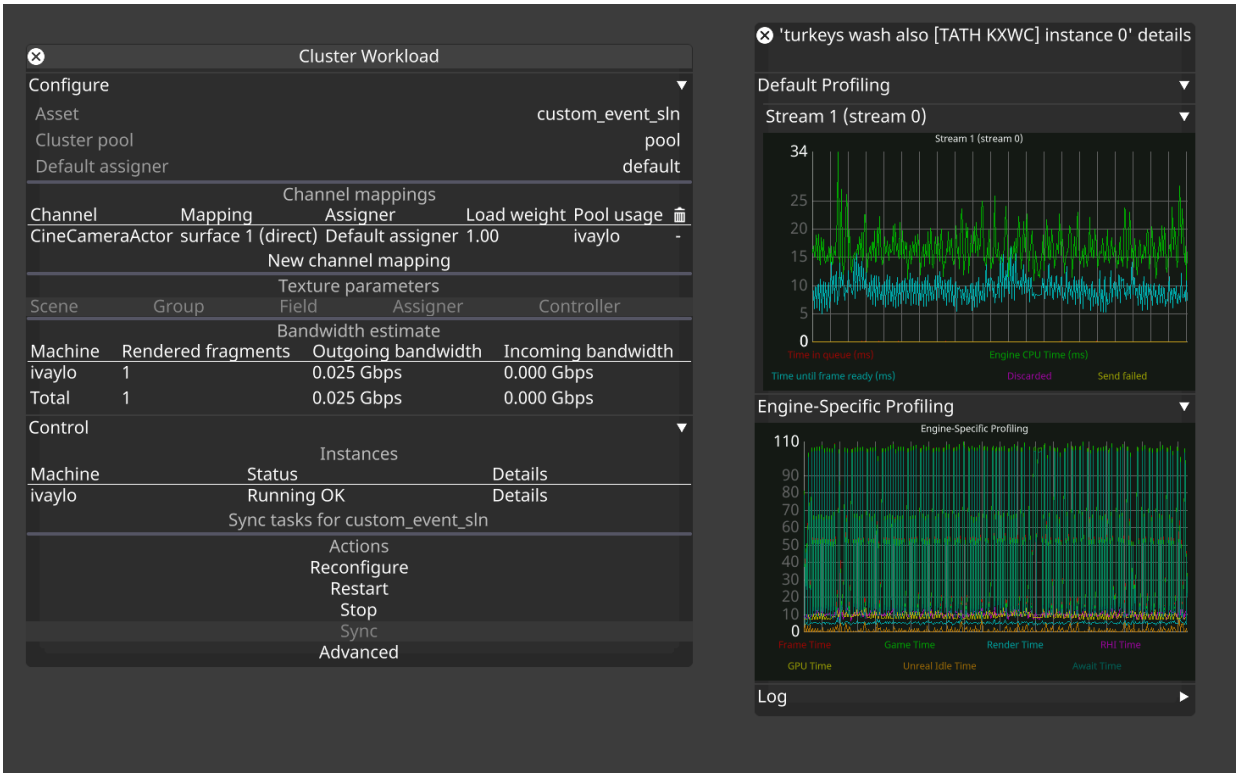
- **Notch**
  - Plugin - included in Designer.
  - Documentation
- **Touch Designer**
  - Plugin
  - Documentation
- **Unity**
  - Plugin
  - Documentation
- **Unreal Engine**
  - Plugin
  - Documentation

Understanding RenderStream Graphs

Where can I find RenderStream graphs?

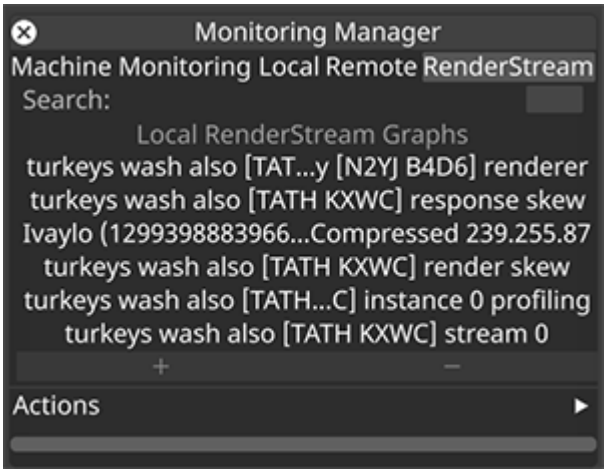
There are two places where we can find RenderStream graphs:

1. The **Details** view of one of the machines in a workload
   - shows engine and stream profiling for a specific render node machine
   - can be accessed via the 'Details' button next to a specific machine in workload widget



Workload widget and the stream profiling graphs that are accessible from it

2. The 'RenderStream' tab of the Monitoring Manager
   - lists all graphs for all streams and workloads
   - graphs are grouped by actors receiving the streams
   - can be accessed via the Monitoring Manager (left-click on FPS bug in Disguise)

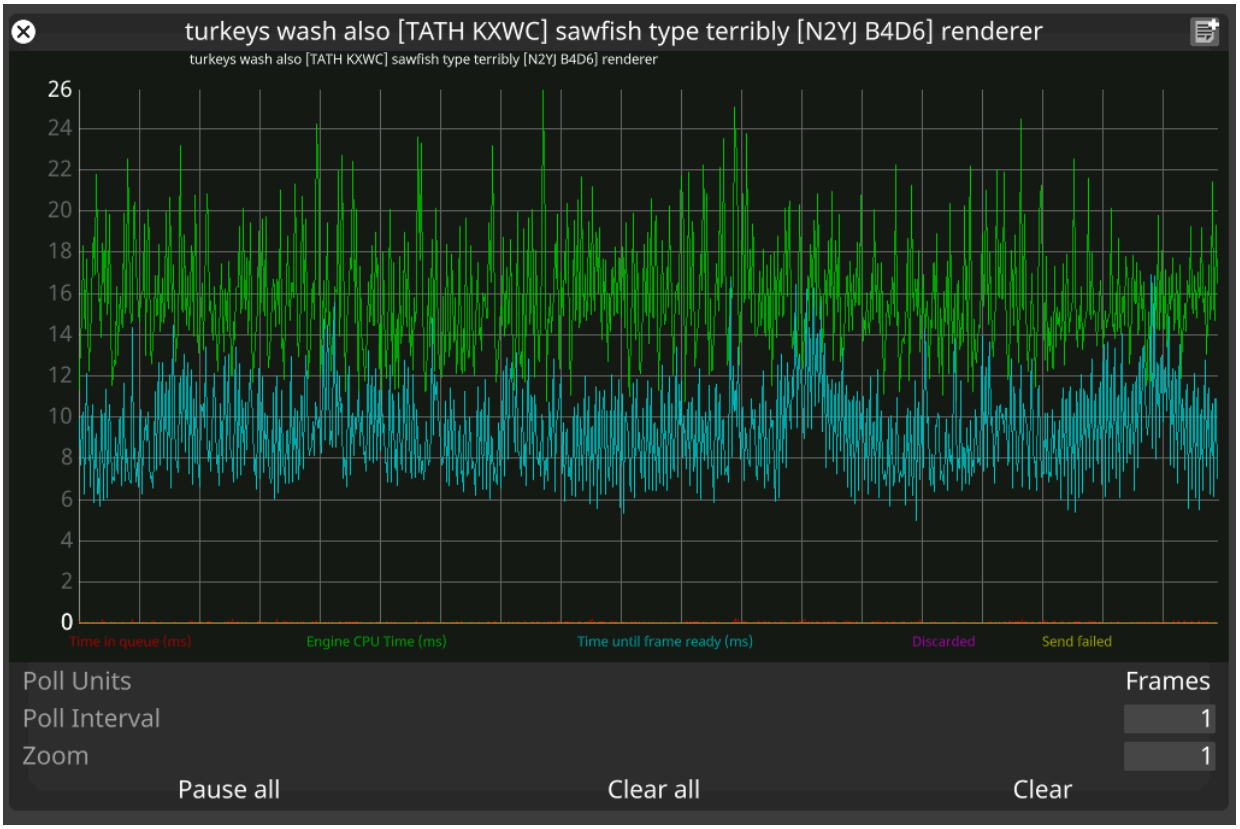


Monitoring Manager

Graphs available for RenderStream in the

RenderStream graphs

Renderer graph



What is it?

Engine-agnostic profiling data captured by the RenderStream plugin from the render node.

Time in queue (ms)

The time a frame request spends waiting to be picked up for processing by the workload instance.

Engine CPU time (ms)

How much time the frame request spends being processed. It is the time between the frame request being picked up for processing and the call to send the resulting frame back to disguise.

Time until frame ready (ms)

How long it takes to send the frame.

Discarded

Shows the frames dropped by the render node.

Send failed

Shows problems in sending the frames back to disguise.

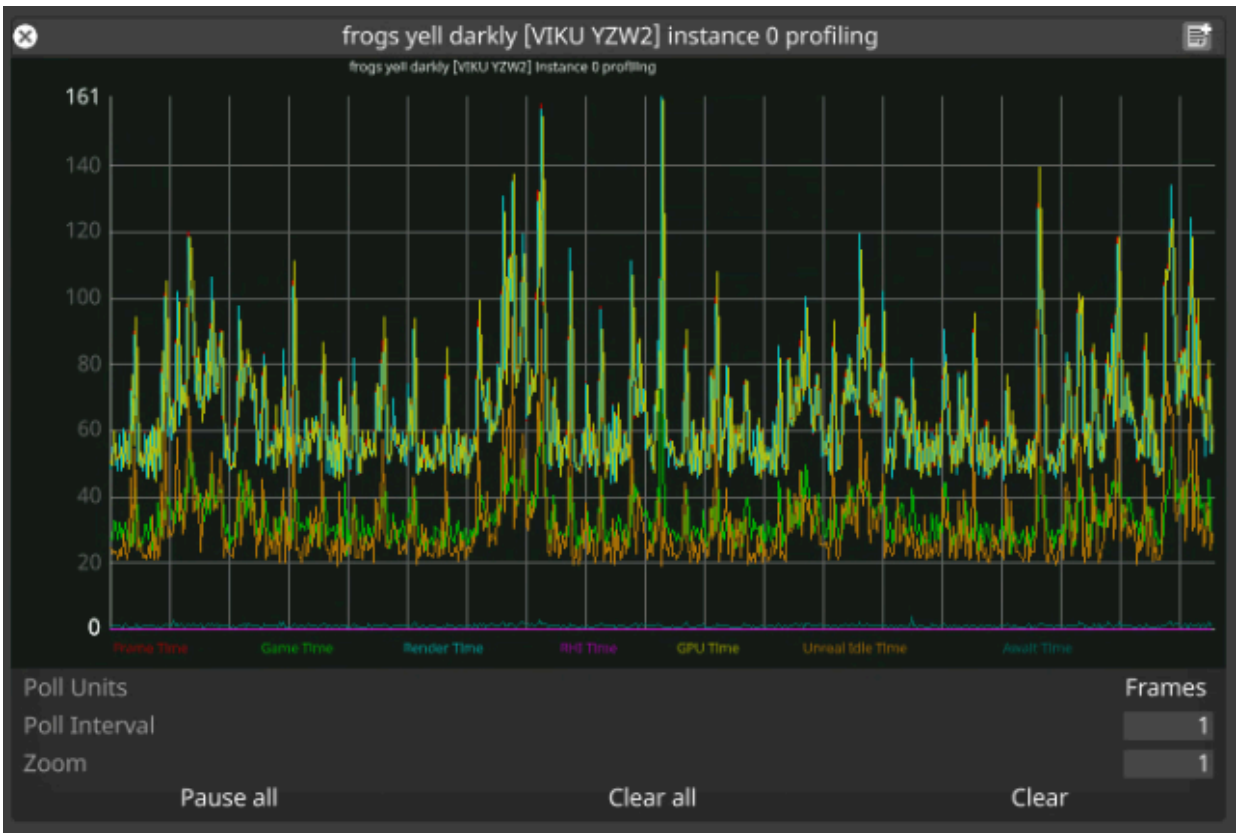
How do I use it?

Normally, time in queue should be minimal with most of the time spent generating the frame indicated by **Engine CPU time (ms)**. No requests should be discarded or marked as send failed.

High **Time in queue**: engine can't keep up with the rate of incoming requests from disguise. Eventually, if too many requests are waiting in the queue, the render node will discard request, which will be reflected in the **Discarded** line in the graph.

High **Time until frame ready**: indication of sending/networking issues

Profiling graph



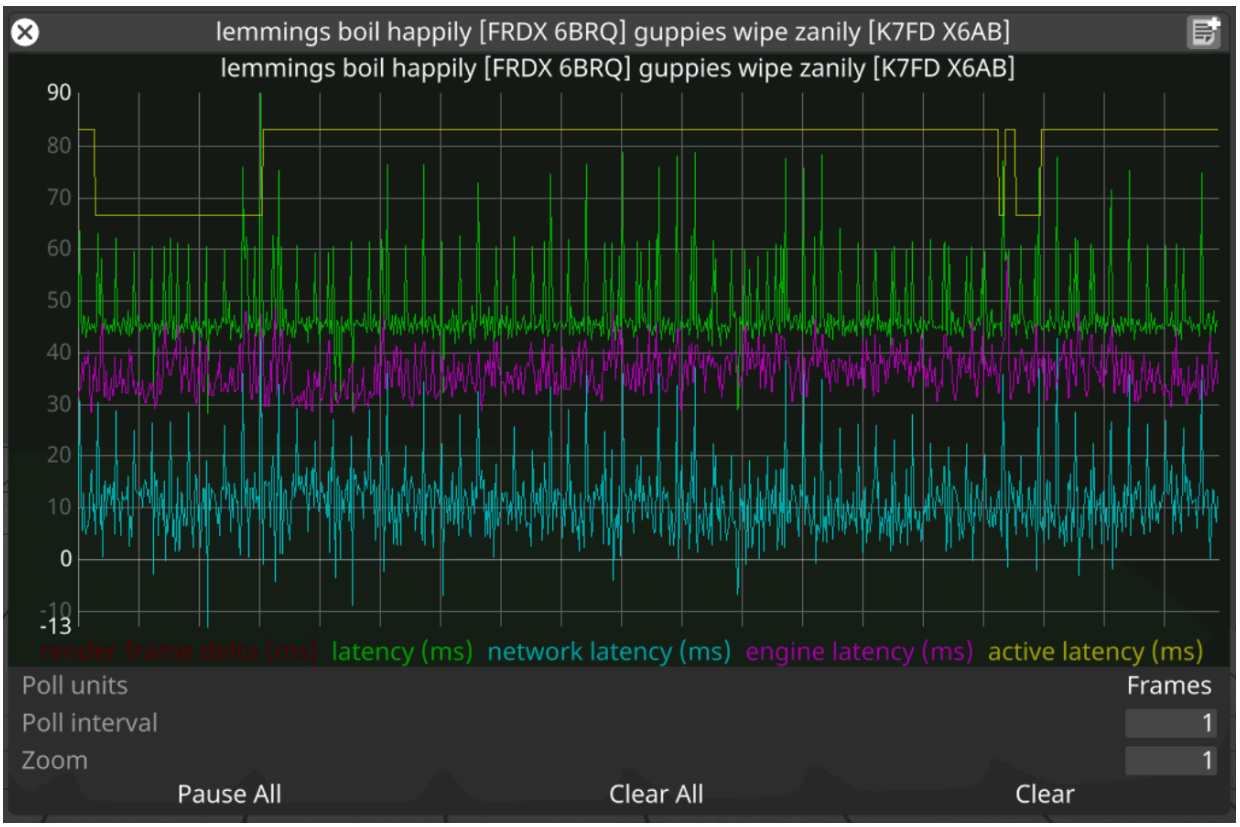
What is it?

Engine-specific profiling information for a workload instance. This data is specific to each RenderStream plugin implementation, or could be missing depending on the implementation as well.

How do I use it?

The data we get completely depends on what engine data a particular plugin implementation is tapping into. In the case illustrated above, we have Unreal Engine with granular data about how much time each step of the UE pipeline is taking. Abnormally high values in any of these sections could indicate high CPU usage (blueprints, geometry, physics, etc.), high GPU usage (resolution, post-processing, etc.), or other issues. For example, high Unreal Idle Time could indicate network slowdowns in nDisplay synchronisation.

Stream graph



What is it?

Data generated by the receiver of a stream (actor/director) describing the latency of the stream.

render frame delta (ms)

Shows the difference in timestamps between arriving frame and the frame that arrived before it.

latency (ms)

Actual latency of a stream: the time elapsed between disguise requesting a frame and then receiving back the frame for that request.

active latency (ms)

The current active latency which is based on the slowest stream in the session.

How do I use it?

Render frame delta should be constant (at 60fps, delta should be 16.6ms). Fluctuations identify either dropped frame requests or unstable framerate on the RenderStream controller machine.

Latency should not be too far off from the active latency. If there is a stream that is significantly faster than other streams, its latency will show as a lot lower than the active latency. If the

difference between the two is greater than 4 frames worth of time, effort should be made to better balance the workload. Also, if using a manual active latency override, this could be an indication that the override value is not accurate.

Active latency and latency should not fluctuate wildly. That could indicate optimisation is required on the asset side.

Render skew graph

What is it?

The graph shows how well temporally matched are the frames received across the streams of a workload.

Active latency error

Shows how close to the active latency prediction are the displayed frames of a RenderStream workload. This is the difference between the expected timestamp based on active latency and the actual timestamp of the closest matching group of frames.

Streams error

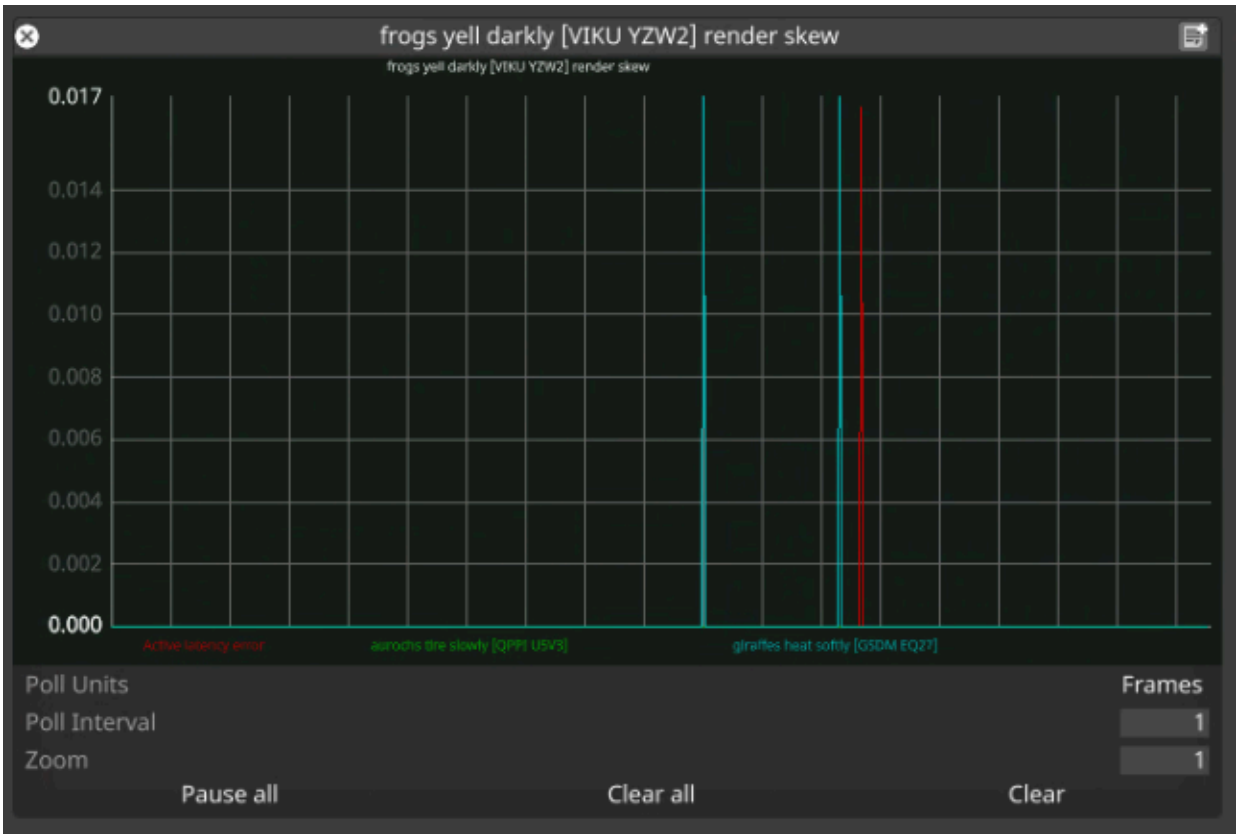
Within the best matching group of frames, these lines indicate the individual stream errors.

How do I use it?

Ideally, active latency error should be zero with the individual stream errors being less than a frame worth of time.

- If individual streams have error greater than a frame worth of time, this could be perceived as a tear between fragments that belong to the same actor.
- If the active latency error differs across multiple subscribing actors (across multiple instances of this graph), this could be perceived as a tear on the edge between the outputs of two actors.

Response skew graph



What is it?

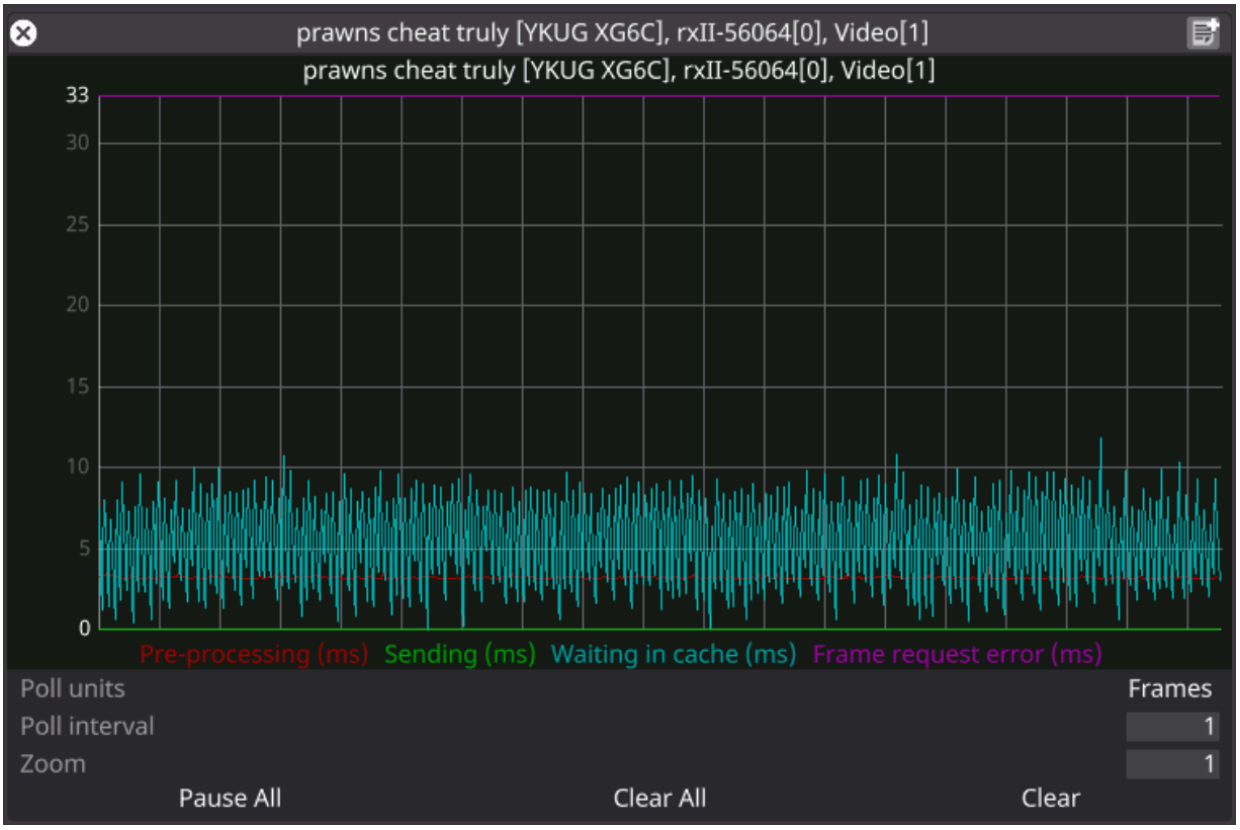
Shows how close to the active latency prediction is the currently processed RenderStream response. This is the difference between the expected timestamp based on active latency and the actual timestamp of the closest matching group of frame responses (frame sent notifications from render nodes).

How do I use it?

This can be used as another indication how well the active latency estimation fits a stream. Ideally, the error in this graph should be less than a frame worth of time. More than a frame of error could indicate network issues on the d3net network adapter, as it is the one used for

sending RenderStream frame requests and receiving the sent notifications replies from render nodes.

Texture parameter latency graph



What is it?

This graph is shown for every texture parameter stream. It shows the latency of texture parameters on their trip to the render nodes.

Pre-processing (ms)

Amount of time spent preparing the texture for sending on the controller machine.

Sending (ms)

Amount of time the sender took to dispatch the texture to the network card on the controller machine.

Waiting in cache (ms)

Amount of time the texture parameter spent waiting to be processed by the render engine on the render node.

Frame request error (ms)

Difference between the current frame request timestamp, and the texture parameter timestamp that the render engine is using for this frame.

In practice, this error tends to be the sum of the latency accumulated during all previous stages of the texture parameter pipeline, including the actual network transmission time.

How do I use it?

Texture parameters experience additional latency on top of the regular stream latency (see Stream graph). The texture parameter latency graphs reveal this additional latency and can be used to diagnose network or processing performance issues specifically relating to texture parameters.

Extending RenderStream

One of the most powerful parts of RenderStream is that it can be extended to support new render engines. In this page we will discuss how to extend support to new render engines.

Features

In this section we will discuss the features of RenderStream you should understand before embarking on integrating it with your chosen render engine.

Features of the RenderStream API

RenderStream is an ecosystem where a Designer server will send timestamped “frame request” messages to a render engine. These messages are accompanied by frame metadata that describe how to render the scene. The metadata types we support include:

- Camera transforms
- Text
- Numbers
- Video Textures

We expect render engines to listen for frame requests via our API and respond to them by submitting finished textures back to the API.

The renderers we support are:

- DirectX 11
- DirectX 12
- OpenGL
- Vulkan

Behind the scenes, RenderStream manages the sending and receiving of those textures over the network to display the images the render engine provides to the Disguise media server.

The render engine can also share the following metadata with Disguise via the RenderStream API:

- Definition of what metadata the engine expects to use to render (Called the “Schema”)
- Live profiling data
- Live console logging
- Custom statuses of the engine

There are also special tools within the API to support clustered rendering in-sync between multiple instances of a render engine across multiple machines.

Features that are not part of the API

Designer and the RenderStream ecosystem work together to provide additional controls of rendering which you should be aware of but do not need to implement yourself:

- Designer is capable of discovering launchable assets that exist on render nodes. This is so they can be displayed to the user and launched/closed
- Designer is capable of launching and closing the render application
- Designer can also copy assets from one render node across a cluster of render nodes for distributed rendering

Existing render engines

Before writing your own implementation it may be valuable to check if your preferred engine is already supported.

Disguise provides in-house support for the following render engines:

- Unreal Engine
- Unity
- Notch

The following 3rd parties have also developed their own RenderStream integrations:

- TouchDesigner
- Volinga
- Spout (implemented by ZeroSpace Studios)

Documentation

If you want to get started with building your own RenderStream integration you can find our official developer documentation site **here**.

We support C, C++ and Python bindings.

C and C++ bindings

For C and C++ bindings you can visit our **RenderStream github** which includes the API definition and samples for all of our supported render engines.

Additionally the following open-source implementations can be used as guides:

- Unreal Engine
- Unity
- Spout

Python Bindings

A set of Python Bindings are available in the **RenderStream-py** project.

These bindings include a script to auto configure a new asset type (`.pyrs`) which will auto install the requirements.txt and configure a virtual environment for the python project to run in.

Custom Environment Variables

We support setting a custom list of environment variables for custom engines. This can be done by adding a .env file to the same directory as the asset executable.

This file needs to be in the format assetname_variables.env, all in lowercase.

Here is an example of a environment variable file:

```
NAME=VALUE # No spaces around the equals sign
NAME2=VALUE2

# You can also define a variable over multiple lines
MULTI="""
This is a
multiline variable
"""
```

Contributing

We welcome pull requests and comments through Github if there are extensions or additions you would like to see to support your implementation. If you would like to reach out to us for support please contact **support@disguise.one**.

RenderStream - Notch Integration

RenderStream is the proprietary Disguise protocol for controlling third-party render engines from Designer. This topic covers the steps specific to configuring the RenderStream workflow with Notch.

> **Warning**
>
> For cluster rendering it's recommended to use render nodes from the same Disguise product range, e.g. all RX series machines. Mixing of machines from different product ranges is untested and unsupported.

Notch is the third-party render engine available from **Notch.one**. Notch can be configured to run either internally via a Notch layer or externally via RenderStream. This topic will cover the RenderStream-Notch workflow. For more information on using a Notch layer, visit this **link**.

Notch Host

Notch Host is a secondary application that is included as part of the main Disguise installation process. The purpose of this application is to load Notch blocks externally Designer and create RenderStream output streams which can be accessed by all machines across the same network.

> **Note**
>
> **Please note**: Ensure the correct version is installed on all machines in the network, including render nodes.

Project Directory Setup

- Place Notch blocks in the following path: `C:\Users\USERNAME\Documents\Renderstream Projects` or `D:\Renderstream Projects` if using an RX hardware system.
- The Asset discovery system scans all sub-directories to identify content, so you can place all blocks in the `Notch` folder.

Configure a RenderStream Layer

1. Launch and create a new **RenderStream layer**.

2. For camera base mappings:
    1. Add a new camera to the stage.
    2. Create a new Camera Plate/Spatial mapping.
    3. Add the camera to the mapping.
3. Right-click the **Workload** field to open the **Cluster Workload** widget.
4. Expand the **Configure** separator.
5. Select **Asset** and choose your Notch block.
6. Right-click the Asset to open its editor:
    1. Set **Source Machine** to the machine with the asset (used for syncing content).
7. Select **Cluster pool** and create a new cluster.

Within the Cluster Pool

1. Add the desired machines.
2. Right-click on the **default** ClusterAssigner within the **Default assigner** field to open the **ClusterAssigner** widget.

Within the Cluster Assigner

1. Select the distribution strategy and video transport options (changes to transport settings require restarting the workload).

Within each Channel Mappings table

> **Note**
>
> Notch over RenderStream does not support multiple channels, only a single default channel will be available. When you create multiple channel mappings with the same channel, a unique notch block instance will be created for each mapping. This means the scene will be rendered multiple times.

1. Select **New channel mapping** and assign your desired mapping.
2. Repeat for all desired channel-mapping combinations.
3. Expand the **Control** separator.
4. If the machines in the Cluster Pool do not have the content or the project has changed, press **Sync**.
5. Ensure all **Sync tasks** are marked **completed**.
6. Click **Start**.
7. Wait for Workload status to switch to **Running** and confirm that all streams are received.

> **Note**
>
> **Please note**: Notch blocks can only be split if they contain an exposed camera. If you attempt to split a block that does not have an exposed camera across multiple render nodes, the content will simply be duplicated.

Official Notch Documentation

View the official Notch documentation:

- Exposed Parameters in Notch
- Camera Tracking & Exposable Cameras
- Exposing Lights
- Exposable Nulls (Objects)
- External control surfaces

RenderStream - TouchDesigner Integration

RenderStream is the proprietary Disguise protocol for controlling third-party render engines from Designer.

TouchDesigner is a visual development platform that can be used to create interactive media systems, architectural projections, and live music visuals, all in real-time. For more information on RenderStream support in TouchDesigner, please visit Derivative's website here.

Beginning with r22, Disguise added support for a new major real-time engine. Disguise's proprietary RenderStream infrastructure, enabling bi-directional data transfer between Disguise and third-party render engines, now supports the integration of TouchDesigner. This integration, made possible thanks to RenderStream's new support for Vulkan API, will allow users to bring a real-time 3D scene developed in TouchDesigner into Designer and build it into their final project.

RenderStream with TouchDesigner is supported in two different modes, RenderStream and RenderStream Local.

See How It Works

TouchDesigner Integration

- Requires TouchDesigner 2022.28040 or later
- Requires TouchDesigner Pro or TouchPlayer Pro license
- Requires Designer software version r22 or later

3 nodes have been added to for the integration. They are:

- **RenderStream In CHOP**

This node is the primary operator to control and configure a connection with RenderStream. Along with being the sync-point for when a frame starts rendering, it also brings in all of the control channels, and also takes a DAT with schema information to create controllable parameters within Disguise.

https://derivative.ca/UserGuide/RenderStream_In_CHOP

- **RenderStream In TOP**

This node is used to receive image data sent over RenderStream.

https://derivative.ca/UserGuide/RenderStream_In_TOP

- **RenderStream Out TOP**

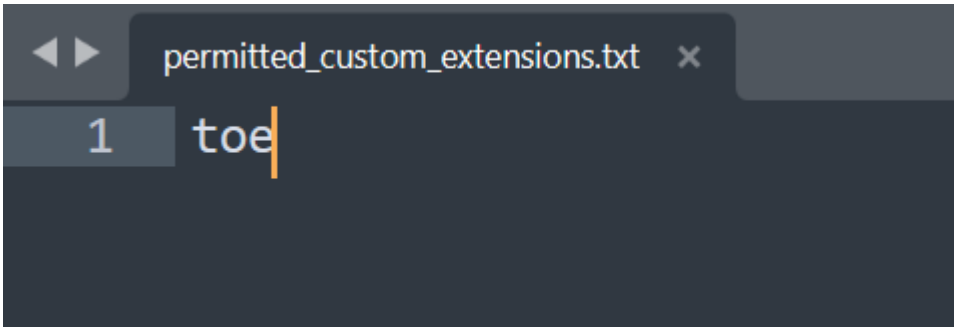
This node is used to send image data out to RenderStream.

https://derivative.ca/UserGuide/RenderStream_Out_TOP

Additional setup

In addition to the above, for TouchDesigner projects to be recognised as valid RenderStream assets, you must create a file called `permitted_custom_extensions.txt` and place it in the root of the `RenderStream Projects` folder. You must do this on any render node you want to discover the asset on.

This file will list the file extensions that Designer will identify as valid RenderStream assets, so you should add `toe` or `.toe` as the first line of the file.



Example of a simple permitted custom extensions file

You can add additional extensions if you want files with other extensions to be detected as RS assets. The default supported file extensions that do not need to be added to this file are:

- `.exe` - Unity and custom executables
- `.uproject` - Unreal Engine
- `.dfxdll` - Notch
- `.nvol` - Volinga

RenderStream Input - TouchDesigner Configuration

This page provides a quick start guide for setting up video input for TouchDesigner using RenderStream.

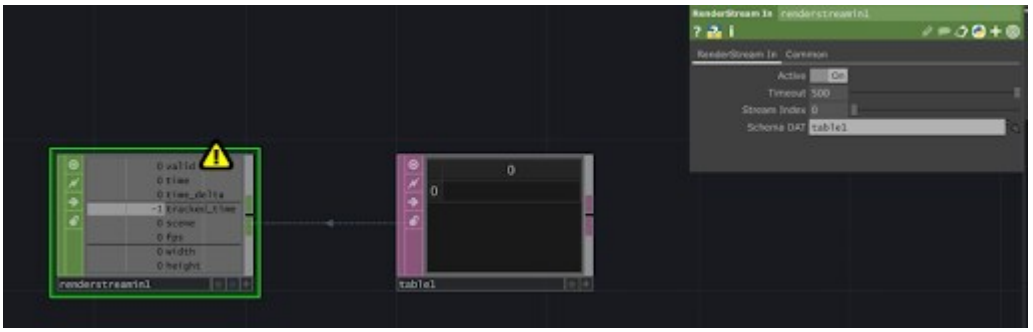
TouchDesigner officially supports RenderStream input and output. Here, we focus on input only.

Prerequisites

- Requires build 2022.28040 or later
- Requires Pro license
- Requires r22 or later
- Adding ‘permitted_custom_extensions.txt’ on the RenderStream Project folder, with ‘toe’ written

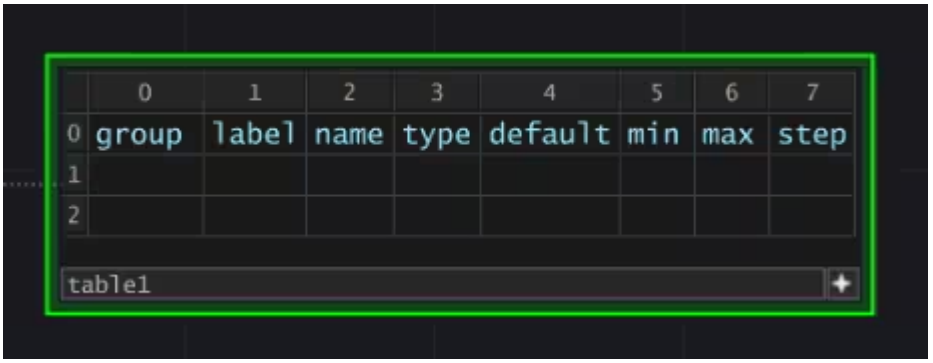
Configuration

1. Create **RenderStream In CHOP**.
2. Create **Table DAT**.

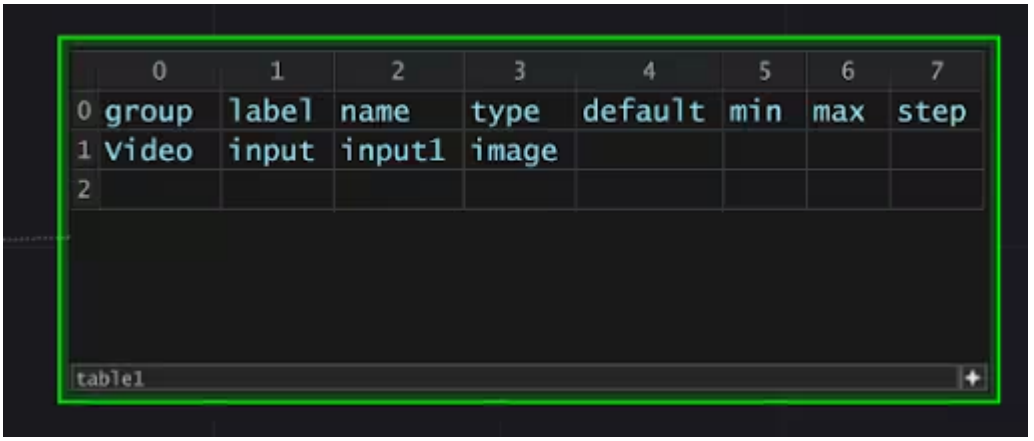


3. Add Table as **Schema DAT** for RenderStream in. Make a note of the node name.

4. Set table to be 8 columns.

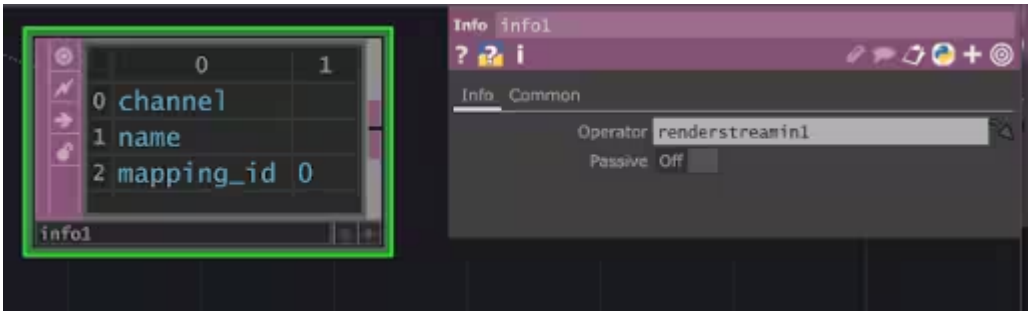


5. Set the first row with video in column 0, input in column 1, input1 in column 2, and image in column 3 as the image below.

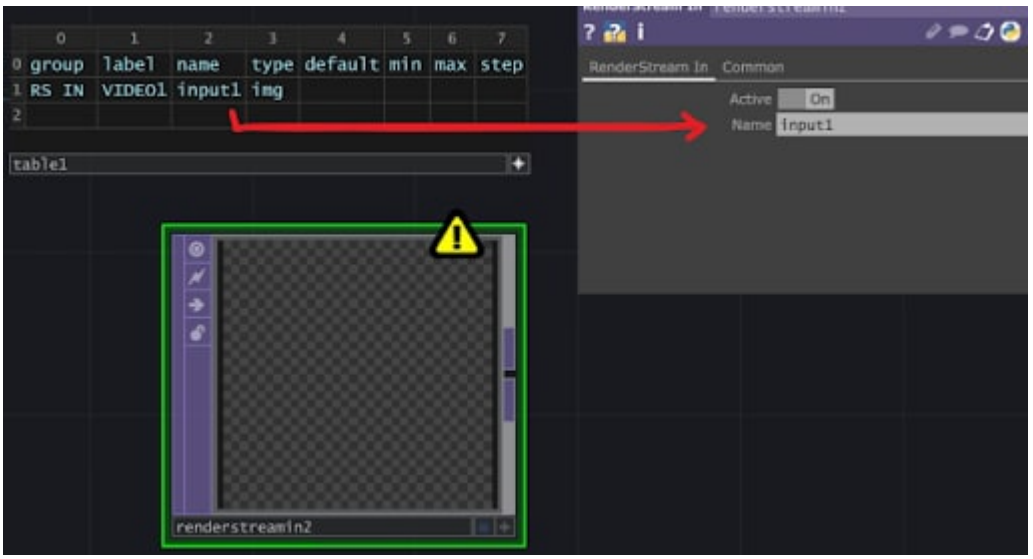


These can be exposed to d3. Note that that **Type** must be set to **Image**.

6. Create **Info DA**' and set its Operator as **RenderStreamIn** CHOP node.



7. Create **RenderStream In TOP**, and set the name to be the same as we set in the table.



These values will now be created as a JSON file.

8. Save the TD project, and run TouchDesigner with RenderStream layer in Disguise. The layer properties will show the values we set in step 5 above.
9. Set the texture using an available asset. or you can arrow the video layer into RenderStream layer.

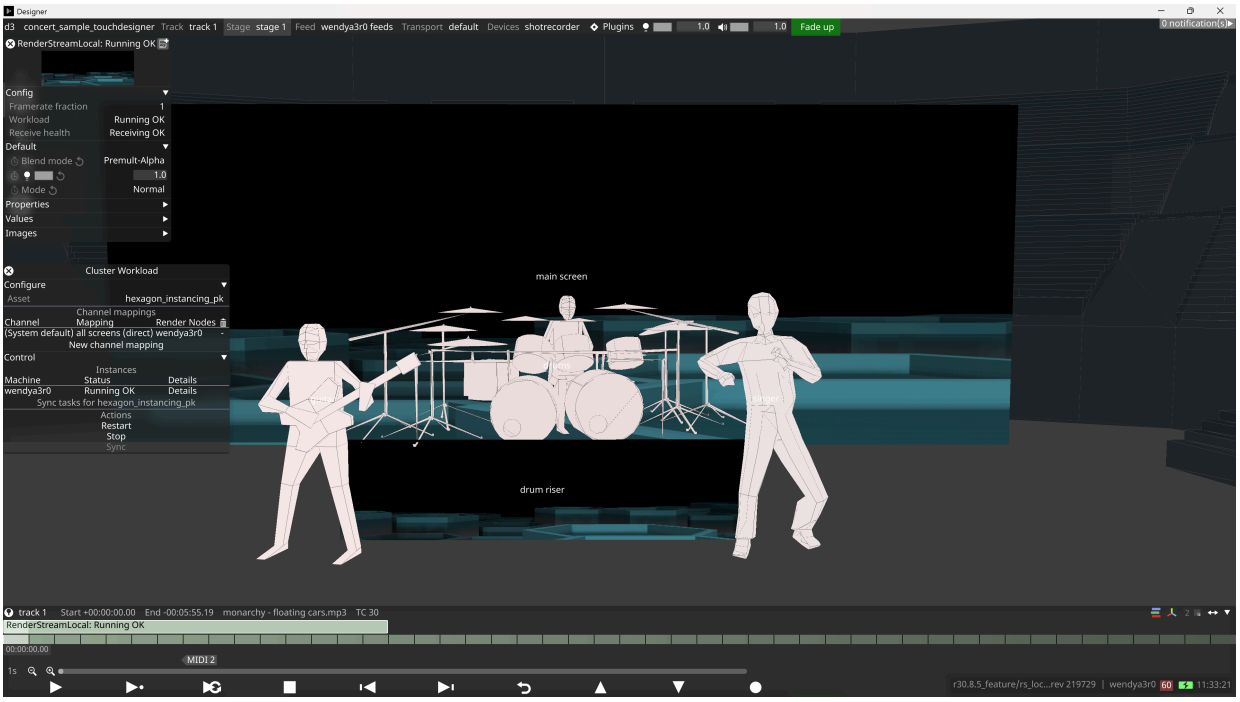
The **RenderStream in TOP** shows an image of the asset that was selected, which is sent from Disguise.

Now we can use it as a video source in TouchDesigner.

TouchDesigner on the EX range

The EX range supports the TouchDesigner RenderStream plugin only via the RenderStreamLocal layer. This is enabled with the **optional EX Real-Time connector (RTC) license**, which also enables a user to run a Notch block on the EX range. The Real-Time Connector license is purchased as a one-time cost. There is no need for a RenderStream license to enable this functionality. TouchDesigner using the RenderStream layer (where the RenderStream workload is deployed to an RX server) is not supported on the EX range.

TouchDesigner is the only RenderStream plugin type which is enabled on the EX range, provided the correct license is added. To use RenderStream integrations other than TouchDesigner, a VX or GX range media server must be used.



RenderStream Local with Touch Designer

TouchDesigner integration details are described at [RenderStream - TouchDesigner Integration]

Learn more about optional EX licensing **here**.

RenderStream - Unity Integration

RenderStream is the proprietary Disguise protocol for controlling third party render engines from Designer. This topic covers the steps to configure Unity with RenderStream.

For cluster rendering, we recommend the use of render nodes from the same media server product range, e.g. all rx series machines. Mixing of machines from different product ranges is untested and unsupported.

Unity is a third-party game engine that can be used to create 3D scenes. Unity projects can be adapted such that they create RenderStream output streams using Disguise's Unity plugin. This plugin supports both camera and non-camera based mappings.

A Unity Pro license is required. Please visit the **Unity website** for more information on purchasing licenses and training on the software.

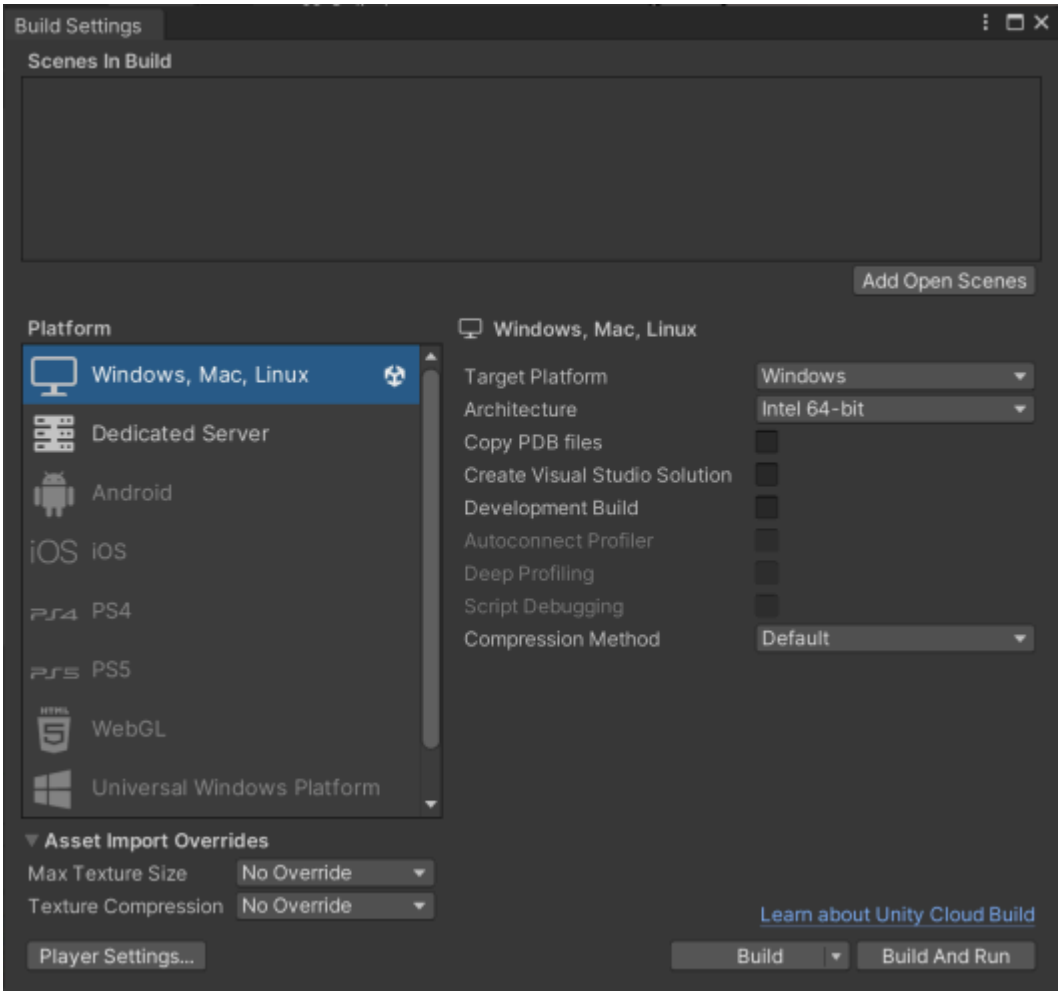
Plugins

- In order to communicate with Designer, Unity requires the installation of a plugin on the render node.
- The RenderStream pre-packaged plugins for Unity are available in the **Disguise RenderStream-Unity GitHub**. For the most up to date Unity plugin, you can compile the plugin from the source code under **releases**.
- Place the plugin into this folder : `PROJECT_ROOT/Assets/DisguiseUnityRenderStream`
- When adding a plugin to a Unity project, it is important that it is placed in the correct location and that the folder containing the plugin files is named correctly otherwise unexpected errors may occur.
- The available plugin uses DX11, `Direct3D11`. Unity has developed the integration further for DX12, `Direct3D12` which they are making available.

Unity project setup

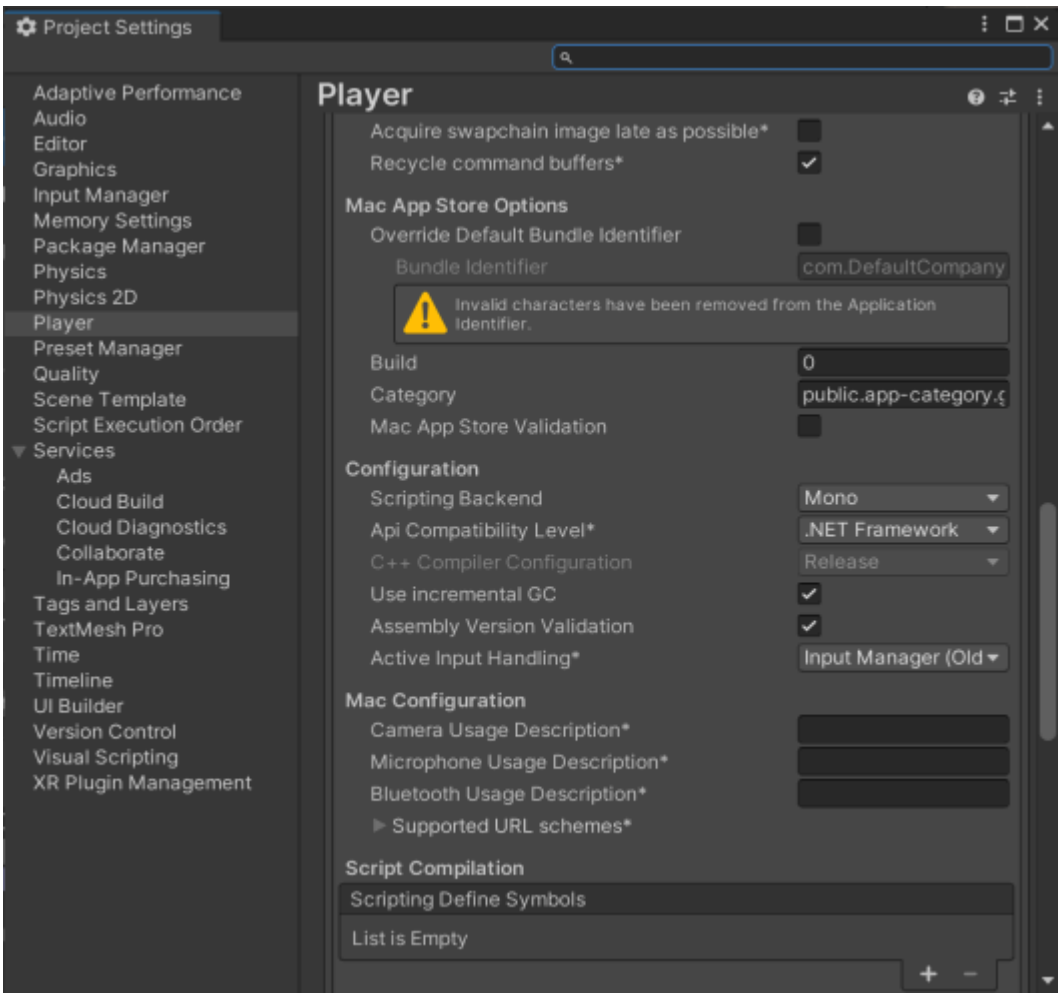
1. Launch Unity, navigate to Projects and select **New**.
2. Create a new 3D project.
3. Name the project, set the location to:
   `C:\Users\USERNAME\Documents\Renderstream Projects` or
   `D:\Renderstream Projects` if using a system with a media drive (e.g. RX) and select **Create**.
4. Open the project folder and place the plugin inside the 'Assets' folder.

5. Select **File** followed by **Build Settings**:
    1. Set **Architecture** to **Intel 64-bit**.



6. Navigate to **Player Settings...** and **Configuration**:

7. Set **Api Compatibility Level** to **.NET Framework**



Options

Set GameObject Channel visibility

1. In **Project Settings** > **Tags and Layers** and **Layers**.
2. Name an empty **User Layer**.
3. Select any object from the scene.
4. In the Inspector panel, add your new layer in the **Layer** parameter.
5. Select your Camera(s)
6. Select whether or not you want the Camera(s) to see the objects in your newly defined Layer by opening the **Culling Mask** dropdown from the Inspector panel.

Building the Unity project

RenderStream integrates with Unity builds rather than the Unity editor/game mode.

1. If you have multiple scenes in your project, ensure that the scene you wish to build appears in File > Build Settings.
2. Build the project.
3. Save the project and then close Unity.
4. Ensure that the build folder is copied to your RenderStream Projects folder.

Ensure the correct version of Designer is installed on all machines in network, including render nodes.

RenderStream Layer configuration in Designer

1. Create a new RenderStream Layer in your Designer project.
2. Select **Asset** and choose your executable.
3. Right-click the Asset to open its editor and set **Source Machine** to the machine with the asset (used for syncing content).
4. Select **Cluster pool** and create a new cluster.
5. Within the Cluster Pool, add the desired machines.
6. Select **Cluster assigner** and create a new **Cluster Assigner**.
7. Within the **Cluster Assigner:**, select the Asset.
8. Select **New Channel Mapping**.
9. Within each Channel's row, select the relevant mapping type, assigner and load weight.
10. If the machines in the cluster pool do not have the content or the project has changed, press **Sync**.
11. Ensure all **Sync Tasks** are marked completed.
12. Press **Start**.
13. Wait for Workload status to switch to **Running**.

Firewall settings can interfere with transport streams and/or cluster communication. You may see a Windows security dialog pop-up the first time you run the asset.Click Enable / Allow on this Windows pop-up to allow RenderStream communication to/from designer.

If the firewall is the problem check your outbound firewall rules on the Unity machine, and the inbound firewall rules on the Disguise machine for either software being specifically blocked.

Unity assets can only be split across multiple nodes when using 3D Mappings (i.e. Camera Plate or Spatial). Attempting to split using a 2D Mapping will not work; all nodes will render the entire frame.

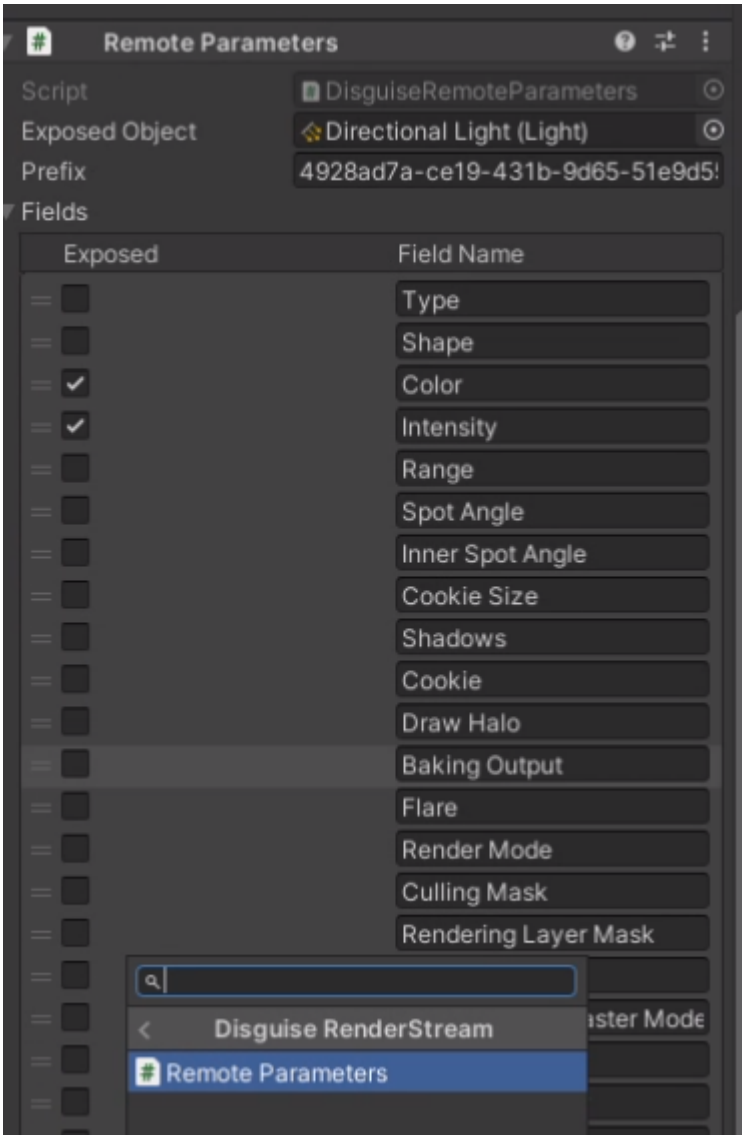
Exposed parameters

The Unity plugin allows you to expose certain options for each component of an object available within the scene. These options will be presented as parameters within the RenderStream Layer in Designer. Modifying the value of a parameter in turn changes the value

of the corresponding option in Unity thus altering the options of the selected object within the scene.

Exposing a parameter in Unity

1. Select an object from within the scene (e.g. any light source).
2. Select ‘Add Component’ at the bottom of the Inspector panel.
3. Add the ‘Remote Parameters’ component.
4. Drag and drop the component you wish to expose (e.g. Light) into the ‘Exposed Object’ field.
5. Expand the ‘Fields’ separator.
6. Select all options you wish to expose (e.g. Colour and Intensity).
7. Save all changes.
8. Build the project.
9. Save and close Unity.
10. Ensure that the build folder is copied to your RenderStream Projects folder.
11. Open the RenderStream layer in and start the workload.
12. Modify parameter value(s).

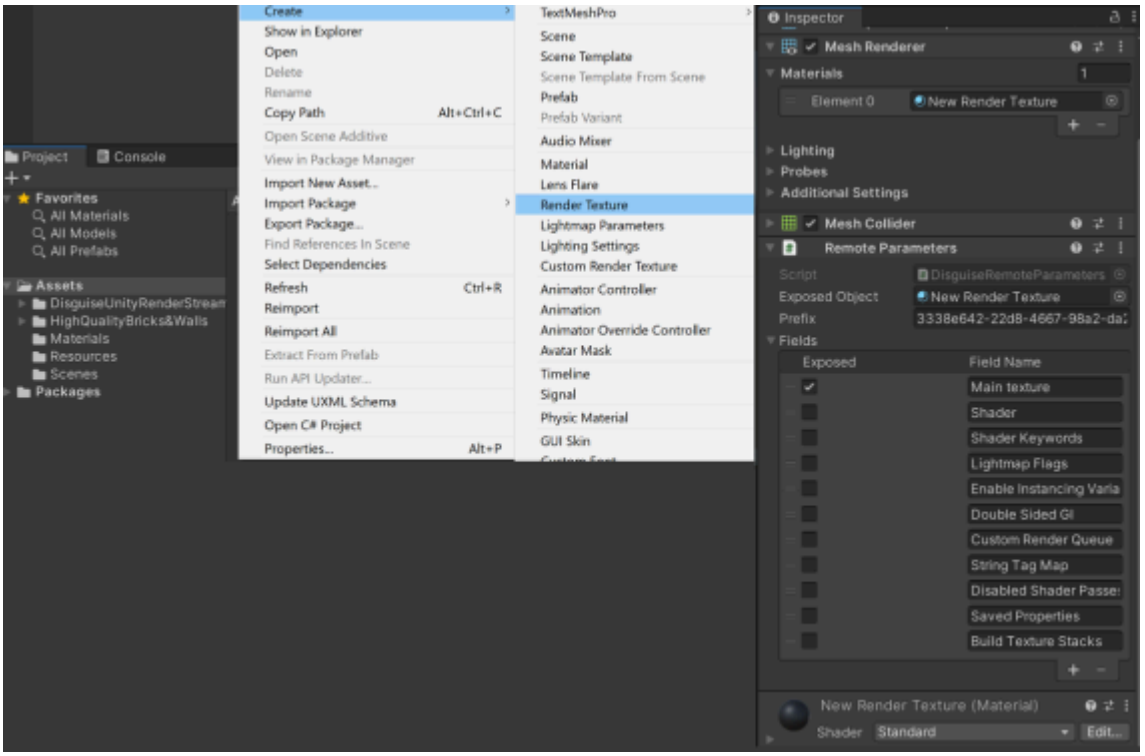


Remote Texture Parameters

The Unity plugins offers support for sharing textures remotely through the use of exposed parameters. This allows a two-way flow of video content between Designer and the Unity engin

1. [Optional] Add a Plane (or any other 3D game object) to the scene.
2. Create a new Render Texture: Select ‘Assets’ in the Project panel. Right-click inside and select Create > Render Texture.
3. Drag and drop the new Render Texture onto your Plane (or 3D game object of your choice) in the scene. Confirm that the new Render Texture has been added as a ‘Material’ component to your game object. Confirm that the new Render Texture has been set as a material element in the ‘Mesh Renderer’ component of your game object.
4. Expose the Render Texture as a remote parameter: Add a ‘Remote Parameters’ component to the Plane (or 3D game object of your choice). Drag and drop the new Render Texture (Material) component into ‘Exposed Object’. Open the ‘Fields’ separator and ensure that “Main Texture” is enabled (no need to enable any other fields).

5. Build the Unity project.
6. Save and close Unity.
7. Open the RenderStream layer in Designer and start the workload.
8. Create a new layer (e.g. Video) and assign media (e.g. Ada).
9. Move the new layer underneath the RenderStream Layer with Ctrl+Alt+down arrow.
10. Alt+drag to arrow the newly created layer into the RenderStream Layer.
11. Confirm arrowed input appears in the RenderStream content.



Screenshot showing how to create a Render Texture and expose it on an object.

Remote Parameters-3D Object Transforms

You can also expose a Unity GameObject's transform parameters; this will allow you to control the object's movements in two ways which is defined by the field you expose. When exposing a GameObject's Transform, you will have the options to expose the following fields:

- **Transform:** This allows you to control the full 3D transform (translation, rotation, scale) of the GameObject using a null object in Designer. This workflow is known as **3D Object Transform.** The null object acts as a 'proxy', allowing you to move objects in 3D via the on-screen transform handles, or by linking it to a dynamic transform data source, e.g. a tracking device.
- **Local Rotation, Local Position, Local Scale:** If you expose the local options these will give you the ability to keyframe the rotation, positions and scale in Designer allow for quick and easy manipulation of a Unity object on the Disguise timeline.

Note: If you expose all of the fields, the Local Rotation, Local Position and Local Scale will override the Transform in Designer. This will result in the null object not controlling the Unity GameObject.

Remote Parameters-Text Parameters

The Exposed Parameter workflow can also be used to expose live text parameters using a “3D text” actor in the Unity engine. You can use the Remote Parameters workflows to expose the text input field in Designer allowing you to edit the text in real time.

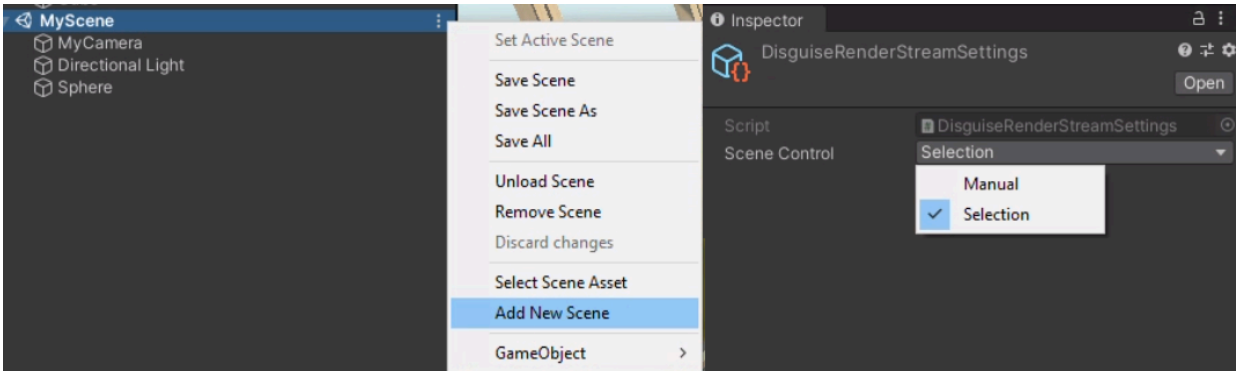
Scenes

Scenes in Unity can be composed of any number of game objects which are unique to that scene (by default). The Unity plugin offers two forms of multi-scene support:

Manual - this option restricts Designer’s control of scenes and instead merges all Channels and remote parameters into a single scene.

Selection - this option allows scenes to be controlled from inside Designer; Channels are merged into a single list (duplicates removed) and remote parameters are per-scene.

1. Create a new scene, or open an existing scene.
2. Select **Resources** then **DisguiseRenderStreamSettings** from the Project panel.
3. Set **Scene Control** option accordingly.
4. Ensure all required scenes are selected in the Build settings.
5. Build the Unity project.
6. Save and close Unity.
7. Ensure that the build folder is copied to your RenderStream Projects folder.
8. Open the in and Start the workload.
9. Modify the **Scene** parameter as part of your normal sequencing.



Time control

The Unity plugin offers Timecode support. This means that if any game object has a ‘Playable Director’ component and is animated using timeline functionality, it will be reflected in Designer

timeline.

Adding time control to an object in Unity

1. Select **Window**followed by **Sequencing** followed by **Timeline**.
2. Optionally, move the Timeline Window from the main panel to the bottom panel (i.e. drag and drop Timeline Window next to Console Window).
3. Select an object from within the scene (e.g. any user placed prop).
4. Click the **Create** button within the Timeline Window.
5. Save the new Timeline.
6. Hit the Record button on the newly created Animator for the chosen object within the Timeline.
7. Add an initial keyframe by right-clicking on any animatable property of the object (e.g. Position) and select **Add Key**.
8. Move the playhead along the Timeline.
9. Modify your chosen property either by using the 3D controls within the Scene or by updating the value directly from within the Inspector panel (a keyframe will be added automatically when a value is changed).
10. Hit the Record button again to stop recording.
11. Return the playhead to the beginning of the Timeline and play the sequence to confirm your animation is correct.
12. With the object still selected, select **Add Component** at the bottom of the Inspector panel.
13. Add the **Time Control** component.

Build and test time control

1. Build the project.
2. Ensure that the build folder is copied to your RenderStream Projects folder.
3. Save and close Unity.
4. Open the in and Start the workload.
5. Play your timeline in Designer.

Useful Unity information

- There is no need to attach the Disguise RenderStream script to the cameras in Unity. Cameras will auto-configure on asset launch.
- When using the ‘Manual’ scene selection option in Unity, game objects from all built scenes will not appear in the “Default” scene. In order to merge scenes and/or dynamically load/unload them, a custom script must be used.
- The “Default” scene will be the first indexed scene from within the ‘Scenes In Build’ table in Unity’s build options.

- The exposed parameters from all scenes will still show within Designer, even if game objects from all built scenes are not merged into the “Default” scene.
- When using the ‘Selection’ option in Unity, game objects and exposed parameters are unique to each scene. There is no shared object scene similar to the “Persistent” level in Unreal Engine.
- When launching a Unity executable for the first time, a Windows Firewall popup will appear. If the executable is not allowed through the firewall, Designer will not be able to receive the RemoteStream Game objects included as part of a Unity template may not be be able to be controlled via Timecode. This is a Unity issue rather than one with the Disguise script.
- When using the Unity High Definition Render Pipeline (HDRP), an alternative to the Universal Render Pipeline (URP):
  - Both the ‘Windows 10 SDK’ and ‘Game Development with C++’ modules must be installed as part of your Visual Studio installation.
  - The ‘Scripting Backend’ must be set to “Mono” when building the executable.
- Since Unity Assets are built executables, Disguise will not recognise the ‘Engine’ of them; they will simply be reported as “Custom”. Disguise will not be able to report the Unity plugin version used within built executables. If an incompatible Disguise-Unity plugin combination is used, no explicit notification will be shown.

RenderStream - Unreal Engine

RenderStream is the proprietary Disguise protocol for controlling third party render engines from Designer. This set of topics covers the steps needed for configuring Unreal Engine for use with RenderStream.

Unreal Engine is the third party render engine developed by Epic Games.

An Unreal Engine Scene on an xR stage visualised in Designer

This section covers a range of workflows to help implement real-time graphics for virtual production and xR.

Phase 1: Project & system foundation

Get your Unreal Engine project built and ensure the foundational Disguise plugins and directories are configured correctly.

1. UE Project Setup Learn how to correctly structure a Blank Game or Film project, install the RenderStream plugin, set up your CineCamera Actor, and add a RenderStream Channel Definition.
2. DDC Setup (Derived Data Cache) Optimise load times and performance by configuring your Derived Data Cache. This is essential for ensuring your render nodes don't spend unnecessary time compiling shaders.

3. **OCIO Setup (OpenColorIO)** Set up precise colour management pipeline rules ensuring what you see in Unreal Engine perfectly matches the output on your LED volumes or displays.

Phase 2: Scene configuration & control

Learn how to pass commands, textures, and parameters back and forth between Unreal Engine and the Designer software.

1. **Scene Levels** Understand how to manage Unreal Engine maps/levels and ensure the right environments are loaded via RenderStream.
2. **Exposed Parameters** Expose blueprint variables and properties so you can control your Unreal environment in real-time directly from the Designer timeline.
3. **3D Object Transforms** Link transformations so you can manipulate the position, rotation, and scale of 3D objects in your Unreal scene directly through Disguise.
4. **Remote Textures** Send live video feeds or dynamic imagery from Designer straight into materials inside your Unreal Engine environment.
5. **Remote Text Parameters** Update text objects dynamically from Disguise—perfect for live broadcast graphics or easily changeable virtual signage.
6. **Level Sequences** Trigger and control pre-animated cinematic sequences (Sequencer) inside UE directly from the Disguise timeline.

Phase 3: Optimisation & troubleshooting

Ensure your project is running smoothly, beautifully, and without framerate drops across your cluster.

1. **Multi-User Editing** Configure your environment for collaborative workflows so multiple artists can work within the same Unreal Engine scene simultaneously on a live stage.
2. **Scene Optimisation Verification Checklist** Run through this critical checklist to guarantee your scene hits the target framerate without dropping frames during a live shoot.
3. **UE RenderStream Troubleshooting** Encountering unexpected errors or dropped connections? Check this page for common issues, error logs, and immediate fixes.

UE Project Setup

This topic covers the steps needed for configuring an Unreal Engine scene for use with RenderStream.

Unreal Engine (UE) is a third-party game engine that can be used to create 3D scenes. Unreal Engine projects can be adapted such that they create RenderStream output streams using Disguise's UE plugin. This plugin contains several components that enable both camera based mappings with tracking as well as non-camera based (2D) mappings (e.g. direct).

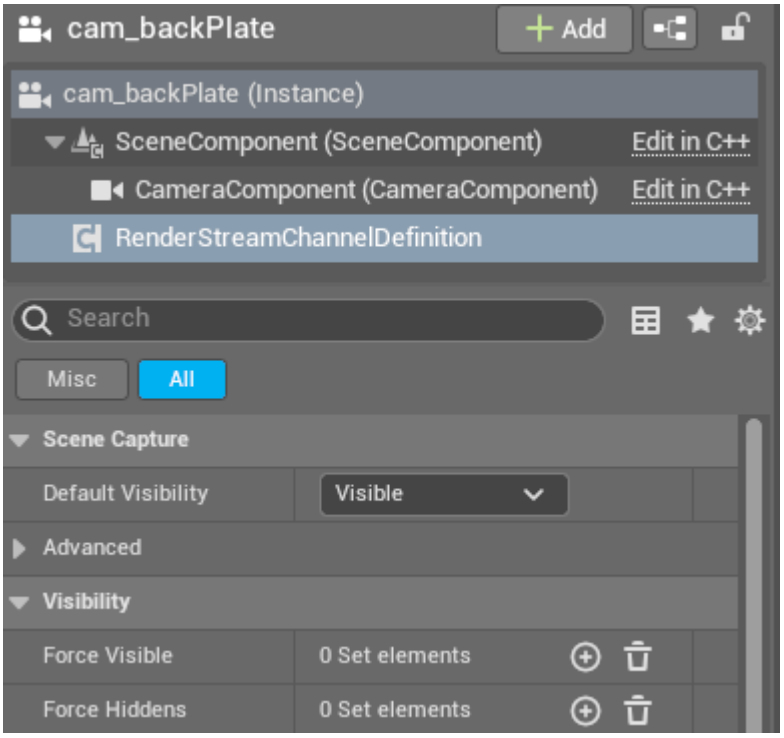
Plugins

- In order to communicate with Disguise, Unreal Engine requires the installation of a plugin on the render node.
- The RenderStream pre-packaged plugins for Unreal Engine are available in the **Disguise RenderStream-UE GitHub**.
- When adding a plugin to a UE/project, it is important that it is placed in the correct location and that the folder containing the plugin files is named correctly otherwise you may encounter unexpected errors. The plugin cannot be added as an engine plugin, it must be in the project.

Unreal Engine Project Setup

1. Launch the Unreal Engine Editor.
2. Create a new **Game > Blank**, or **Film > Blank** project with default settings and ensure it is saved in: `D:\RenderStream Projects` (or `C:\Users\USERNAME\Documents\RenderStream Projects` if you are using a personal laptop/PC).
3. Open the project folder in Windows Explorer and create a new folder with the name, `Plugins`.
4. Place the RenderStream plugin in the Plugins folder.
5. Restart Unreal Engine.
6. Search for **CineCamera** in the Place Actors panel.
7. Drag and drop a new **CineCamera** into the scene.
8. Click the Add Component button from the Camera Actor's Details panel and add a **RenderStream Channel Definition**.
9. The name of the Channel in Designer will be the name of the camera actor, so name the camera accordingly.
10. The setting Default Visibility defines whether the camera will see most actors in the scene.

11. Open **Window > Message Log > RenderStream Validation** and resolve the issues mentioned on the window, these are based on nDisplay limitations/features. Resolve these settings and save the scene making sure the notification has disappeared from the list.

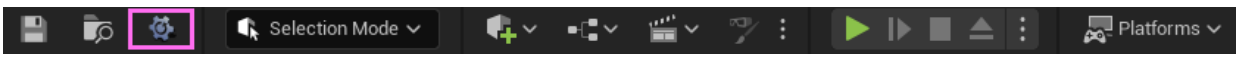


12. [**Optional**] Force the visibility of select Actors within the Camera's Channel Definition (useful for splitting content between the front and backplates): If you wish to add multiple Actors to a visibility set at the same time, lock the current selection into the Details panel, select the Actors you wish to add either from within the scene or the Inspector panel, right-click the visibility set you wish to add the Actors to, and select **Add from Selection**.
13. [**Optional**] To make Unreal Engine run at full speed when out of focus go to Edit → Editor Preferences and untick "Use Less CPU when in Background" (search "cpu").
14. Save the project.

Unreal Editor or Packaged Build?

The standard RenderStream workflow operates directly within the Unreal Editor, which offers flexibility and rapid iteration. However, as of July 2025, a new option is available directly from the UE Editor's Viewport Toolbar. This feature allows you to build and package your project into a standalone executable (.exe) for use with RenderStream, which can offer performance benefits and a more streamlined deployment.

When the build process is triggered, two .exes are generated by Unreal. One is a launcher, and the other is the actual project, which has a longer binary and will be nested in the Binaries folder. They will both appear in RenderStream's asset discovery.



Unreal Engine RenderStream Build button

There are some prerequisites to building a project that need to be observed before using this method:

1. **Environment Setup**: Install **Visual Studio Community 2022** and select the **"Game development with C++"** workload during installation, as this automatically includes the correct **Windows SDK** needed to build your project for **Windows 10** and **Window 11**.
2. **Project Configuration**: Inside the Editor, go to **Project Settings** to configure the core packaging rules. Here you must **Set Default Maps**, and under the **Packaging** section, choose your **Build Configuration** (e.g., Development or Shipping), explicitly list all **Maps to Include** in the build, and confirm **Windows** as the target platform.
3. **Asset & Lighting Finalisation**: With the settings configured, finalise all project data. This involves ensuring all shaders are compiled and, if you are not using Lumen, select **Build Reflections**. It's important to **Rebuild Lighting** on "Production" quality if you are using baked lighting. Alternatively, select **Build All Levels** to update everything at once.
4. **Final Save & Build**: Finally, perform a **Save All** to make sure every change is committed. After saving, you are ready to initiate the final build process.

For a more detailed explanation of the packaging process and the differences between engine versions, please consult the official Unreal Engine documentation.

Useful Unreal Engine Information

Saving a project

Unreal Engine has several methods for saving a project. When making changes to a project for use with RenderStream, it is best to close the project before attempting to start a workload. This will prompt you to save all changes to your project. If your Unreal Engine project is not saved correctly, you may experience unexpected behaviour.

Blank GAMES project or blank FILM project?

Note that a key difference between a blank GAMES project and a blank FILM project is the way that project manages the skybox. A Games project has an auto-updating skybox relative to the sunlight (Directional Light) for a day/night cycle, but a film project has a refresh material button for updating the skybox relative to the sunlight.

Derived Data Cache (DDC) Setup

Configuring a Shared Derived Data Cache (DDC) will improve the startup speed of an Unreal Engine project shared across a RenderStream network. This process minimises the lengthy compilation process for Unreal projects.

The DDC is where shaders for Unreal Engine projects are compiled and stored. When opening a new project on a standalone workstation or on a Disguise render node, shaders need to compile for quite some time, even if the project is very simple.

One of the benefits of working with identical, standardized hardware like the Disguise RX range of render nodes is that the DDC can be compiled on one machine and then shared across a network to other nodes in the RenderStream cluster, eliminating the need to individually compile shaders on each render node.

Shared Derived Data Cache (DDC) Setup

For Unreal projects created externally, if the DDC is not stored within the project, a single compilation is required after the project is copied to the server to create the DDC.

Create a DDC folder on the Director server

1. Create a DDC folder on the D Drive, ensuring the URL has no spaces in the path.

- Bad example: D:/RenderStream Project/DerivedDataCache
- Good example: D:/RenderstreamProject/DerivedDataCache

Apply folder sharing permissions

1. Right-click the DDC folder and select **Properties**.
2. Click on the Sharing tab, then click **Share**.
3. Select the people you wish to share with. In the input text field type `Everyone` and click Add. Everyone is a preset sharing permission.
4. Select **Everyone** from the list and right-click the Permission Level field. Select **Read/Write**.
5. Click **Share** to confirm the settings.

Mapping the Shared folder as a Network Drive

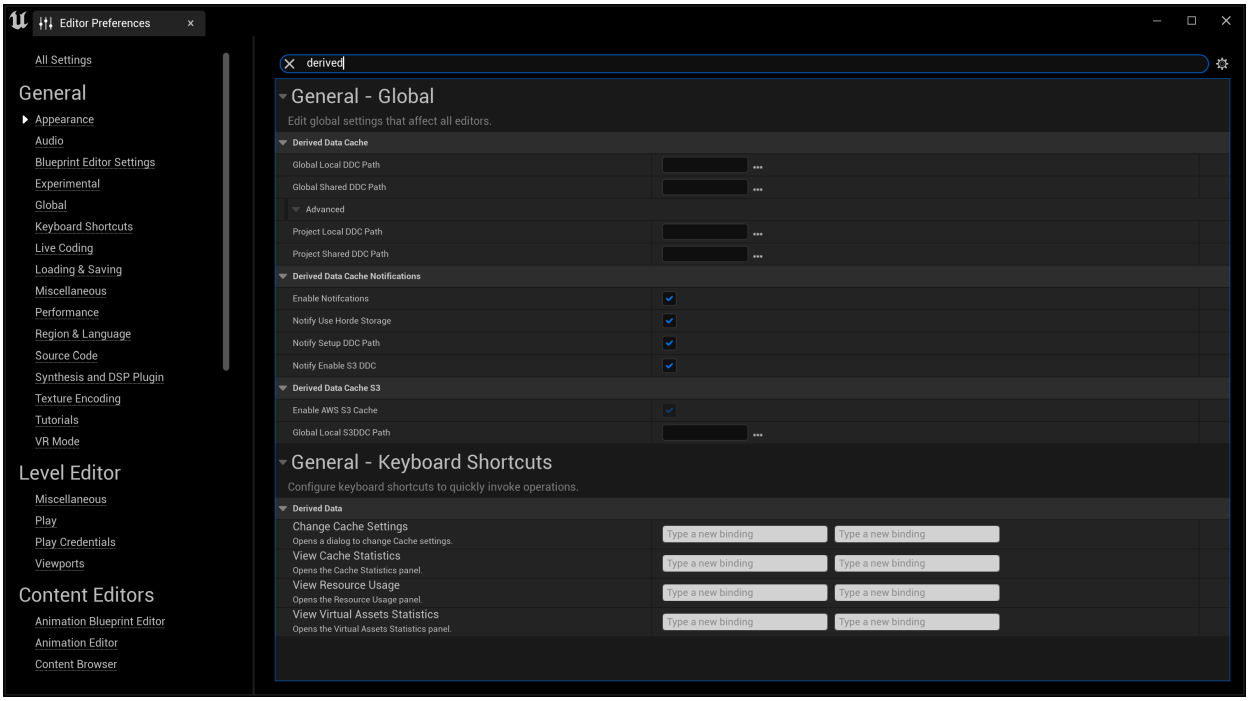
1. Type `IPConfig` in the CMD window and make a note of the IP address of the D port of the Director server. When creating a cache and storing data, you can choose the IP of the network you want to use.
2. Next, go to the Actor or RX server.
3. On the Actor or RX server, select **This PC > Open Computer** from the tab at the top.
4. Click on **Map network drive** in the Computer menu and select the option **Map network drive**.
5. Select the network folder tha you would like to map. For example, the **Z** drive.
6. Add the Director's IP address. An example for this uses the following format: `\\25.25.25.101`.
7. Click *Browse** and select **DDC** from the list of folders, then click **OK**.
8. Confirm he Z drive folder path, for example: "\25.25.25.101\DDC**
9. Click **Finish**.
10. Check that the DDC folder of the Director server is shared in **This PC > Network Locations**.

All connected Actors and/or RX servers must have the same mapped Network Drive.

Unreal Engine Network Drive location

1. In the Unreal Engine folder, go to **Engine > Config > BaseEngine.ini**.
2. Open **BaseEngine.ini** with Notepad.
3. Use **CTRL+F** to find where `DerivedDataBackendGraph` appears in the file.
4. Change the folder path to **Z:\DDC**

Alternatively, in Unreal Editor Preferences, under the General > Global header set the Shared/Local Derived Data Cache location to the same Shared and Local paths.



Derived Data Cache settings

Check DDC settings

1. Open the Unreal Project on the Actor or RX server and check if a cache folder is created in the Network drive DDC folder.
2. In the Unreal Project, click Derived Data at the bottom right of the editor, and select **View Cache Statistics**.
3. Confirm that the Z drive DDC path is listed on the Cache Statistics under **InstalledDriverDataBackendGraph > Name**.

The project also requires a single compilation after the project is first opened.

For details, please refer to Epic Games' DDC guidelines.

Local Derived Data Cache (DDC) Setup

Unreal Engine uses a local DDC by default, storing cached data on the user's machine. This setup requires minimal configuration:

1. Open the Unreal project.
2. Navigate to **Edit > Project Settings**
3. Search Derived Data Cache, and go to the Derived Data Cache section.
4. Specify the directory where the cache will be stored. Ensure there is enough disk space to accommodate the cached data.

Build the Derived Data Cache using the command line

Open a Command Prompt, enter the following [Change UE_5.o to whichever version you are using!]:

```
cd [_Drive that UE is installed to_]
```

```
cd [UE Install Location]\Epic Games\UE_5.0\Engine\Binaries\Win64\
```

```
UnrealEditor.exe "[ProjectFolderLocation]/[ProjectName].uproject" -run=DerivedDataCache -fill
```

You can click and drag the project file into the Command Prompt window instead of typing the [ProjectFolderLocation].

There are multiple storage types available for DDC. Read more about it in the official Epic Games documentation.

Unreal OCIO Setup

A key component of virtual production is maintaining consistent colour across the entire rendering pipeline. This page covers the steps required to set up OpenColorIO (OCIO) in Unreal Engine to achieve consistent colour management throughout the production process.

Before proceeding, ensure you have finished setting up the project by following the steps in UE Project Setup.

Requirements

RenderStream Plugin

- See UE Project Setup | Plugins for details on setting up the RenderStream plugin.

OCIO config file

- Place your chosen OCIO config file into the **Content** folder of your chosen Unreal Project.
  - `RenderStream Projects/<UEProject>/Content/<OcioConfig>.ocio`
- A standard OCIO config file can be downloaded from the OpenColorIO GitHub.

> **Warning**
>
> Both Unreal Engine and Designer must use the same OCIO config file to ensure colour consistency. If the OCIO config includes LUTs, ensure that both the config file and its associated LUT folder structure are preserved for both Unreal Engine and Designer.

Unreal Engine Colour Validation

To ensure accurate colour representation in your Unreal Engine project, it is recommended to utilise the Unreal Validation Framework from Netflix. This framework provides tools to validate and maintain colour consistency throughout your project.

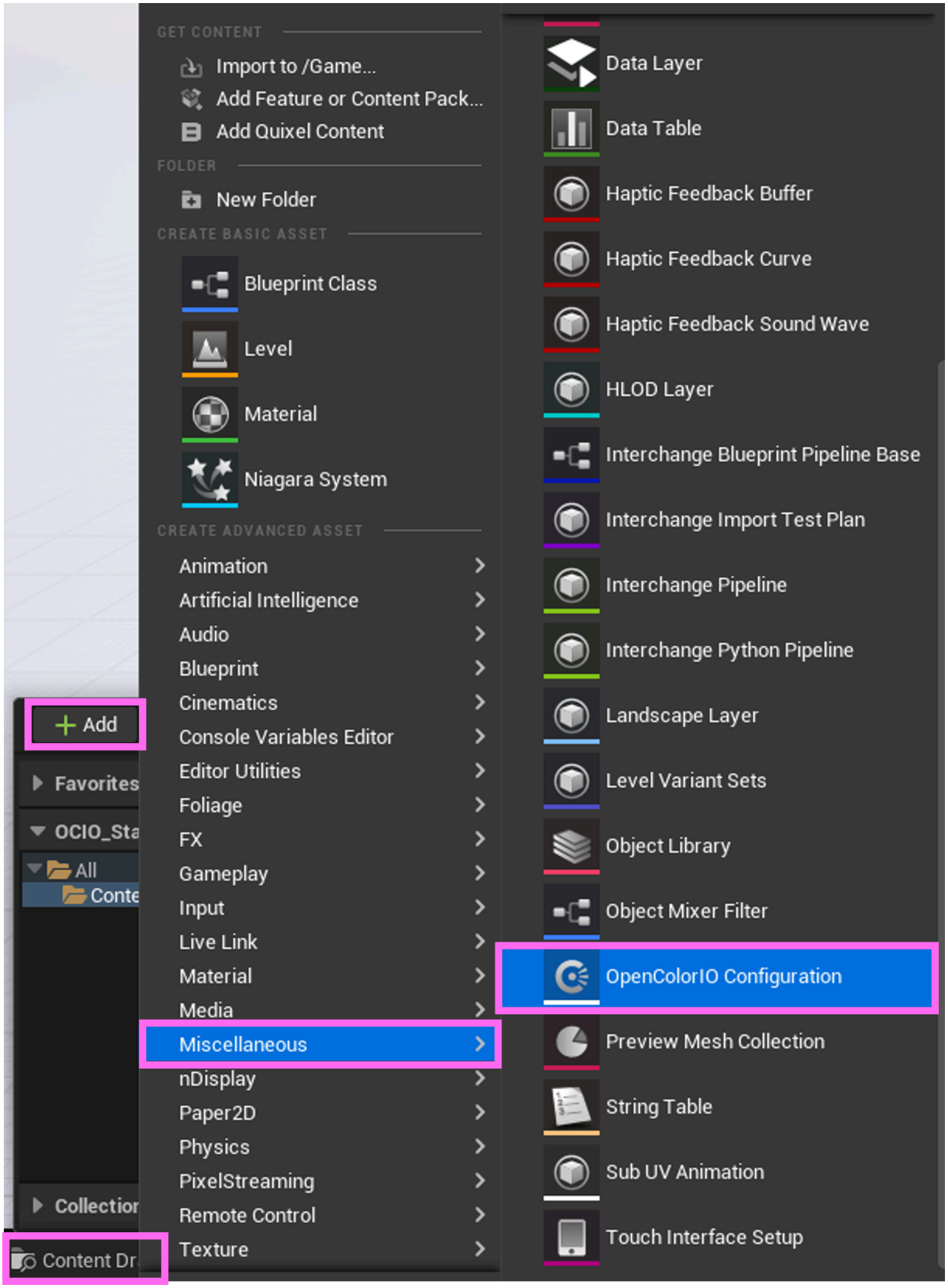
- UnrealValidationFramework GitHub
- A Validation Framework For Unreal Engine Blog

OpenColorIO Configuration

To use OCIO, Unreal requires an **OpenColorIO Configuration** to specify which specific file and colour spaces you would like to use.

Adding a new configuration

1. Open the **Content Drawer**.
2. Click the **Add** button.
3. Navigate to **Miscellaneous > OpenColorIO Configuration** and select it.

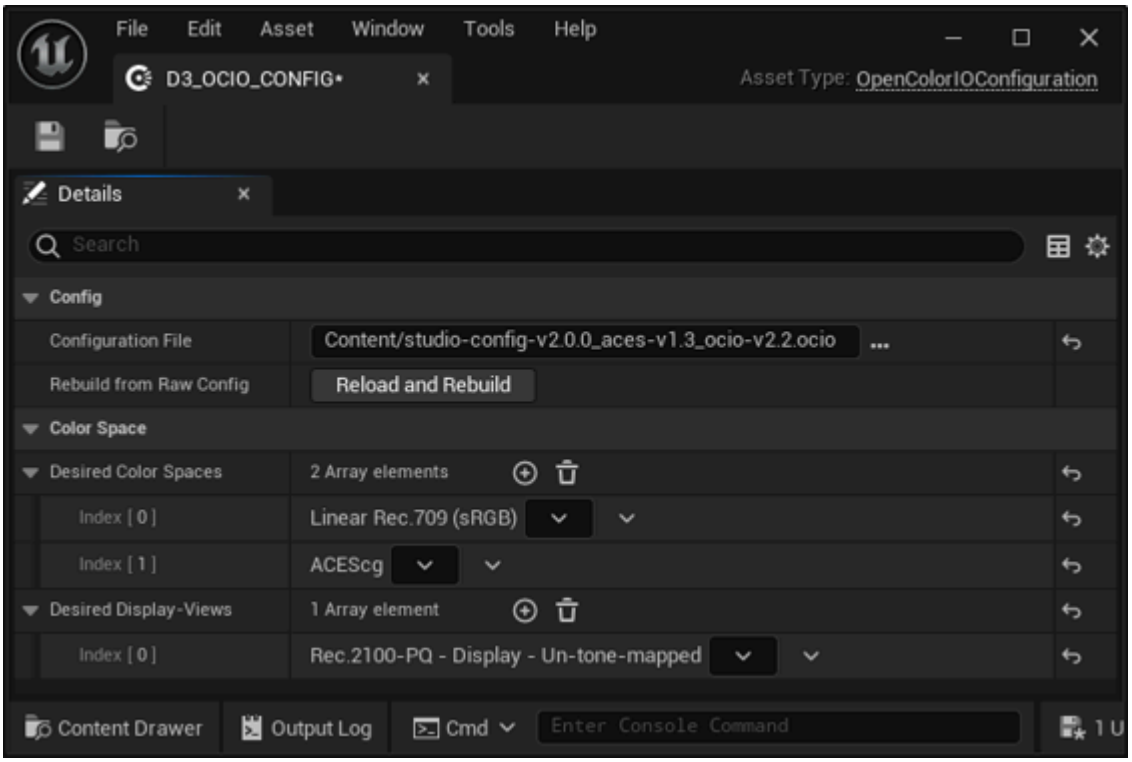


Add OpenColorIO Configuration

Prepare colour spaces

Next we need to expose the relevant colour spaces from the **OpenColorIO Configuration** to configure the colour spaces in the RenderStream Plugin.

1. In the **Content Drawer**, double-click the **OpenColorIO Configuration** you just added.
2. Under **Config**, set the **Configuration File** to the `.ocio` config file you added from the OCIO Config File section.
3. Add the **Desired Color Spaces** and **Desired Display-View** to expose the colour spaces and display-views for later use.



OpenColorIO Configuration Settings

Desired Color Spaces

Exposed **Desired Color Spaces** are typically used for source colour spaces in RenderStream plugin:

- Unreal Engine uses `sRGB / Rec709` by default as the **Working Color Space** in **Project Settings**.
  - The corresponding colour space in the OCIO config is `Linear Rec.709 (sRGB)`.
- Another typical **Working Color Space** is `ACES AP1 / ACEScg`.
  - The corresponding colour space in the OCIO config is `ACEScg`.

Desired Display-View

Exposed **Desired Display-View** are typically used for destination colour spaces in RenderStream plugin:

- `Rec.2100-PQ - Display - Un-tone-mapped` is recommended for common RenderStream use cases.
- The destination colour space refers to the colour space in which content will be sent over RenderStream

Project Settings

Once the **OpenColorIO Configuration** is set up, we must configure the **Project Settings** to apply the colour transforms properly.

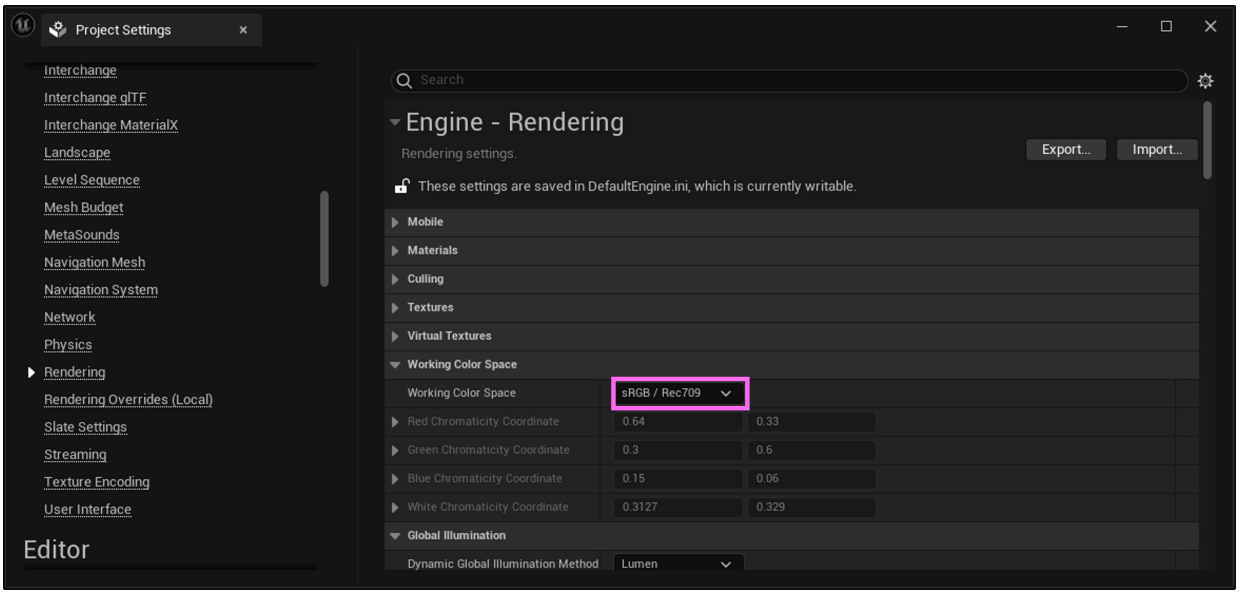
Navigate to **Edit > Project Settings**.

Engine - Rendering

In **Project Settings**, under **Engine > Rendering**, you can define the **Working Color Space** of Unreal Engine. All textures in Unreal Engine must match the colour space set as the **Working Color Space**.

For example, if the **Working Color Space** is `ACES AP1 / ACEScg`, then all textures must be in `ACES AP1 / ACEScg`. By default, Unreal uses `sRGB / Rec709`.

We advise sticking with Unreal Engine default colour space unless its strictly necessary to use a different colour space.



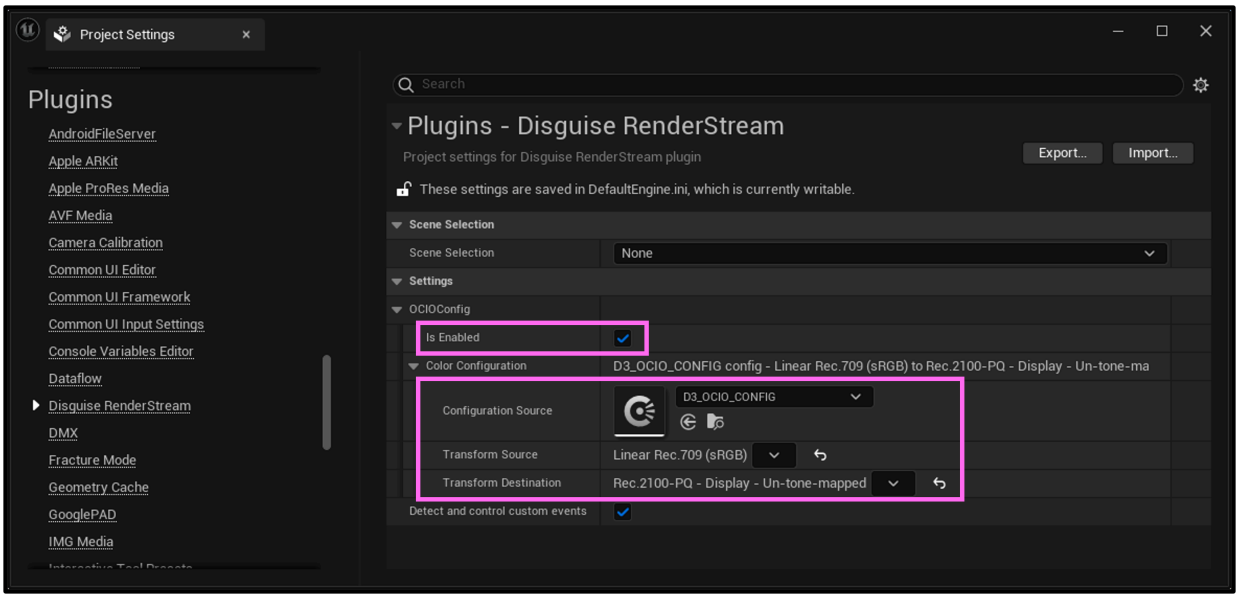
Engine Rendering Settings

> **Note**
>
> For more detailed information on colour spaces in Unreal Engine, refer to this Epic documentation - Color Pipeline.

Plugins - Disguise RenderStream

To avoid unintentional colour transforms, it's crucial to enable and configure OCIO with the desired colour spaces.

1. Navigate to **Plugins > Disguise RenderStream**.
2. Expand the **OCIOConfig** section.
3. Select the **Is Enabled** checkbox.
4. Set the **Configuration Source** to the **OpenColorIO Configuration** you created earlier. (See: Adding a New Configuration).
5. Set the **Transform Source** to match the Unreal Engine **Working Color Space**.
   - Ensure the colour space is exposed in **OpenColorIO Configuration** (See: Prepare colour spaces).
   - Refer to Transform Source for matching colour space between Unreal Engine and OCIO.
6. Set the **Transform Destination** to match the **Input transform** of the RenderStream asset in Designer.
   - The colour space must be between 0 and 1.
   - `Rec.2100-PQ - Display - Un-tone-mapped` is recommended for normal use cases.
   - See: Set Input Transform on RenderStream Layer.



Plugins - Disguise RenderStream Settings

Transform Source

UE Working Colour Space	Matching OCIO Config Colour Space
sRGB / Rec709	Linear Rec.709 (sRGB)
ACES AP1 / ACEScg	ACEScg

> **⚠ Warning**
>
> Unreal Engine provides a **UE > Working Space** option for **OpenColorIO Configuration**, but this may not produce accurate colour in RenderStream.

Transform Destination

We recommend using `Rec.2100-PQ - Display - Un-tone-mapped` as the **Transform Destination** for the following reasons:

1. **Rec.2100** closely aligns with the achievable gamut of the display.
2. PQ (Perceptual Quantizer) provides a gamma encoding that preserves details across the range. In contrast, a standard linear encoding would lose detail in the darker regions of the image. This is important because RenderStream send frames in a 10-bit signal format, rather than a 16-bit floating-point format.
3. The colour space shared between Unreal Engine and Designer is an intermediate composite stage; therefore, tone mapping is unnecessary.

Likewise, we recommend using a colour space that fits within the 0 to 1 range and with gamma curve encoding. This ensures that frames can preserve as much data as possible within the RenderStream 10-bit signal format. Note that colors outside the 0 to 1 range will be clamped.

Camera Actor Settings

Unreal Engine applies several post-processing effects which change colour grading. To maintain consistent colour through the rendering pipeline, it's important to disable any unnecessary settings affecting colour.

1. Open the **Details** section of your **CameraActor** or **CineCameraActor**.
2. Navigate to **Post Process > Lens > Bloom**
   - Set **Intensity** to `0.0`.

3. Navigate to **Post Process > Lens > Exposure**
   - Set **Metering Mode** to `Manual`.
   - Set **Exposure Compensation** to `0.0`.
   - Set **Apply Physical Camera Exposure** to `Untick`.
   - Set **Min EV100** and **Max EV100** to `1.0`
     - This will disable auto exposure of lens.
4. Navigate to **Post Process > Lens > Image Effects**
   - Set **Vignette Intensity** to `0.0`.
5. Navigate to **Post Process > Color Grading > Misc**
   - Set **Blue Correction** to `0.0`.
   - Set **Expand Gamut** to `0.0`.
   - Set **Tone Curve Amount** to `0.0`.

OCIO settings in Designer

After configuring Unreal Engine, you must set up the colour settings in Designer accordingly to produce a consistent colour.

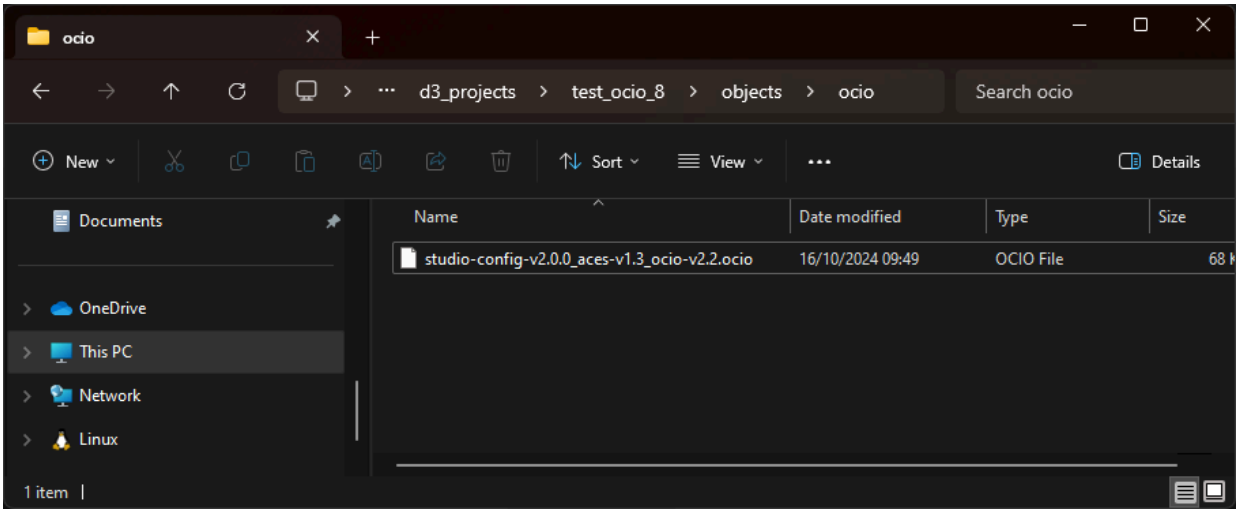
Follow the OCIO page to configure Designer, but with specific focus on matching the OCIO config file and the **Input transform** of the RenderStream.

Importing OCIO Config file into Project

It is recommended to use the same OCIO config file for both Unreal Engine and Designer.

Alternatively, ensure that the OCIO config file contains the same colour space used in the **Transform Destination** in Unreal (See: Plugins - Disguise RenderStream).

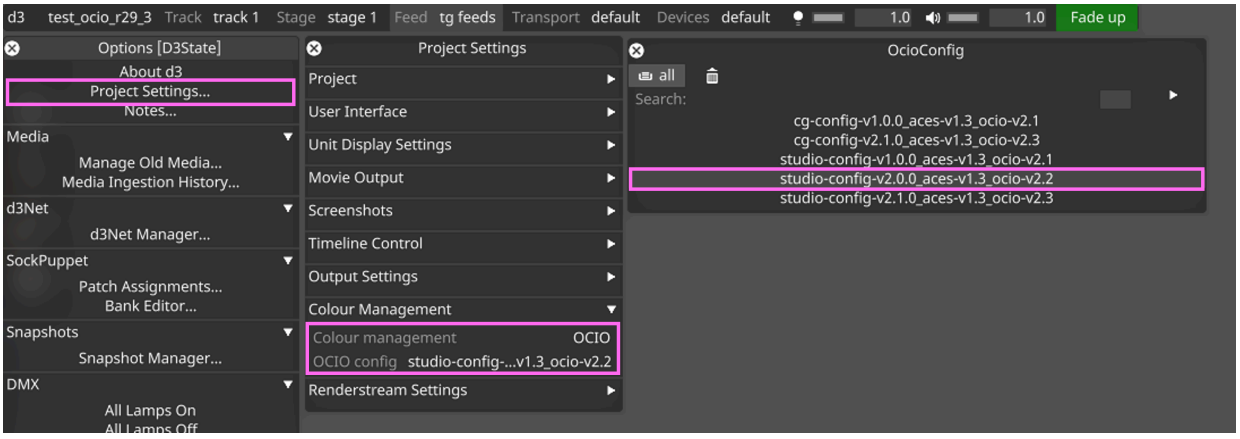
1. Copy the OCIO config file from `RenderStream Projects/<UEProject>/Content/<OcioConfig>.ocio`.
   - See: OCIO Config file.
2. Place the copied config file into the following directory:
   - `d3_projects/<PROJECT_DIR>/objects/ocio/`.



Copied OCIO Config File

Apply OCIO settings in Designer

1. Open **Project Settings...**.
2. Set **Colour management** to `OCIO`.
3. Set **OCIO config** using the OCIO config file used for the Unreal Engine project.

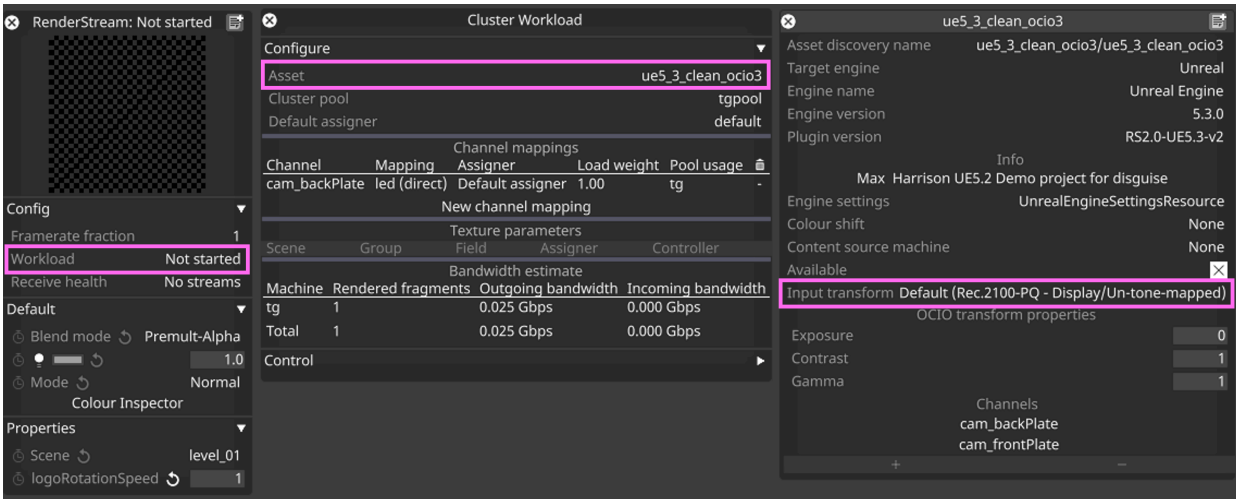


Selecting OCIO Config File

Set Input Transform on RenderStream Layer

Ensure the **Input transform** of the RenderStream layer matches the **Transform Destination** of **Unreal Engine Plugins | Disguise RenderStream**.

1. Open the RenderStream layer widget.
2. Right-click the **Asset**.
3. Set the **Input transform** to match the **Transform Destination** from Unreal.



Input Transform of RenderStream Layer

Verifying Colour with a Colour Checker

It's a good idea to scientifically validate the colour pipeline using a colour checker. This section will guide you through setting up the pipeline to validate the colour accuracy of your rendering using a colour checker.

Designer provides a **Colour Inspector** tool to inspect the pixel values of the final output, ensuring colour accuracy.

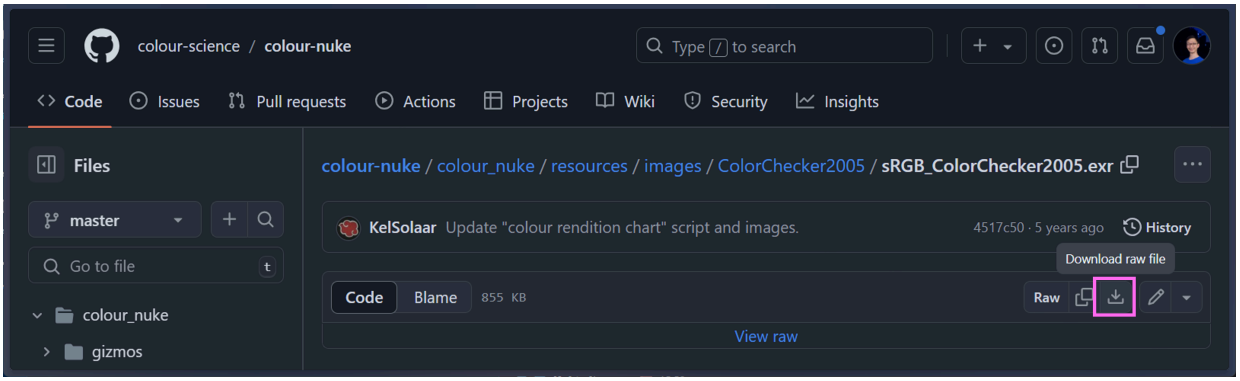
Colour Checker Setup

We'll use a colour checker from Color-Nuke GitHub. Select the `.exr` file based on your **Working Color Space** of Unreal Engine.

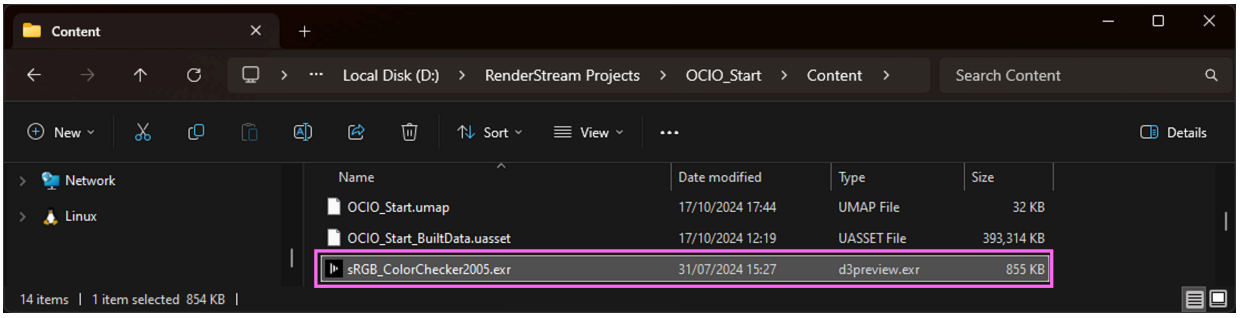
Working Color Space	Link
Rec.709	sRGB_ColorChecker2005.exr - Color-Nuke GitHub
ACES AP1 / ACEScg	ACEScg_ColorChecker2005.exr - Color-Nuke GitHub

Place the downloaded file in the Unreal Engine **Content** folder:

- `RenderStream Projects/<UEProject>/Content`



Color Nuke - sRGB Colour Checker

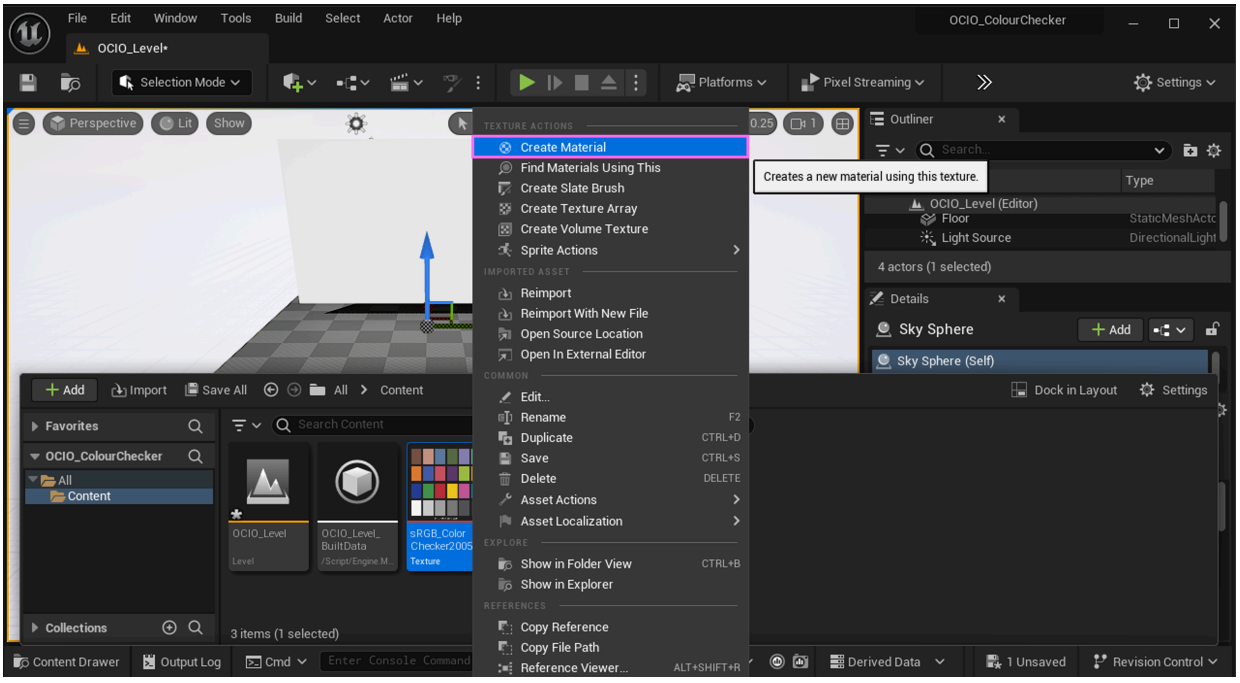


sRGB Colour Checker in Content Folder

Creating a Material in Unreal Engine

Once the colour checker is added as a texture, follow these steps to create a material:

1. Right-click the texture in the **Content Drawer**.
2. Select **Create Material**.

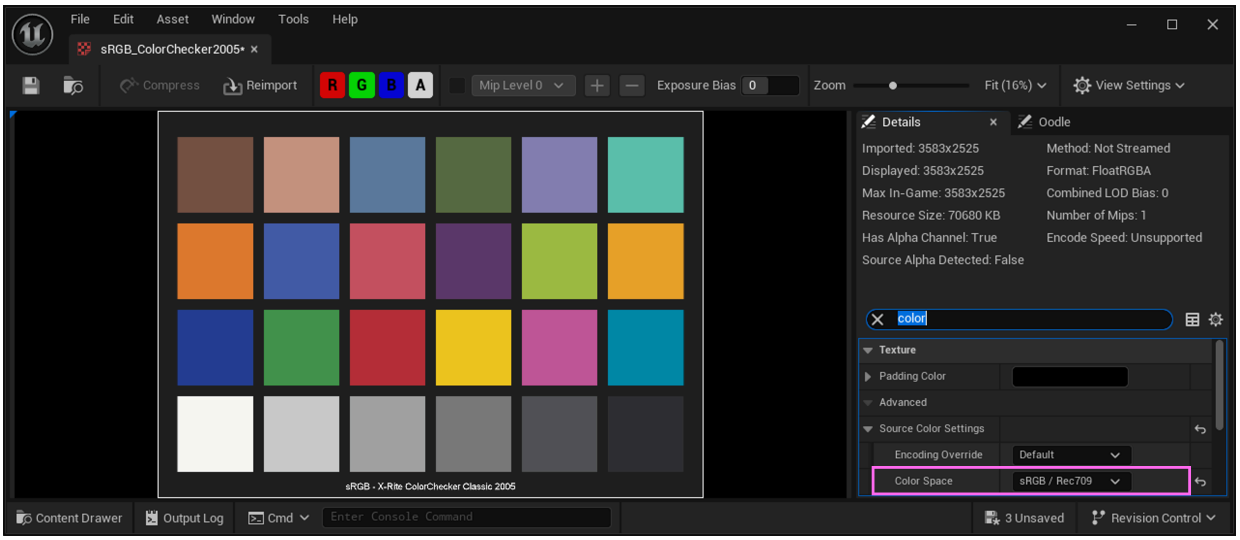


Create Material from Colour Checker Texture

Set colour space of texture (Optional)

If the colour space of the texture is different from your **Working Color Space**, adjust it:

- Double-click the texture.
- Under **Texture > Source Color Settings**, set the **Color Space** accordingly.
  - `sRGB / Rec.709` for sRGB
  - `ACES AP1 / ACEScg` for ACEScg

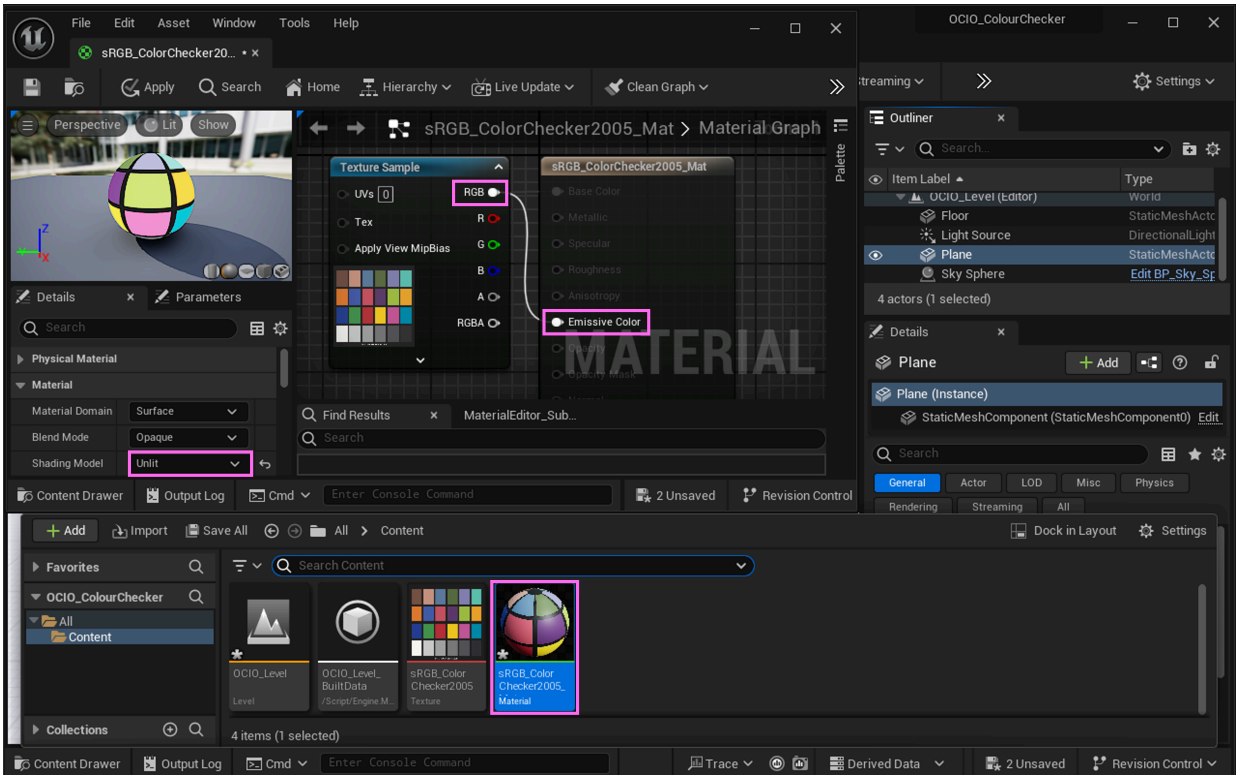


Color Space of Colour Checker Texture

Configure the Material

To avoid lighting effects impacting the colour checker, you need to configure the material to be **Unlit**:

1. Double click the material created from the colour checker texture.
2. Set the **Shading Model** to `Unlit`.
3. Connect the **RGB** node of the **Texture Sample** to the **Emissive Color** input.

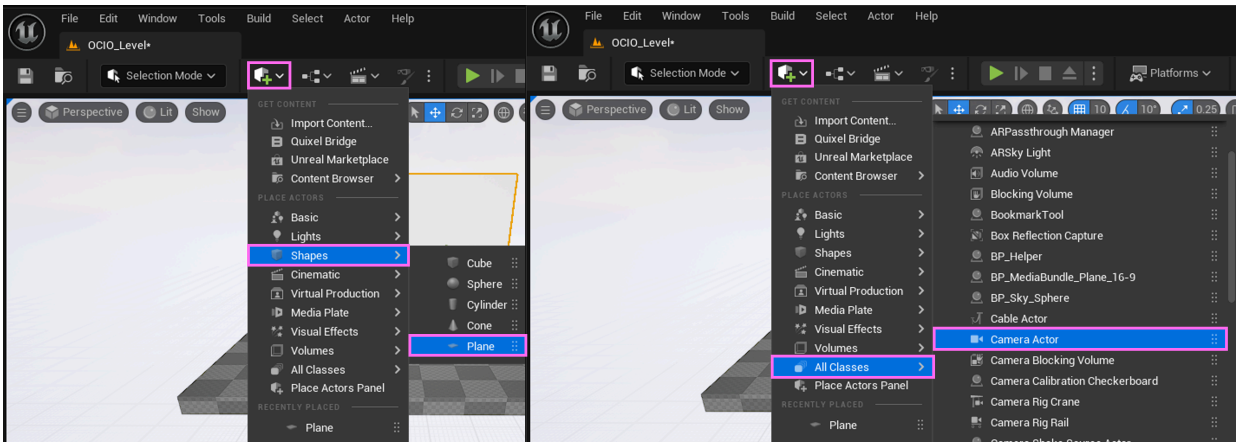


Colour Checker Material Settings

Viewport for Colour Checker

To create a viewport fully filled with the colour checker, it is recommended to add a dedicated **Plane** and **CameraActor** pair. While it is possible to use an existing **CameraActor** and **Plane**, using a dedicated pair ensures better control over the colour checker output.

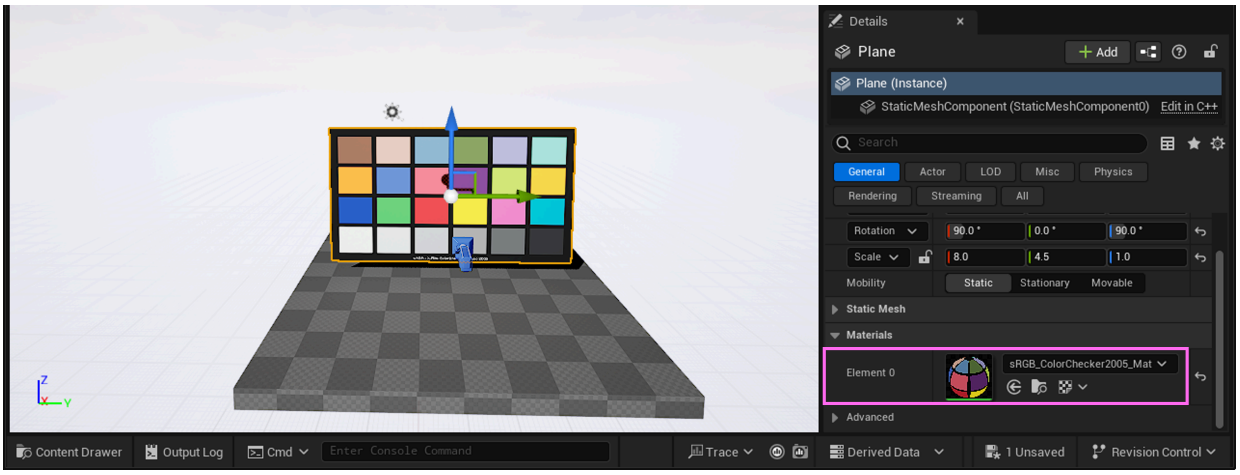
1. Add a **Plane** and a **CameraActor** (Optional).



Plane and Camera Actor

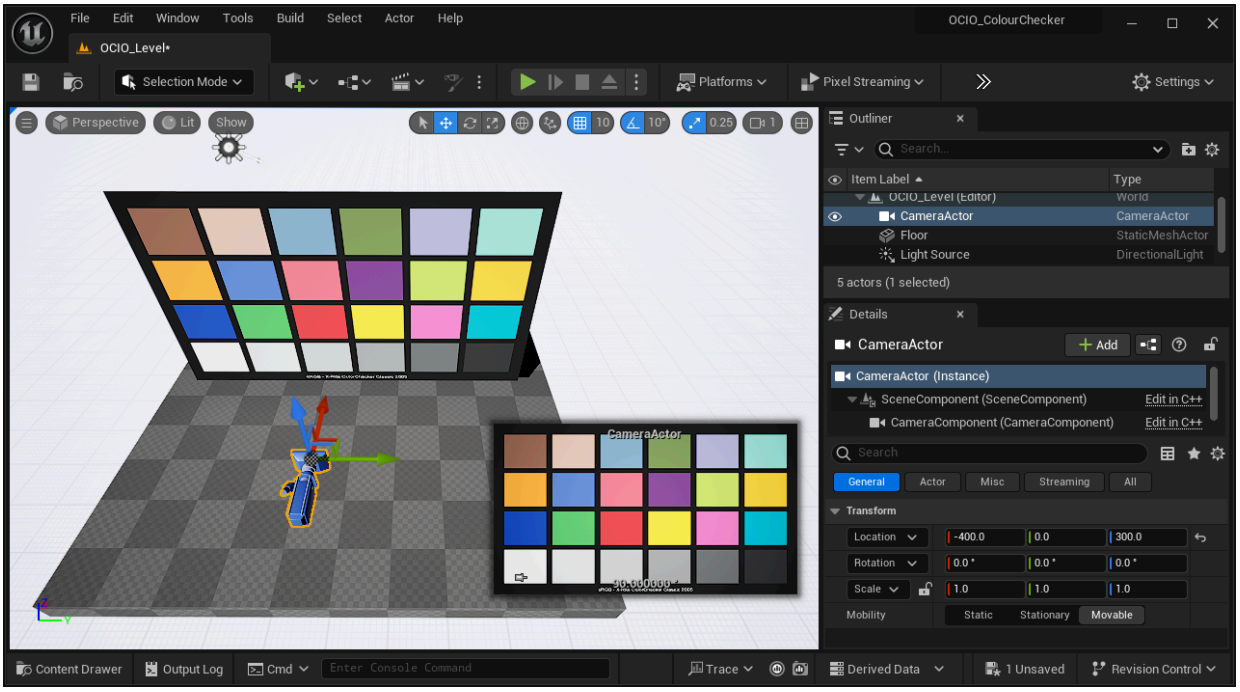
2. Adjust the **Transform** of both actors.
   - See: Recommended Transform settings.
3. Select the **Plane** and open the **Details** panel.
4. Expand the **Materials** section.

5. Under **Element 0**, assign the created colour checker material to the **Plane**.



Setup Plane Material

6. Select the **CameraActor**.
7. Verify that the viewport is fully filled with the colour checker.



CameraActor Result Viewport

> **Warning**
>
> Ensure you have followed the instructions in Camera Actor Settings to prevent any unintended post-processing effects.

Recommended Transform to achieve the colour checker viewport

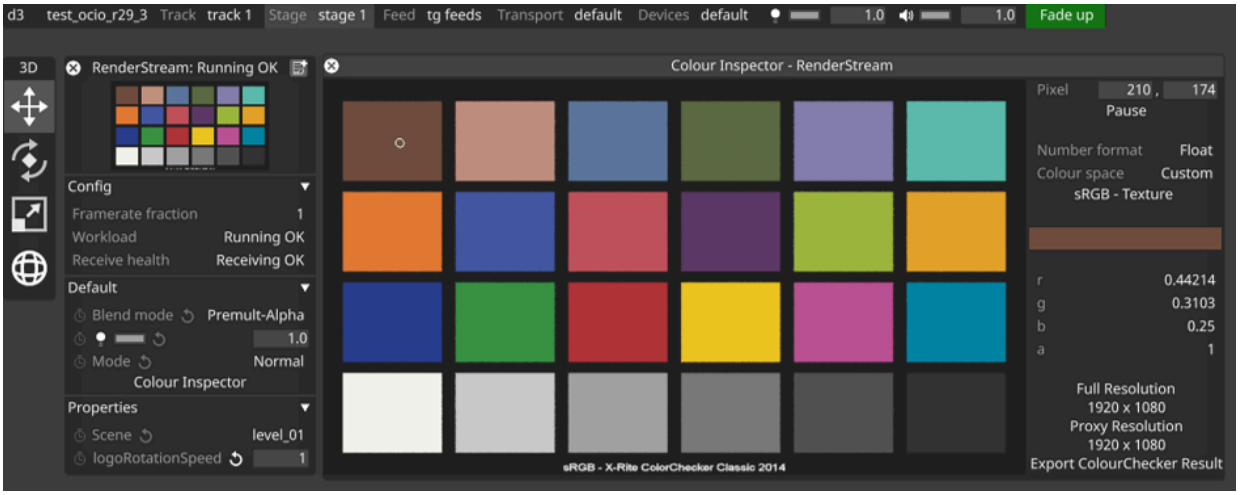
Actor	Location	Rotation	Scale
Plane	`0.0`, `0.0`, `300.0`	`90.0`, `0.0`, `90.0`	`8.0`, `4.5`, `1.0`
CameraActor	`-400.0`, `0.0`, `300.0`	`0.0`, `0.0`, `0.0`	`1.0`, `1.0`, `1.0`

Designer Colour Inspector

> **Note**
>
> Before proceeding, ensure that Designer RenderStream has been properly configured with OCIO (See: OCIO settings in Designer).

The Designer **Colour Inspector** tool allows you to inspect colour values at different stages of rendering. Since we are working with a RenderStream layer, follow these steps to use the **Colour Inspector** for the RenderStream layer:

1. Start the RenderStream workload.
2. Open the RenderStream layer widget.
3. Expand the **Default**.
4. Select **Colour Inspector**.
5. Change the **Colour Space** to `Custom`.
6. Set the **Colour Space** to `sRGB - Texture`:
   - Use `sRGB - Texture` for any type of colour checker (i.e. `ACEScg`).
7. Click on any pixel within the colour checker texture to view its value.
   - You can adjust the **Number Format** if needed.



Colour Inspector

8. Compare the pixel values with the **Manufacturer's sRGB color values** from Wikipedia:
   - https://en.wikipedia.org/wiki/ColorChecker

Natural Colours

Colour Format	Dark skin	Light skin	Blue sky	Foliage	Blue flower	Bluish green
Hex	#735244	#c29682	#627a9d	#576c43	#8580b1	#67bdaa
RGB Floats 0-1	0.450980 0.321569 0.266667	0.760784 0.588235 0.509804	0.384314 0.478431 0.615686	0.341176 0.423529 0.262745	0.521569 0.501961 0.694118	0.403922 0.741176 0.666667

Miscellaneous Colours

Colour Format	Orange	Purplish blue	Moderate red	Purple	Yellow green	Orange yellow
Hex	#d67e2c	#505ba6	#c15a63	#5e3c6c	#9dbc40	#e0a32e
RGB Floats 0-1	0.839216 0.494118 0.172549	0.313725 0.356863 0.650980	0.756863 0.352941 0.388235	0.368627 0.235294 0.423529	0.615686 0.737255 0.250980	0.878431 0.639216 0.180392

Primary and Secondary Colours

Colour Format	Blue	Green	Red	Yellow	Magenta	Cyan
Hex	#383d96	#469449	#af363c	#e7c71f	#bb5695	#0885a1
RGB Floats 0-1	0.219608 0.239216 0.588235	0.274510 0.580392 0.286275	0.686275 0.211765 0.235294	0.905882 0.780392 0.121569	0.733333 0.337255 0.584314	0.031373 0.521569 0.631373

Greyscale Colours

Colour Format	White	Neutral 8	Neutral 6.5	Neutral 5	Neutral 3.5	Black
Hex	#f3f3f3	#c8c8c8	#a0a0a0	#7a7a7a	#555555	#343434
RGB Floats 0-1	0.952941 0.952941 0.952941	0.784314 0.784314 0.784314	0.627451 0.627451 0.627451	0.478431 0.478431 0.478431	0.333333 0.333333 0.333333	0.203922 0.203922 0.203922

Scene Levels and Maps

This topic covers how levels and maps can be configured within an Unreal Engine scene for control via RenderStream.

The Unreal Engine plugin allows you to generate scenes from either Levels or Maps within a project. This functions similarly to how the 'Effects' parameter in Notch blocks work.

Scene selector

Levels/Maps in Unreal Engine (UE) can be composed of any number of actors. There are three options for deciding how scenes will be generated from your Unreal Engine project:

None - this offers no scene control and simply merges all Levels of your default Map into a single persistent level.

Streaming Levels - this option generates scenes from all Levels/Sub-levels within your default Map. The Actors within a Level other than the persistent level are unique to that level. The Actors within the persistent level are shared across all levels. Parameters exposed in levels other than the persistent level will only be available when the corresponding scene is chosen in the RenderStream layer. Parameters exposed in the persistent level will be available across all scenes in the RenderStream Layer.

Maps - this option generates scenes from all Maps within your project. Maps will only display the Actors and exposed parameters present within a sublevel and not include those in any parent levels.

Scene Selector	Persistent Level	Sublevel
None	Actors and parameters from all levels	X
Streaming Levels	Persistent level's actors and parameters	Persistent level's + Sublevel's actors and parameters
Maps	Persistent level's actors and parameters	Sublevel's actors and parameters only

Enable the RenderStream plug and set the Scene Selector

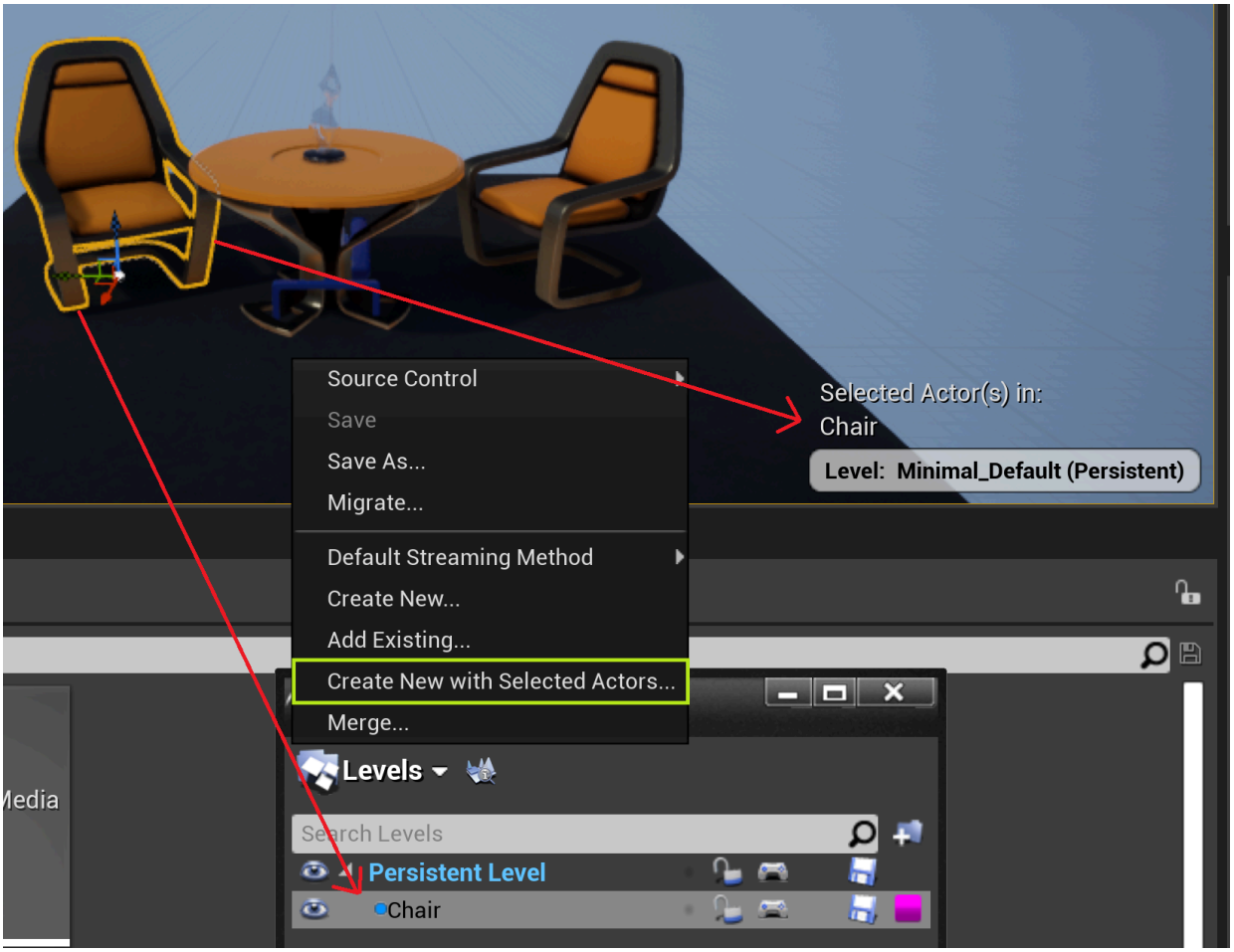
1. From the menu, select **Edit** > **Project Settings...**.
2. Navigate to Plugins/ Disguise RenderStream in the left panel.
3. Set Scene Selector as required. The options are **None**, **StreamingLevels**, or **Maps**.

Create a sub-level

1. Open the scene in the Unreal Engine Editor.
2. Select any Actor from the Outliner panel (e.g. Chair).
3. Select **Window > Levels** from the scene window.
4. Click the Levels dropdown and select **Create New with Selected Actors**.
5. Name the level and click **Save**.
6. Rebuild lighting if required (Unreal Engine may display errors at this point but it will recover).
7. Save and close Unreal Engine.

Run the workload in Designer

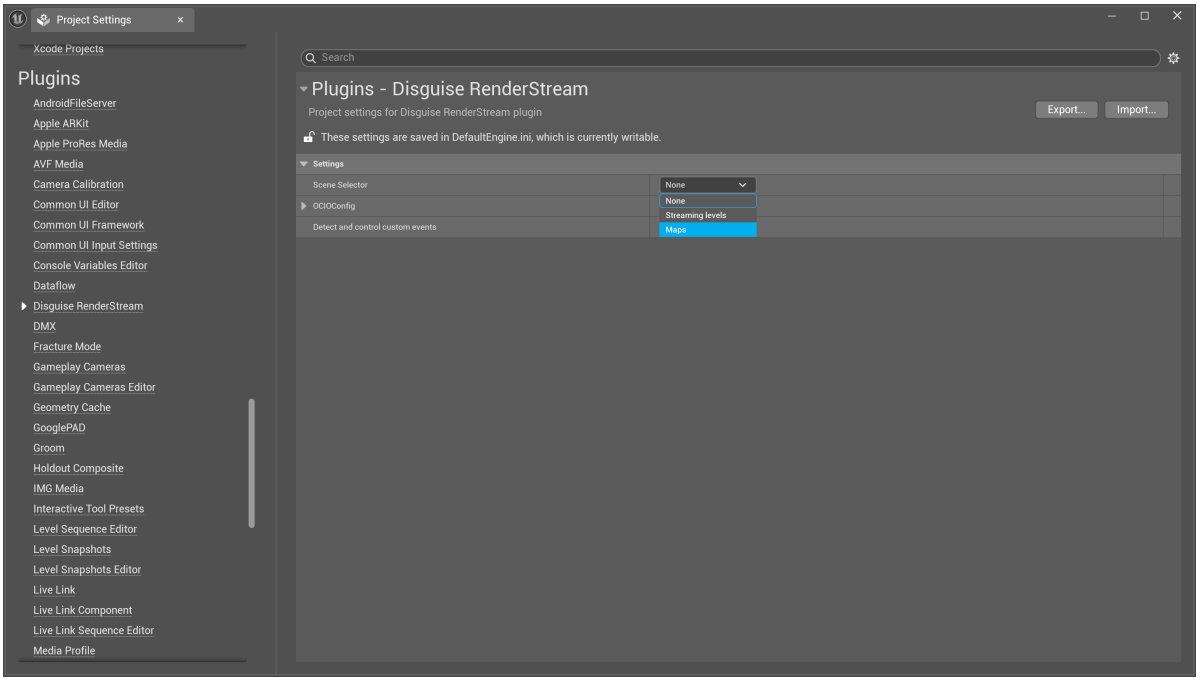
1. Open the RenderStream Layer in Disguise Designer and Start the workload.
2. Modify the Scene parameter.



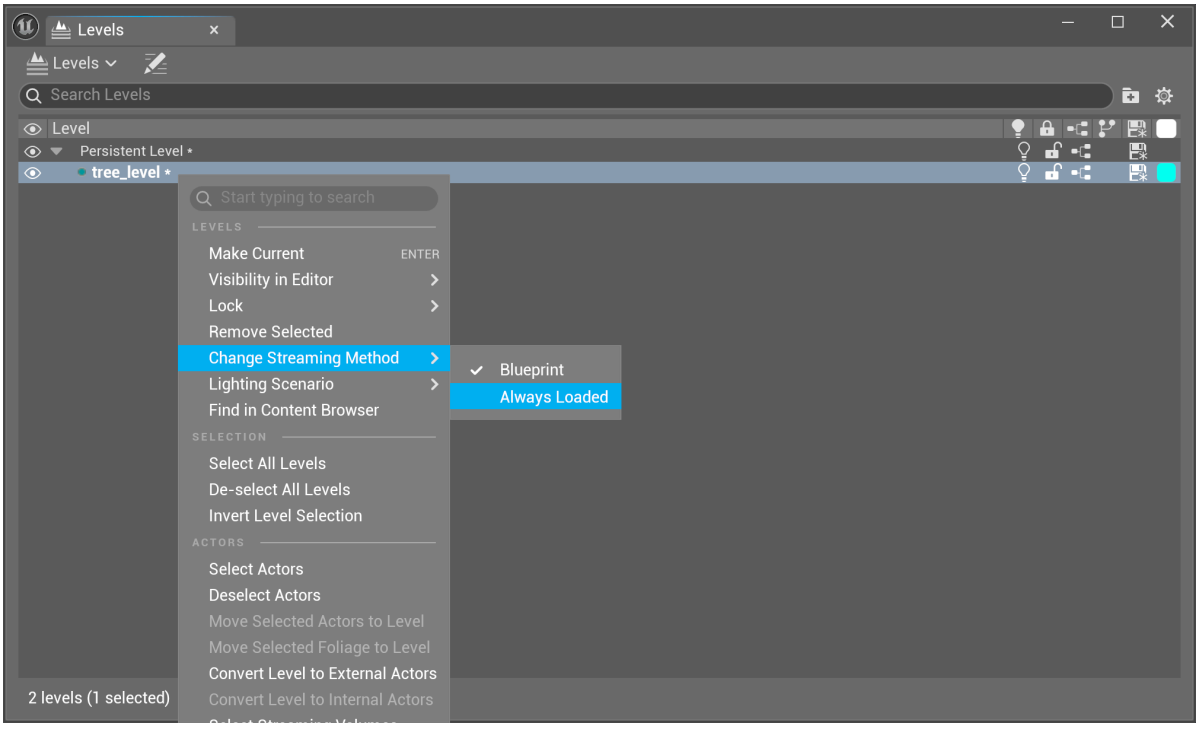
Use Console Variables (CVAR) to control Sublevel visibility

This workflow allows users to control the visibility of Levels in Multi-Level and Multi-Sublevel Unreal projects while using Multi-User Editor.

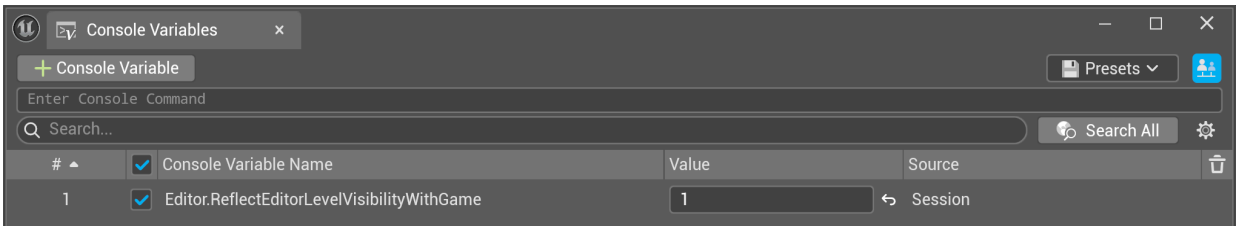
1. Ensure the Scene Selector is set to **Maps**.



2. Right-click the selected Sublevel/s in the Levels panel, and select **Change Streaming Method > Always Loaded**.



3. Start the Multi-User session in the Editor, then launch the workload.
4. Set the CVAR to 1 (Default value is 0) in the Console Variables window (or in Level Blueprint).



Notes

- When using the Maps Scene Selector option, all maps within your Unreal Engine project must be loaded in the Editor at least so that they are cached. If they are not loaded at least once, they will not appear within the scene selection.
- When using the Maps Scene Selector option, sublevels are handled differently compared to when the Streaming Levels option is selected.
  - Maps will only display the Actors present within a sublevel and not include those in any parent levels.
  - Streaming Levels will display the Actors present within a sublevel as well as those included in any parent levels.
- When using the None Scene Selector option, the scene Unreal will present when a workload is started will always be the Game Default Map set within the project settings (unless otherwise specified using the Custom Options field).

Exposed Parameters in Unreal Engine

This topic explains how to expose parameters in an Unreal Engine scene and enable those parameters to be controlled using the RenderStream Layer in Designer.

The Unreal Engine RenderStream plugin allows you to expose variables within the UE scene that can be used to control the parameters of the actors in the virtual environment. These variables are presented as custom RenderStream Layer Editor parameters in Designer. Modifying the value of a parameter changes the value of the corresponding variable in the Unreal Engine level thus altering the parameter/s of the selected actor or setting within the scene.

Prerequisites

- Unreal Engine Scene
- RenderStream plugin enabled in the UE project
- Designer project

View the RenderStream Unreal Engine Integration Project Setup page here.

Exposing Parameters in an Unreal Engine Project Scene

There are two methods for exposing parameters in an Unreal Engine scene.

1. Using an Event Tick.
2. Using a Custom Event.

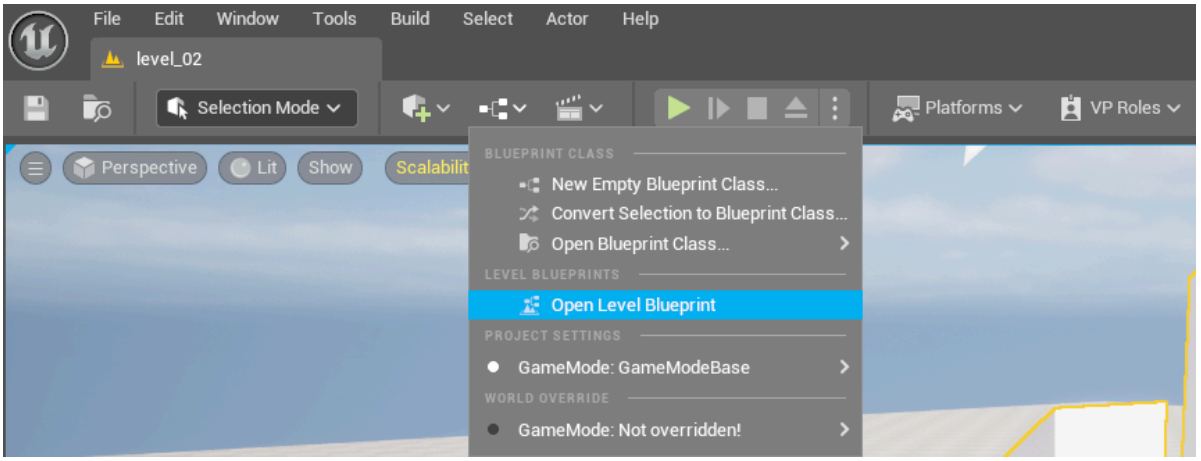
Both options can be added to a Level Blueprint to trigger an event and handle the change when this event is triggered.

Ticking is used to run a code or Blueprint script on an actor or component at regular intervals. If an Event Tick is used, an Event is called every frame. This is a particularly inefficient method of exposing parameters, but is the more widely known method for the workflow.

Custom Events in Unreal Engine are user-defined functions that allow you to trigger specific actions or logic within your Blueprints. These are reusable events that can be called whenever needed. This helps streamline complex interactions and maintain clean, manageable Blueprint scripts.

Creating and configuring a variable in the Level Blueprint

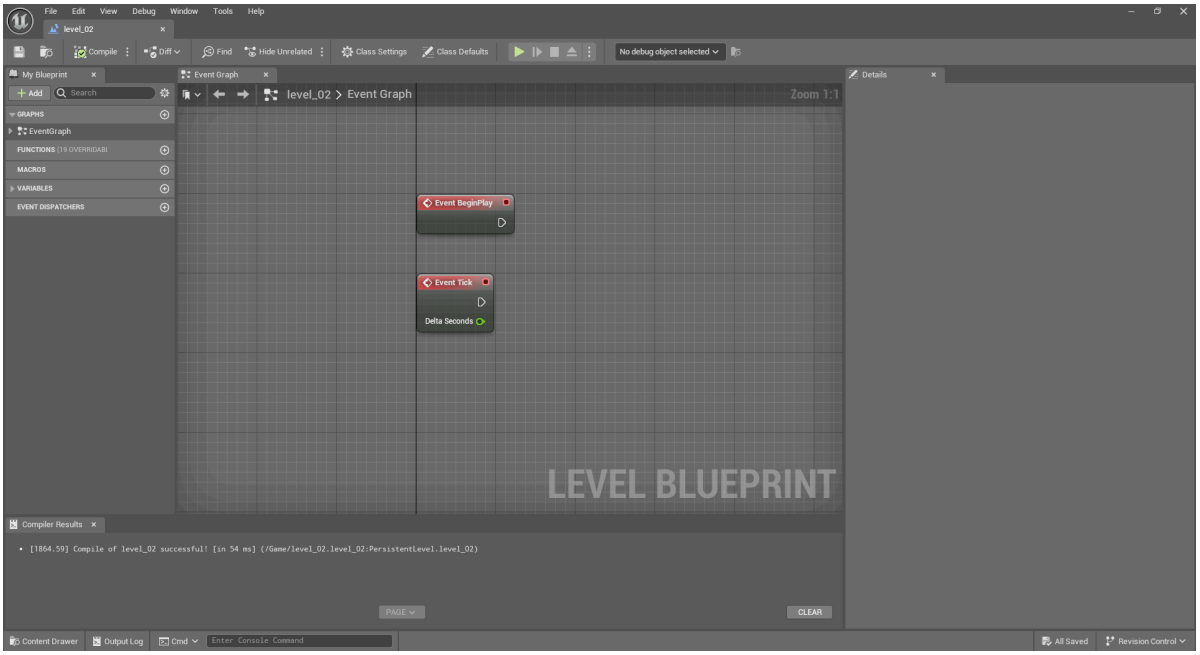
1. Click Blueprint in the Toolbar and select **Open Level Blueprint**.



2. Click the **Add** button next to Variables within the My Blueprint panel.
3. Name the variable according to the desired property (e.g. Fog Visibility).
4. Select the variable.
5. Set the **Variable Type** in the **Details** panel to an appropriate property (e.g. `visibility == boolean`).
6. Click on the check box to enable **Instance Editable**. (Alternatively click on the eye symbol.)
7. Set the **Category** to represent the parameter in Designer. This will appear as a section in the RenderStream Layer Editor.
8. When using a sliding range, configure the ranges for the value.
9. Click **Compile** and then click **Save**.
10. Next, set the Default Value for the variable in the Details panel.

1. Assigning an action to the variable in the Level Blueprint using an Event Tick

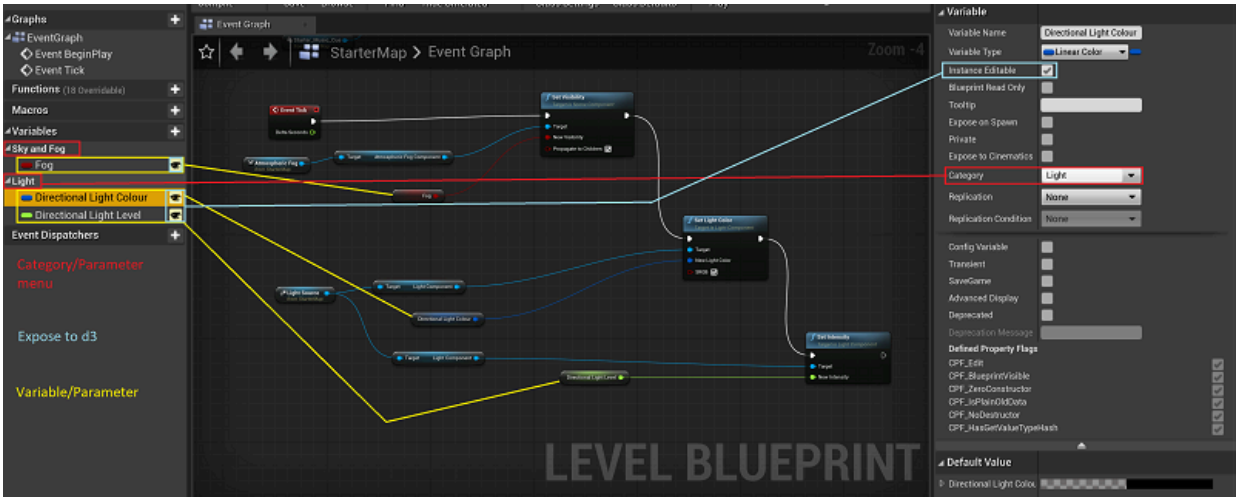
1. In the Level Blueprint, right-click anywhere in the Event Graph and create an **Event Tick** action. Depending on the project you are using, one may already appear disabled and ready for use.



2. Return to the scene.
3. Select an Actor from the Outliner panel (e.g. Atmospheric Fog).
4. Drag and drop the Actor into the Level Blueprint's Event Graph.
5. Drag out from the blue pin (Static Mesh Actor Object Reference) on the Actor component and create any new Rendering Component Action (e.g. Set Visibility or Set Intensity).
6. Drag out from the white pin (Exec) on the Event Tick action and connect it to the white pin (Exec) on the left-side of the Rendering Component Action.
7. Drag out from the **New Component_Name** pin (e.g. New Visibility) on the Rendering Component Action and create a **Get Variable_Name** action.
8. Compile and Save.

Modifying the parameter in Designer

1. Open the RenderStream Layer Editor in Designer and start the Workload.
2. Modify new parameter value/s in the layer editor as part of your normal sequencing.



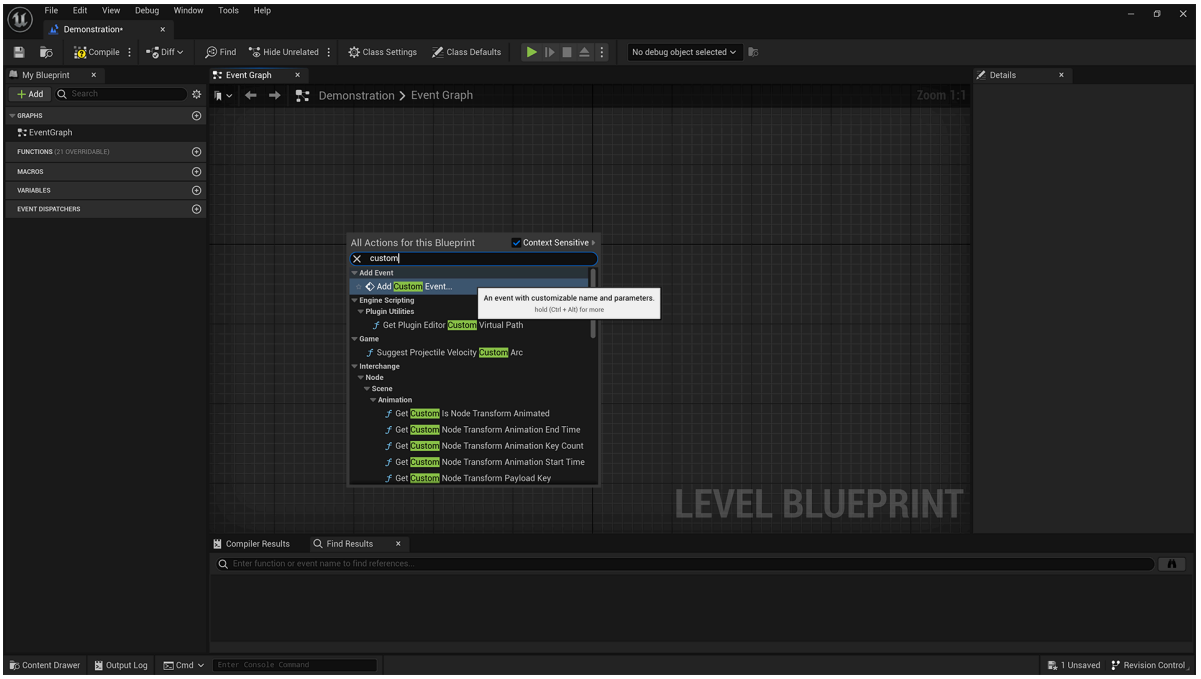
Alternatives to Event Ticks

Exposed Parameters attached to an **Event Tick** are expensive because they will be triggered every tick and re-apply the exposed parameter. To improve performance, Designer has 3 other options that will give you better performance:

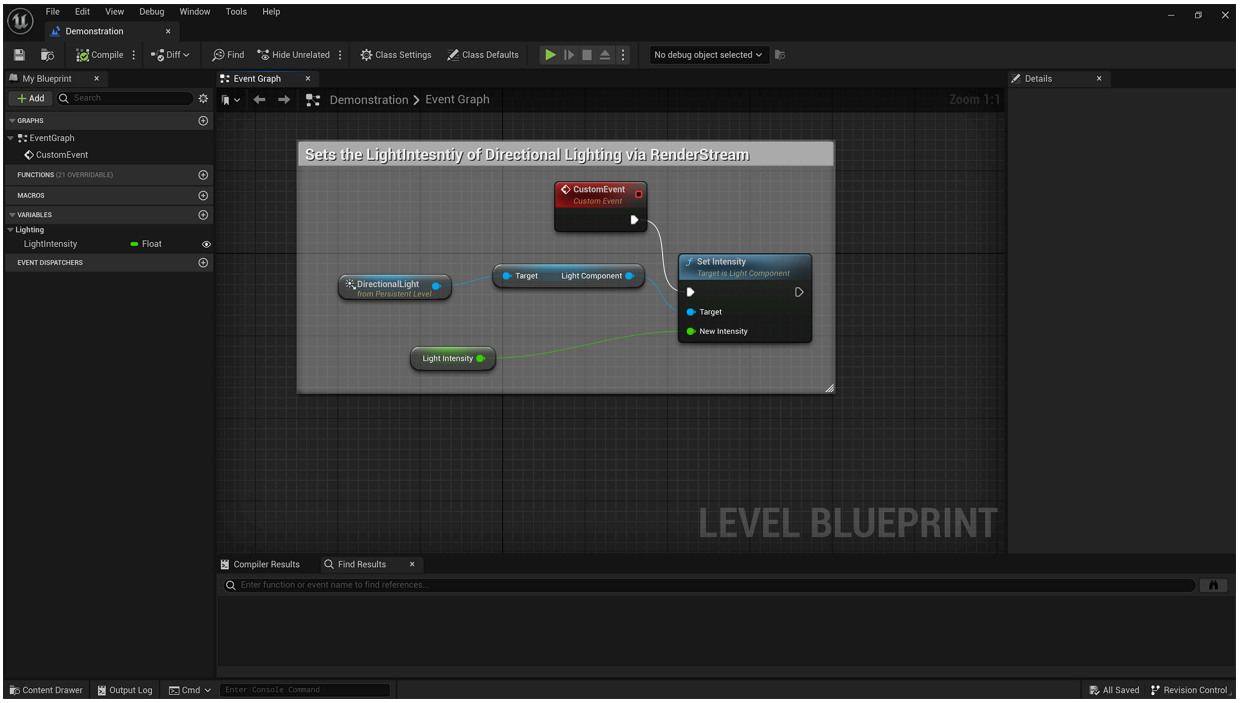
- **On Keyframe** - Will trigger the event only when a keyframe is encountered in Designer.
- **On Change** - Will trigger the event only when the parameter changes.
- **On Reset** - Will trigger the event only when the parameter is set to `0`.

2. Assigning an action to the variable in the Level Blueprint using a Custom Event

1. In Unreal Engine, right-click in the Blueprint's Event Graph, type **Custom Event** to search for the action, and click **Add Custom Event**. Then, name the event.**Custom Event** action. Where you have existing nodes such as EventPlay and Event Tick, these can be removed.



2. Name the Custom Event to reflect the parameter that has been exposed.
3. Return to the scene.
4. Select an Actor from the **Outliner** panel (e.g. Atmospheric Fog or Directional Lighting).
5. Drag and drop the Actor into the Level Blueprint's Event Graph.
6. Drag out from the blue pin (Static Mesh Actor Object Reference) on the Actor component and create any new Rendering Component Action (e.g. Set Visibility).
7. Drag out from the white pin (Exec) on the Event Tick action and connect it to the white pin (Exec) on the left-side of the Rendering Component Action.
8. Drag out from the **New Component_Name** pin (e.g. New Visibility) on the Rendering Component Action and create a **Get Variable_Name** action.
9. Compile and Save.



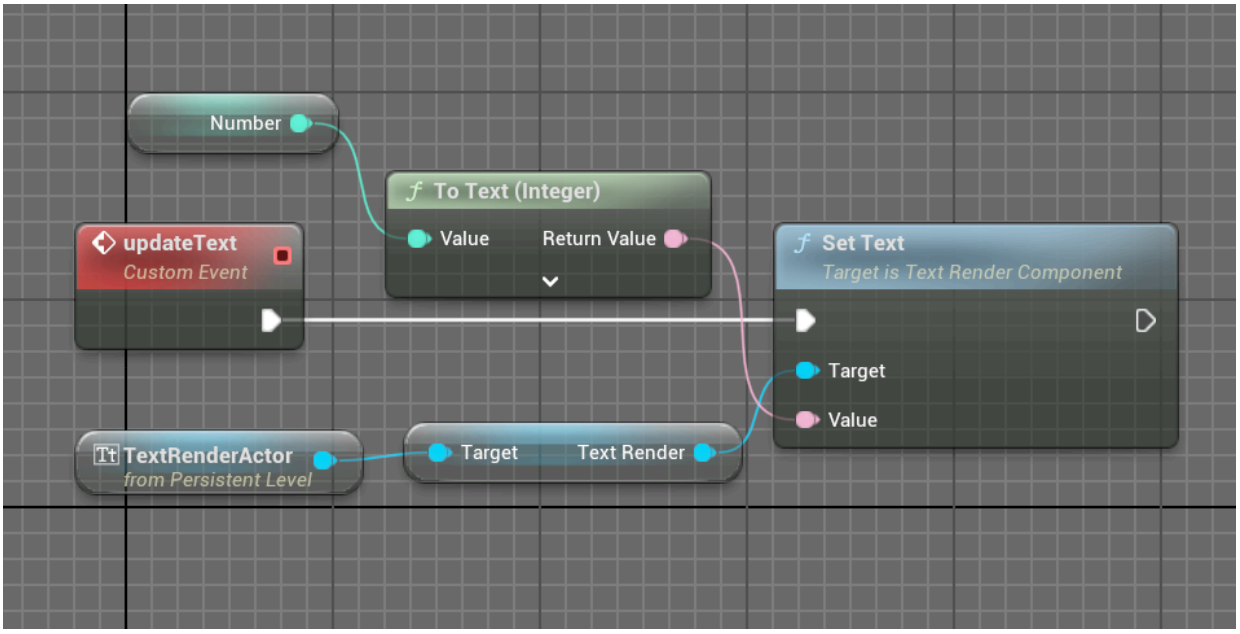
Using Triggers generated by Custom Events in Designer

1. Open the **RenderStream Layer Editor**.
2. Under **Custom Events** you fill find a field called **Trigger Mode**.
3. Select the appropriate **Trigger Mode**.
4. Modify custom event value/s in Disguise as part of your normal sequencing to trigger the custom event when required.

Unreal engine also provides workflows for controlling parameters using the Remote Control API, or their MultiEditor plugins and features. These can be configured outside of Designer entirely if none of the above options are suitable. If you are using Porta, for instance, this is a standard workflow.

Examples

The example Unreal Level Blueprint below demonstrates triggering Custom Events in Unreal from Designer.



Level Blueprint: Creates a TextRenderActor with a number value that can be updated

On Keyframe

This will only trigger the Custom Event in Unreal when a keyframe is passed on the timeline in Designer.

Designer: On Keyframe, the exposed number value changes with each keyframe

On Change

This will trigger the Custom Event in Unreal when the number value in the exposed Custom Event changes. The number has been keyframed to change over time to have it trigger

constantly.

Designer: On Change, the exposed number changes with each change

3D object transforms

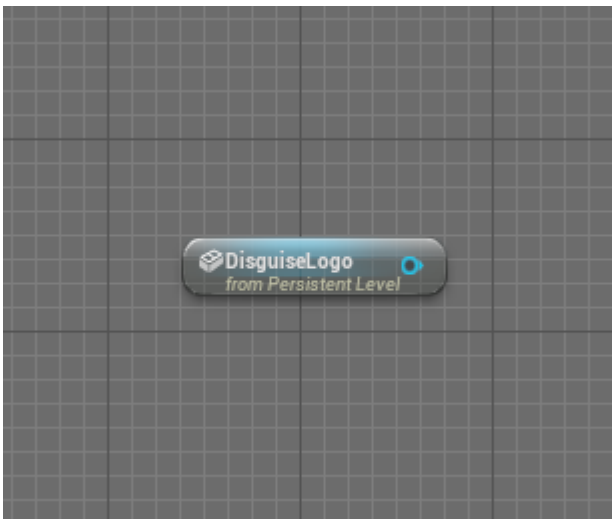
This topic covers the steps needed for configuring 3D object transforms within an Unreal Engine project for use with RenderStream. This feature allows you to control the full range of 3D object transforms (translation, rotation and scale) of an object in from using a null object.

This feature allows you to control the full range of 3D object transforms (translation, rotation and scale) of an object in Unreal Engine from Designer using a null object.

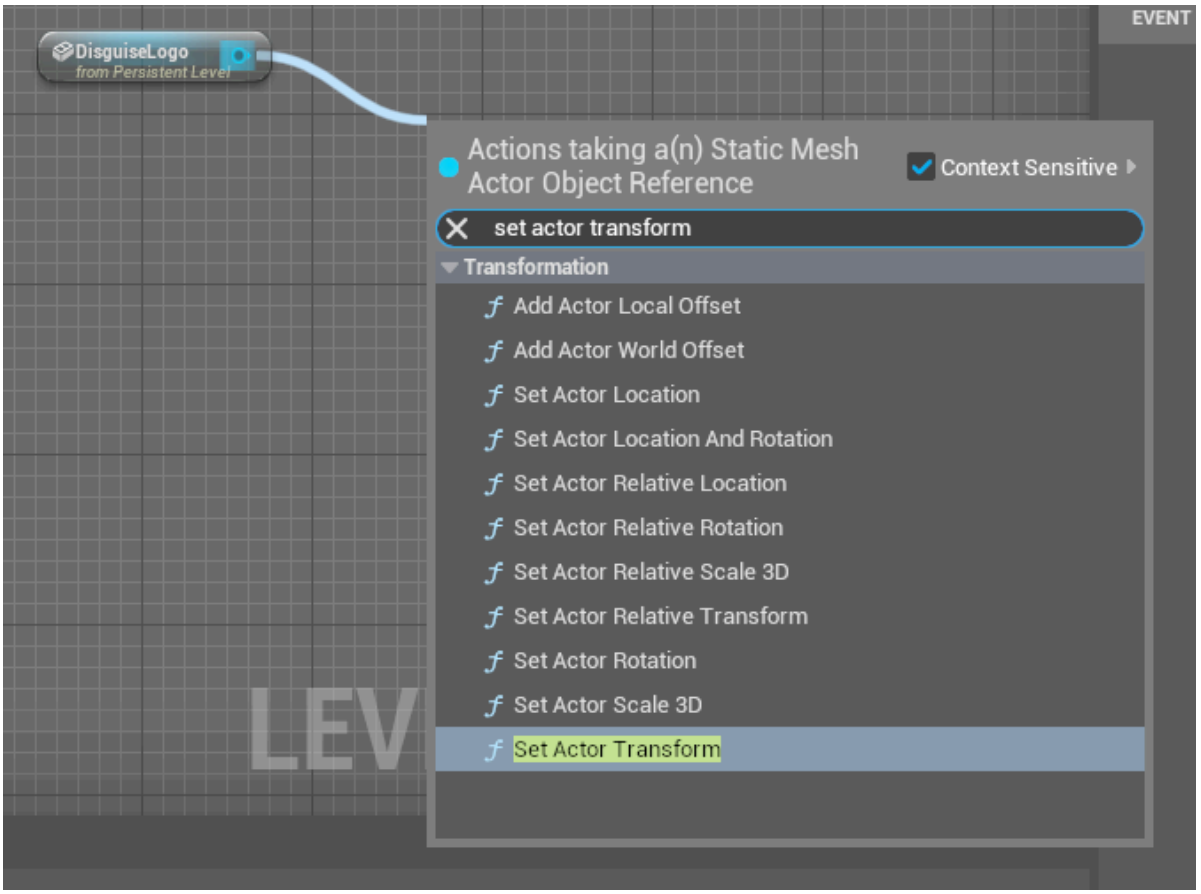
If you wish to expose the Transform (i.e. position, rotation and scale) of an Actor, you must ensure that the Actor's Mobility is set to 'Movable' from within the Details panel. If this is not done, Designer will only be able to control the Actor's scale. Follow these steps to configure the 3D object transforms of an object within an Unreal Engine sce

Unreal Engine Level Blueprint setup

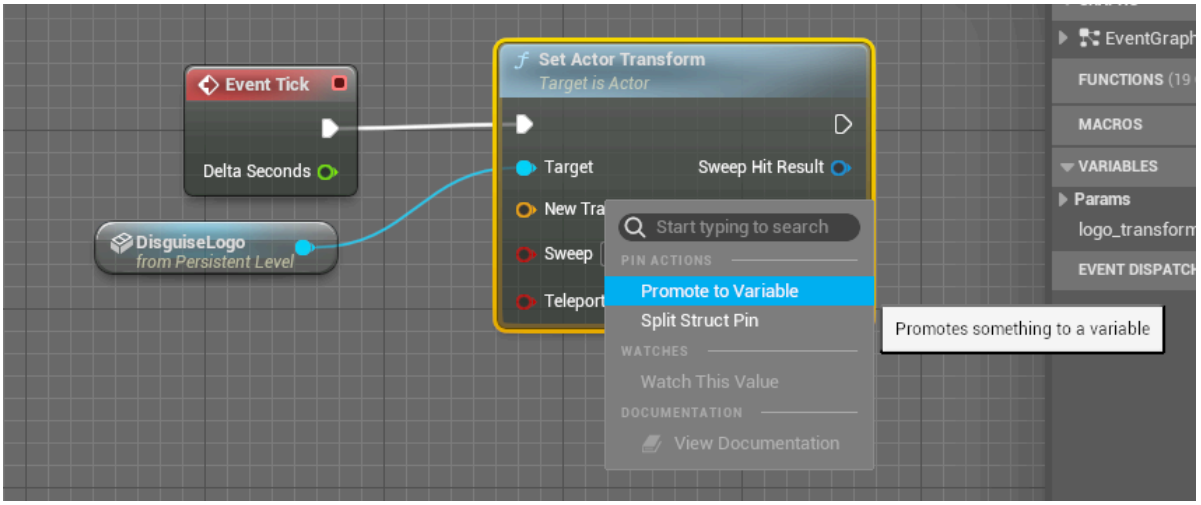
1. To control the transforms of an Actor (eg. Cine Camera Actor, Static Mesh Actor) from within Designer, start by dragging the Actor into the Level Blueprint.



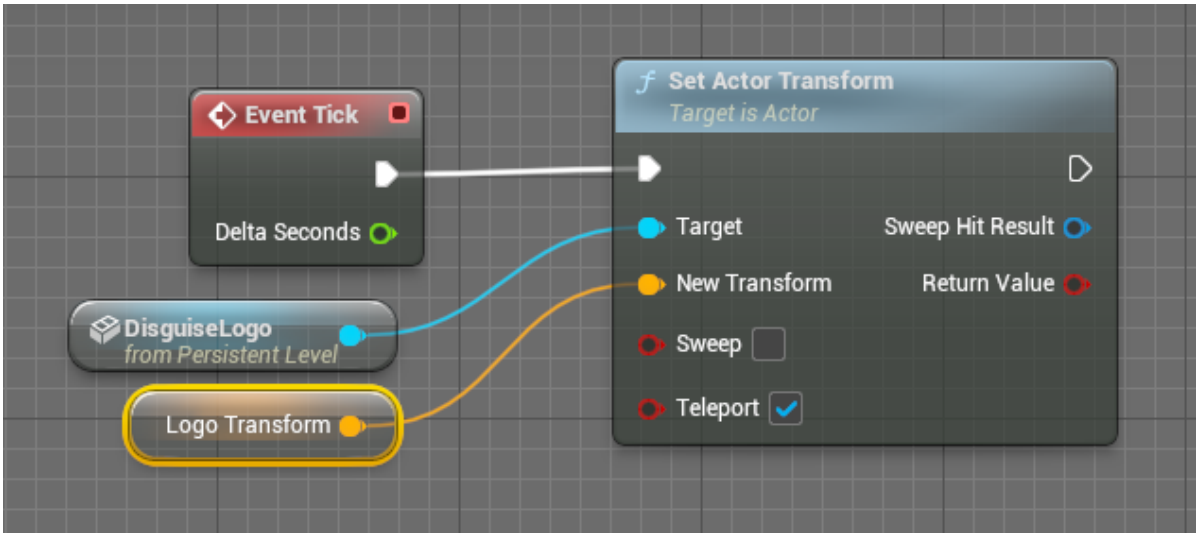
2. Drag the blue pin from the a new node in the Event Graph. When prompted to search for a new action, search for “Set Actor Transform”.



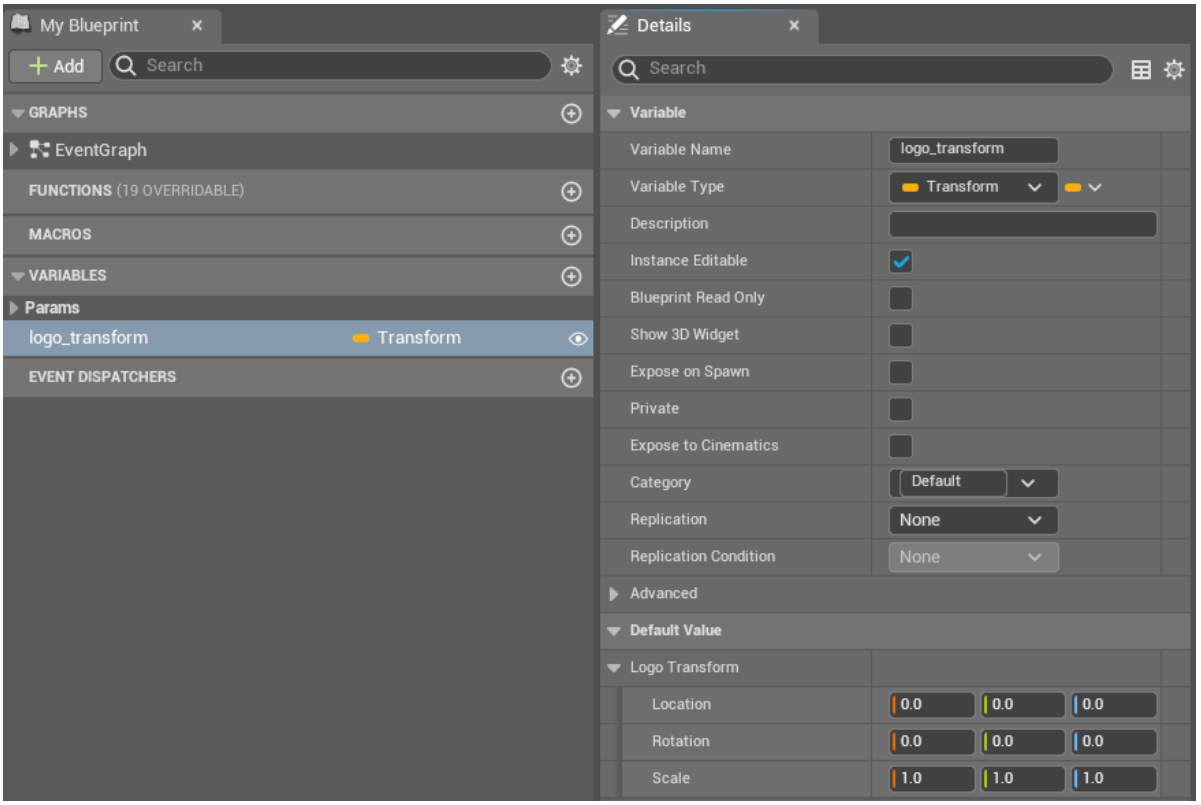
3. Right-click on the “New Transform” and select the option to “Promote to Variable”.



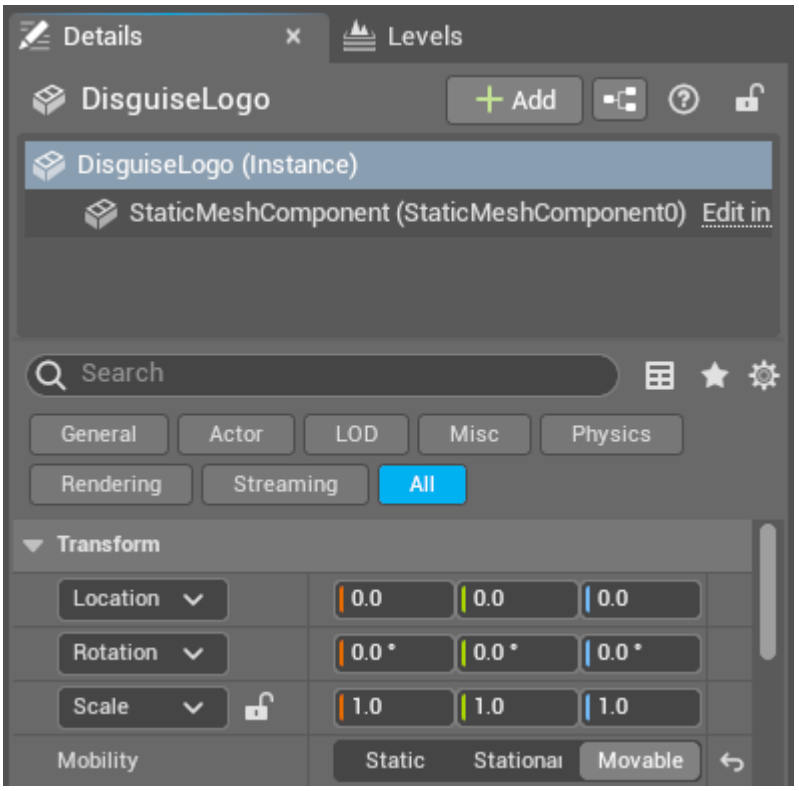
4. The next step is to link this to the “Event Tick”.



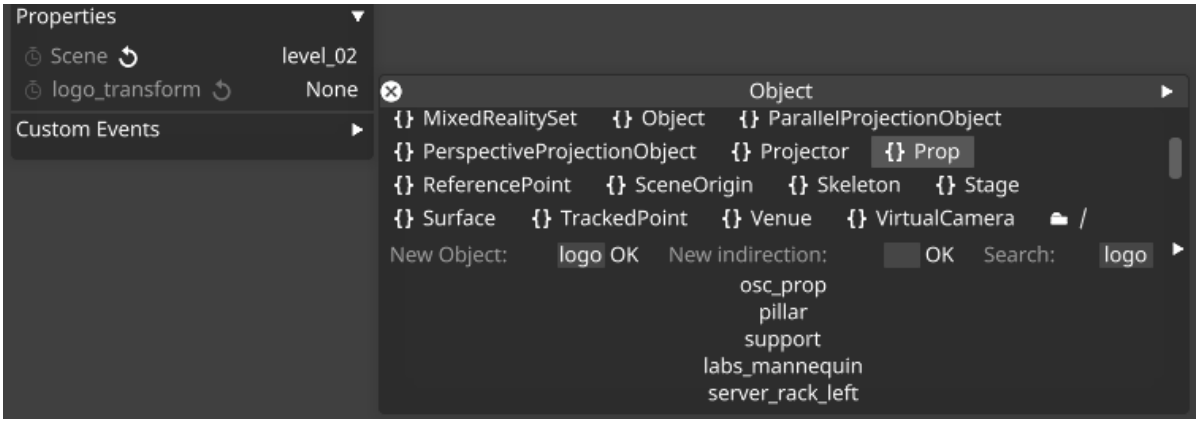
5. Now Compile the Blueprint. This will give you the ability to adjust the transforms of that object. Be sure to set “New Transform” to **Instance Editable**.



6. The last step is to make the object **Movable**. In the default scene all the objects will be set to static; this can be changed in the Object details panel.



7. Save the project.
8. You can now control the Actor from within by selecting a prop to link its transforms to from within the RenderStream Layer Properties tab.



Remote Texture Sharing

Textures can be shared between Designer and Unreal Engine.

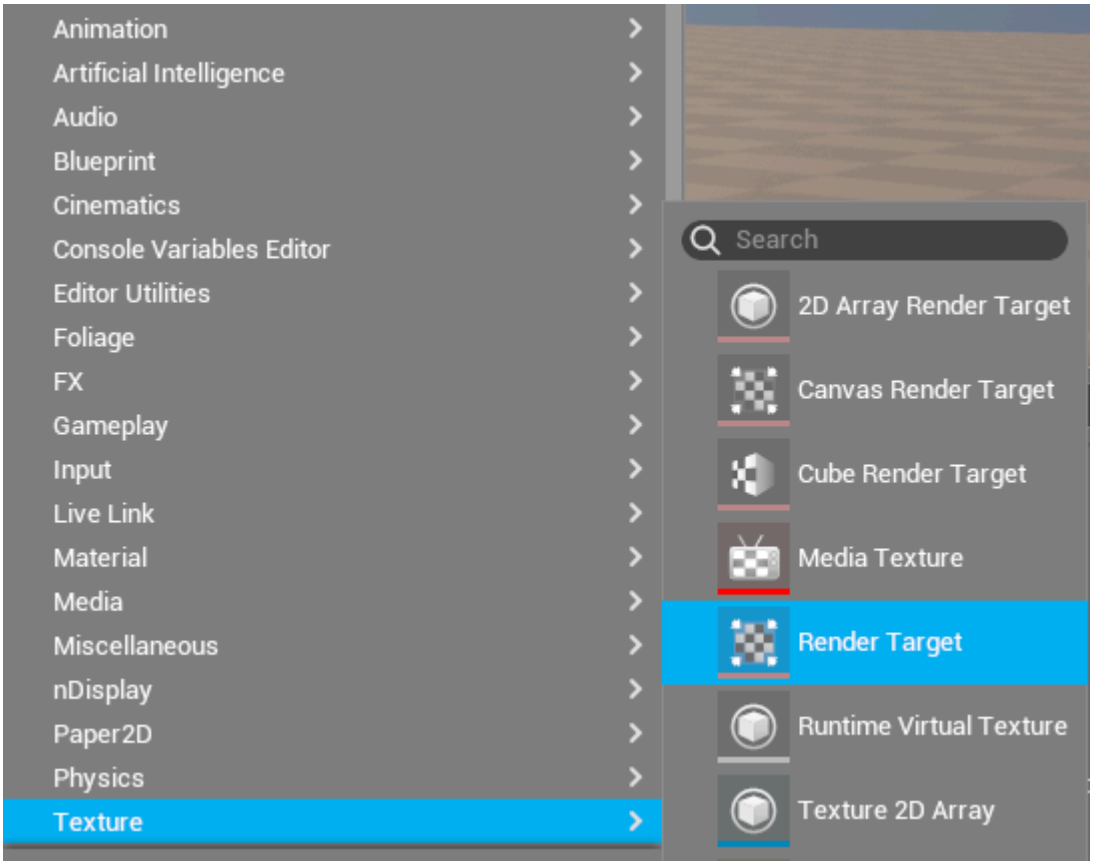
The RenderStream Unreal Engine plugin offers support for the sharing of textures remotely through the use of exposed parameters. This allows a two-way flow of video content between Designer and the Unreal Engine.

Remote Texture Sharing

Create a new material

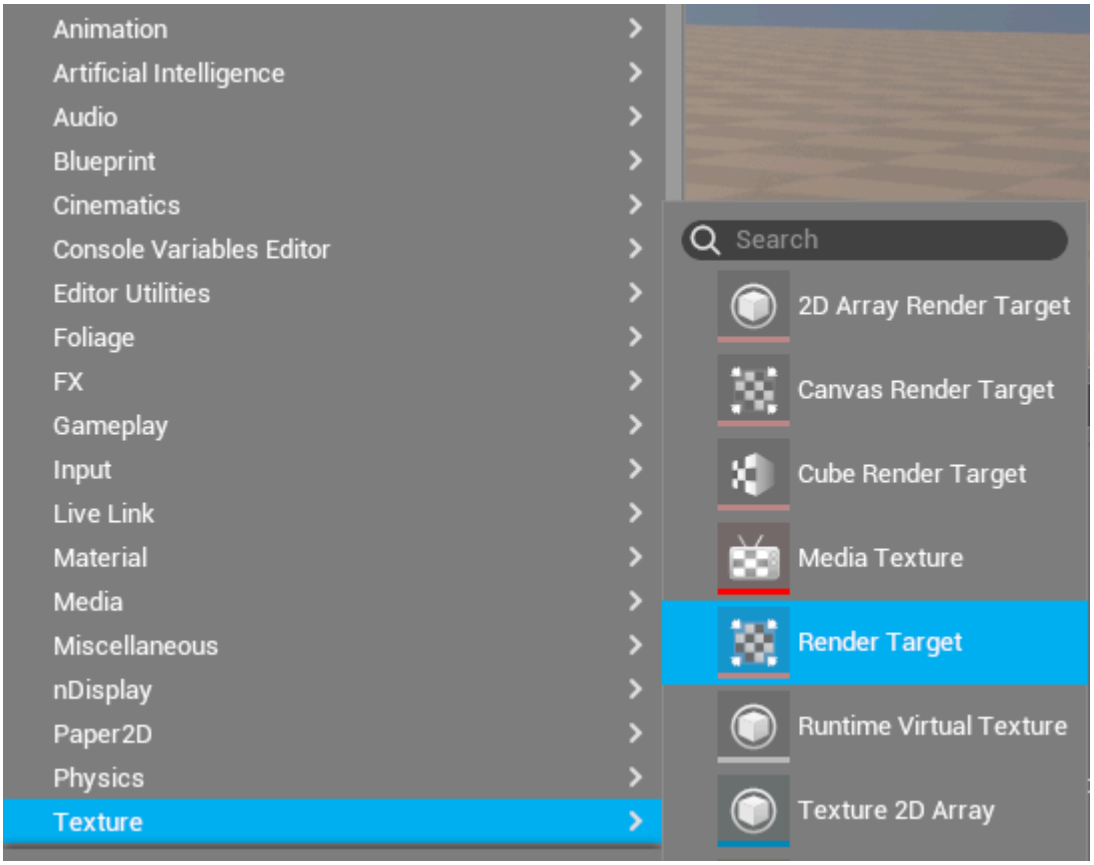
In this example, we will use a Plane to demonstrate remote texture sharing. But you can use any object.

1. Add a Plane (or other 3D object) to the scene.
2. Right-click inside the Content Browser and select **Materials & Textures** > **Render Target** to create a new Render Target.



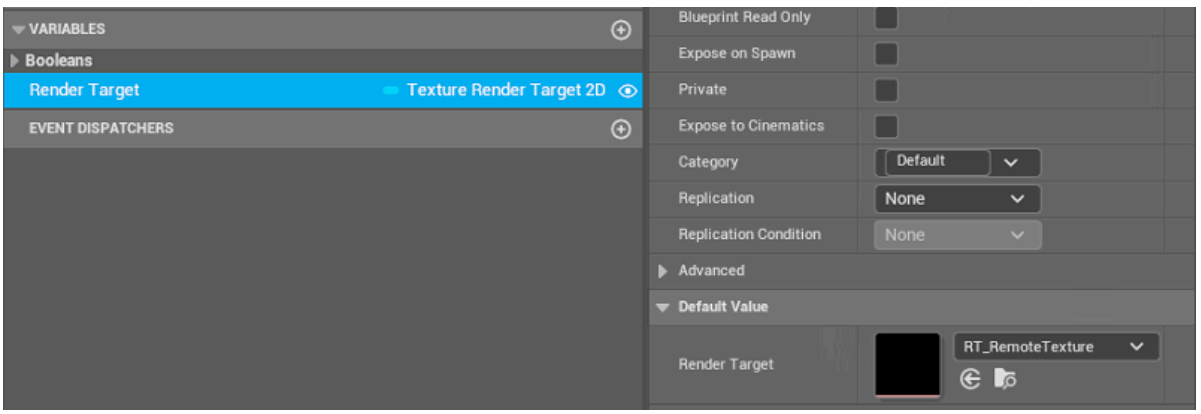
3. Assign the Render Target Texture as a texture of the material on the plane.

4. Confirm that the new Material has been created and set as a material element in the **Materials** component of your object in the Details panel.



Expose the Remote Texture as a remote parameter

1. Open the Level Blueprint.
2. Click the 'Add' button next to Variables and name the variable according to the desired property (e.g. Render Texture) and compile the blueprint.
3. Select the variable and set the 'Variable Type' in the Details panel to **Texture Render Target 2D**.
4. Enable '**Instance Editable**'.



5. Set ‘Category’ to the name of the menu you wish the parameter to appear under in the RenderStream Layer.
6. Save and compile the Blueprint.

Remote Text Parameters

This topic covers the steps needed for configuring text within Unreal Engine for use with RenderStream.

Text Parameters

This feature allows you to render text in the Unreal Engine scene that can be changed live in Designer.

Create a Text Render actor

By adding a Text Render actor to your scene, you will be able to adjust the text properties such as colour, size, and text, from Disguise using the RenderStream Layer. Ensure that you have the RenderStream plugin added and enabled to the Unreal Engine project.

1. Add a **Text Render** actor to your scene. this will give you the ability to generate text in the scene.
2. Open the level Blueprint and then drag **Text Render Actor** from the Outliner into the Blueprint.
3. Drag from the blue pin of the **Text Render Actor**, this will then prompt you to choose a new action.
4. Search for and select **Set Text**.
5. In the Set Text action, right-click on **Value** and select the option to **Promote to variable**.
6. Select the Value node. This will open the Details panel where you can edit aspects of the variable.
7. Click to enable the checkbox **Instance Editable**.
8. Add an EventTick to the Event Graph, and link this to the input of the Set Text action.
9. Save and compile the Blueprint. this will now give you an option to set the default values in the Text Variable.
10. Save and close the project.

Adjusting the Text properties in Designer

1. In your Disguise Designer project, open the RenderStream Layer editor. The text parameter properties should now be visible.
2. Edit the text parameters.

Level Sequences

RenderStream is the proprietary Disguise protocol for controlling third party render engines from Designer. This topic covers the steps needed for configuring Unreal Engine for use with RenderStream.

The RenderStream Unreal Engine plugin allows a Level Sequence built within an Unreal Scene to be triggered via a RenderStream Layer. This means that if any actor is animated within the UE scene using timeline functionality, it can be controlled from within Designer from the timeline.

Level Sequences & Time Control

Create a Level Sequence

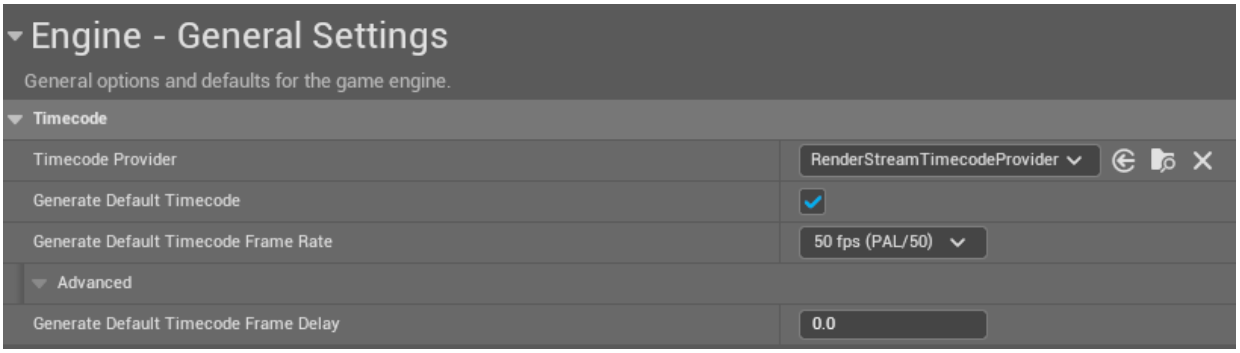
1. Open your UE project.
2. Select an Actor that you wish to animate, or add a new Actor.
3. Click Cinematics in the Toolbar and select **Add Level Sequence**.
4. Name the new Level Sequence and Save the project.
5. In the Sequencer, click the **Track** dropdown and click **Actor To Sequencer** and select the Actor you wish to animate.
6. Ensure your playhead is over the first frame, and add a keyframe.
7. Continue adding keyframes for any or all Transform properties by clicking the small + button (**Add a new keyframe at the current time**).
8. Move the playhead along the Timeline and modify the actor's properties either by using the 3D controls within the scene or by updating the value directly from within the Details panel. Add another keyframe in the same manner.
9. Return the playhead to the beginning of the Timeline and play the sequence to confirm your animation is correct.
10. Open the **Sequence Display Rate** dropdown (fps counter) and set the display rate to that of your Disguise project.
11. Set the Clock Source to **Timecode**.
12. Save the Level Sequence.

Configure the Level Sequence

1. Select the Level Sequence in the Outliner panel.
2. Navigate to the Playback tab in the Details panel.
3. Enable **Auto Play** and configure other options according to your preferences.
4. Save Current.

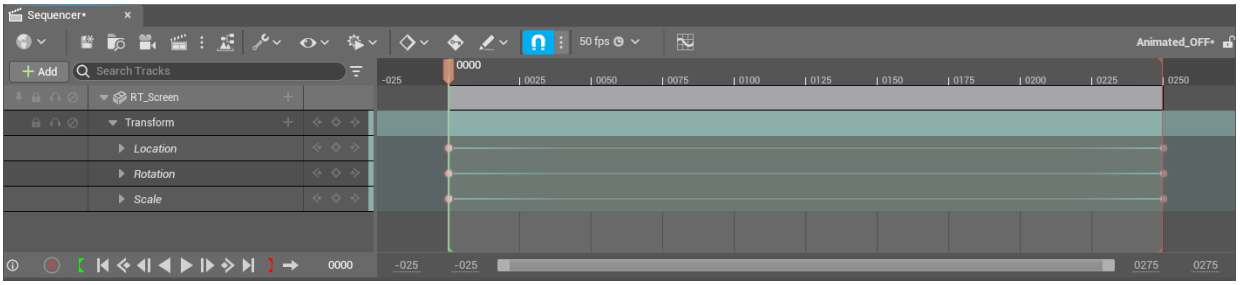
Configure project settings

1. Navigate to **Edit** > **Project Settings...**.
2. In the Engine section of the left panel, navigate to General Settings.
3. Set Timecode Provider underneath the Timecode separator to `RenderStreamTimecodeProvider`.
4. Save the sequence and scene.



Play the animation sequence in Designer

1. Open the **RenderStream Layer** in Designer and **Start the workload**.
2. Play your timeline within Designer to see the animation sequence.



Multi-User Editing

This topic outlines the workflows needed for configuring Multi-User editing within an Unreal Engine project. This includes the configuration of the UE project, and the setup of the Designer project.

If a Multi-User Server is configured and running, any additional machine on the same network as the render nodes will be able to run the project in Edit mode and make live changes. Any changes to the project made while there is an active workload will be reflected in real-time.

- Client-Server Live Collaboration
- Real-time Synchronization of Assets
- Lightweight Transaction-based Sync
- Compliments Source Control
- Not necessary to use Source control but is recommended
- Multiple platform Support During Collaboration
- Works over LAN or VPN

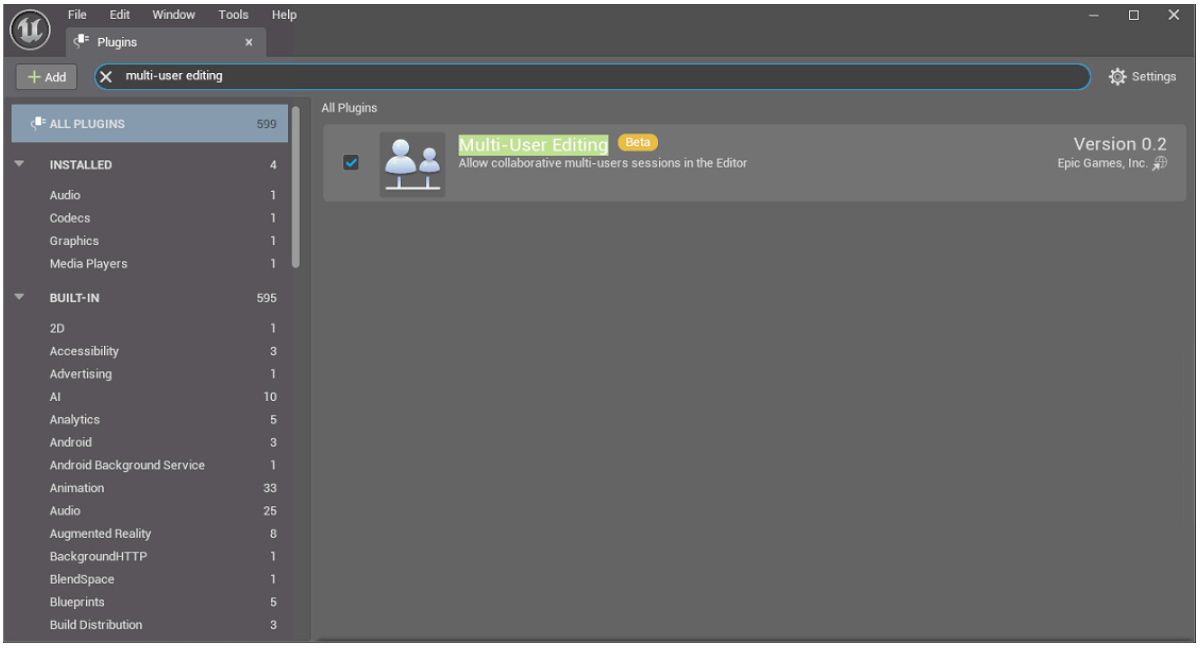
Prerequisites

- All clients must be using the same Engine build.
- All computers must be on the same LAN or VPN.
- All users must be working on matching project name.
- All users must start with exactly the same content - typically synced from source control.

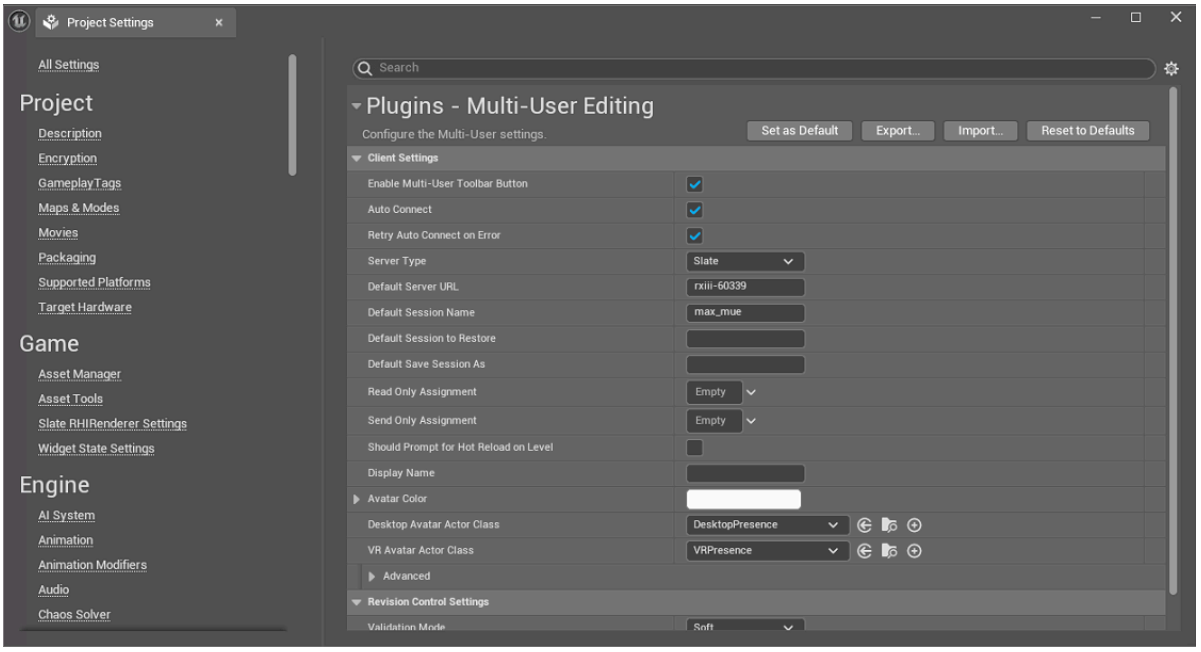
Unreal Engine Workflow

Configure Multi-User Editing in a UE project

1. Ensure the **Multi-user editing** plugin is enabled in the UE project. Navigate to **Edit** > **Project Settings** > **Plugins** > **Multi-User Editing**.



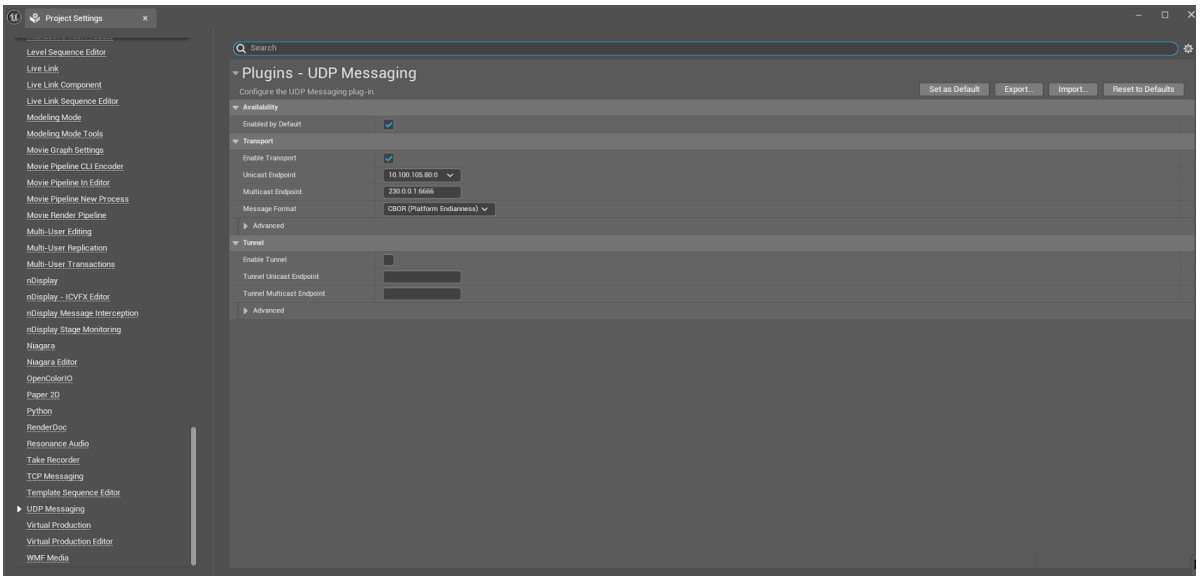
2. Ensure **Enable Multi-User Toolbar Button**, **Auto Connect** and **Retry Auto Connect on Error** are enabled.
3. Set the **Default Server URL** to the name of the machine that will be running the Multi-User Server (e.g. “**rxiii-xxxxx**”).



4. Set **Default Session Name** to something unique (e.g. “MUE1”).

Configure UDP Messaging in UE

1. In **Project Settings** > **Plugins** > **UDP Messaging**, set the Unicast Endpoint to the network address that will used for the RenderStream pool Preferred Synchronisation Adaptor (preferably the Media network).



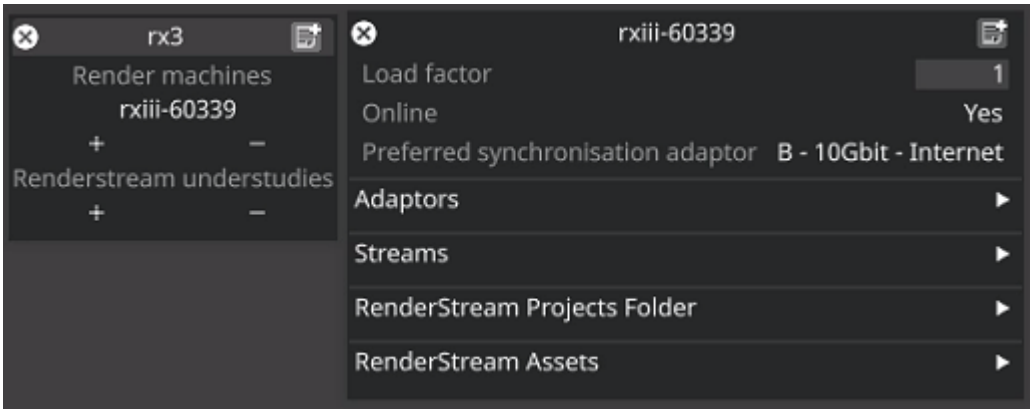
Designer Workflow

Configure Engine Settings in Designer

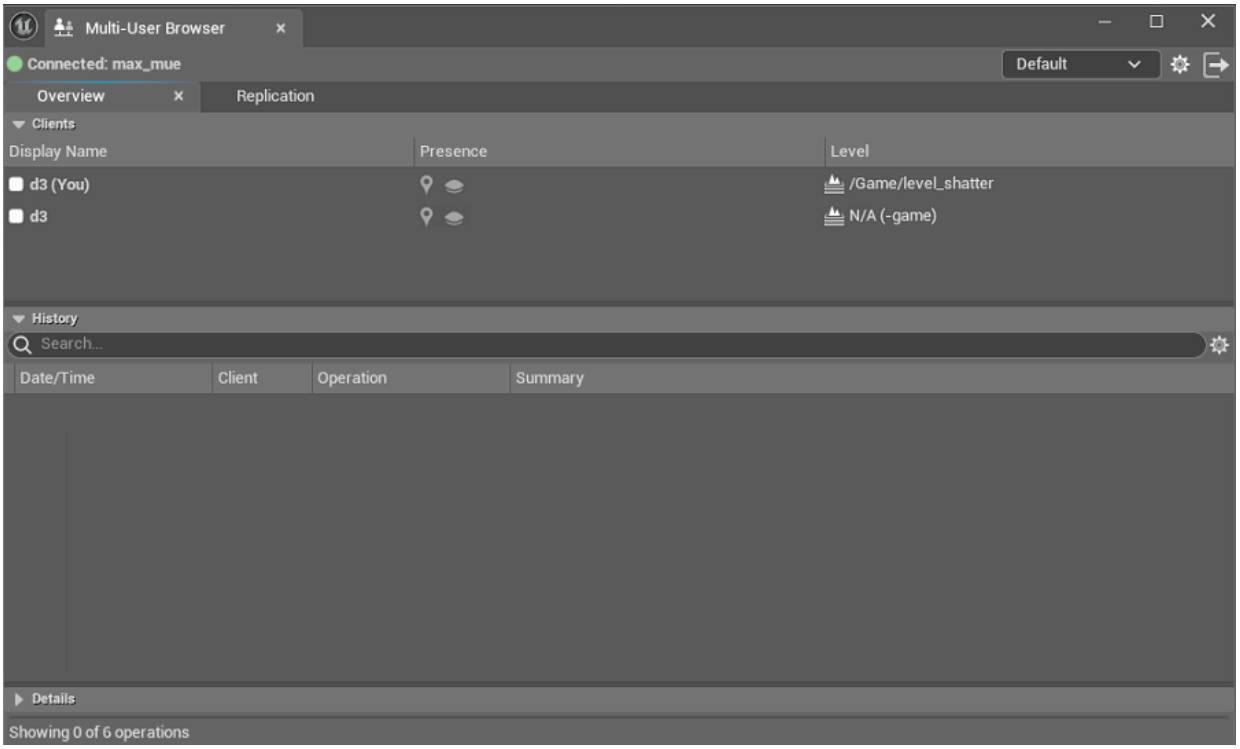
1. Open the RenderStream layer editor and right-click on the Workload name.
2. In the Cluster Workload editor, ensure a RS Asset is selected, then right-click on the Asset name.
3. In the Asset editor, right-click on the **Engine settings** property, `UnrealEngineSettingsResource`.
4. Check the box to **Enable multi-user edit**.
5. In **Multi-user session name**, add the **Default Session Name** you previously created in UE (e.g. “MUE1”).
6. In **Multi-user session host**, set the hostname to the name of the machine that will be running the Multi-User Server, the configured **Default Server URL** in UE. (e.g. “rxiii-xxxx”)

Launch a Multi-User session in Designer

1. Ensure the **Preferred Synchronisation Adaptor** has been set on all the render nodes to the same network port set in the **UDP Messaging** setting in UE (the Media Network). To select the Preferred Synchronisation Adaptor, open the Cluster Pool and right-click the machine name to edit each render node. This automatically defaults to **Any** but needs updating for MUE.



2. Sync the project to all render nodes.
3. Launch the Multi-User Server on the machine specified as the **Default Server URL** either launched standalone `UnrealMultiUserServer.exe` or through the Unreal Editor.
4. **Start the workload**. When the workload is running it will connect to the Multi-User Server.



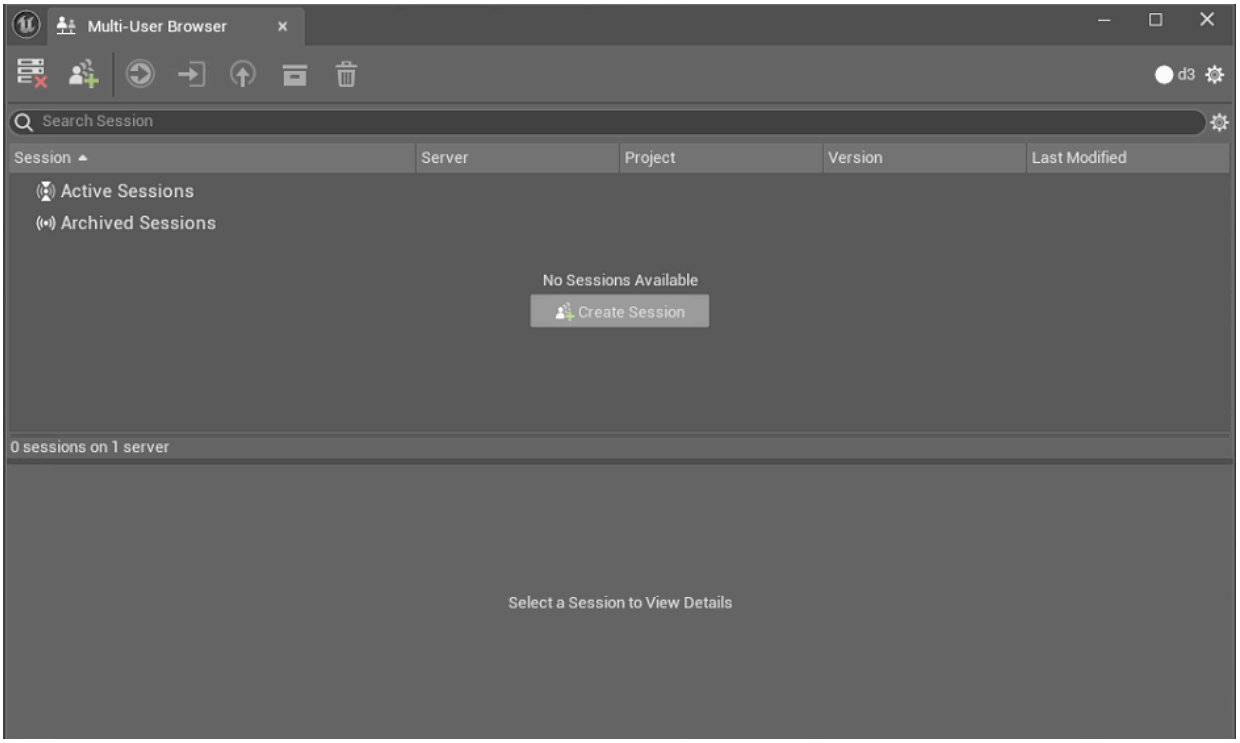
Joining a Multi-User Session

Multi-User Browser

Once a Multi-User Server and Session have been launched on your network, joining is very simple.

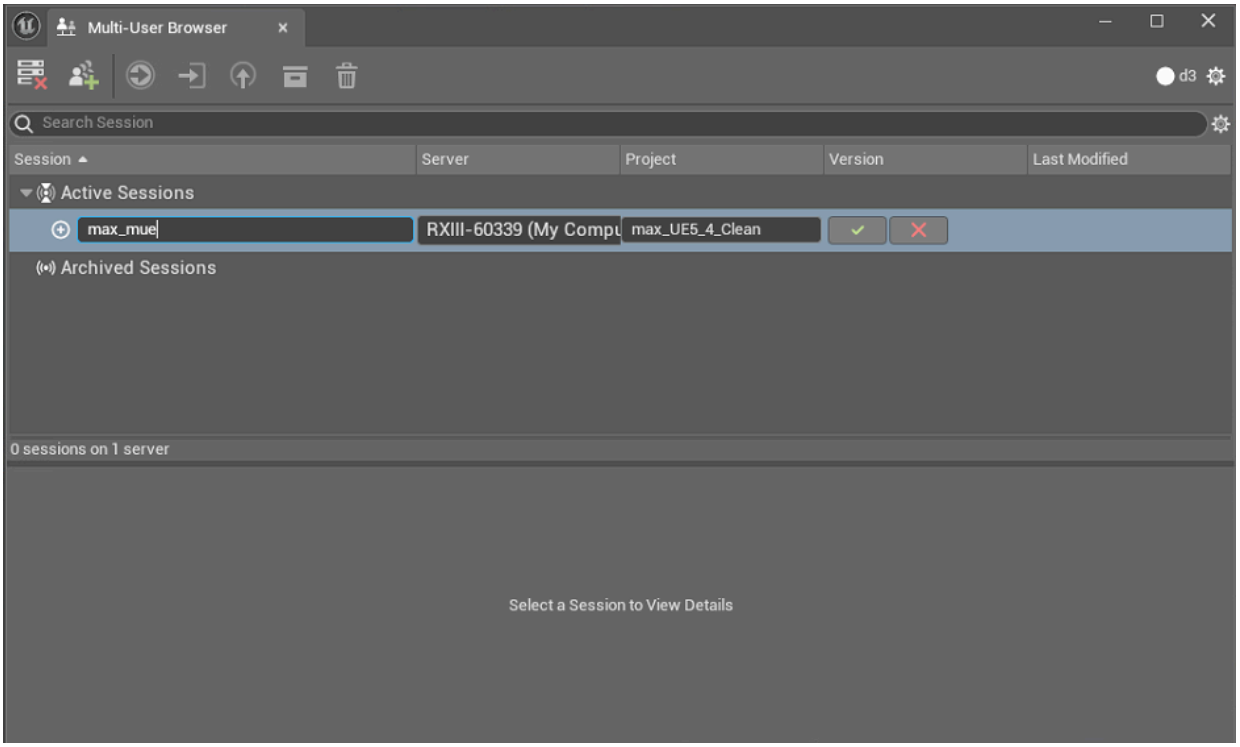
1. Open your project and navigate to **Window > Developer Tools > Multi-User Browser**. This will open the Multi-User Browser window.

-



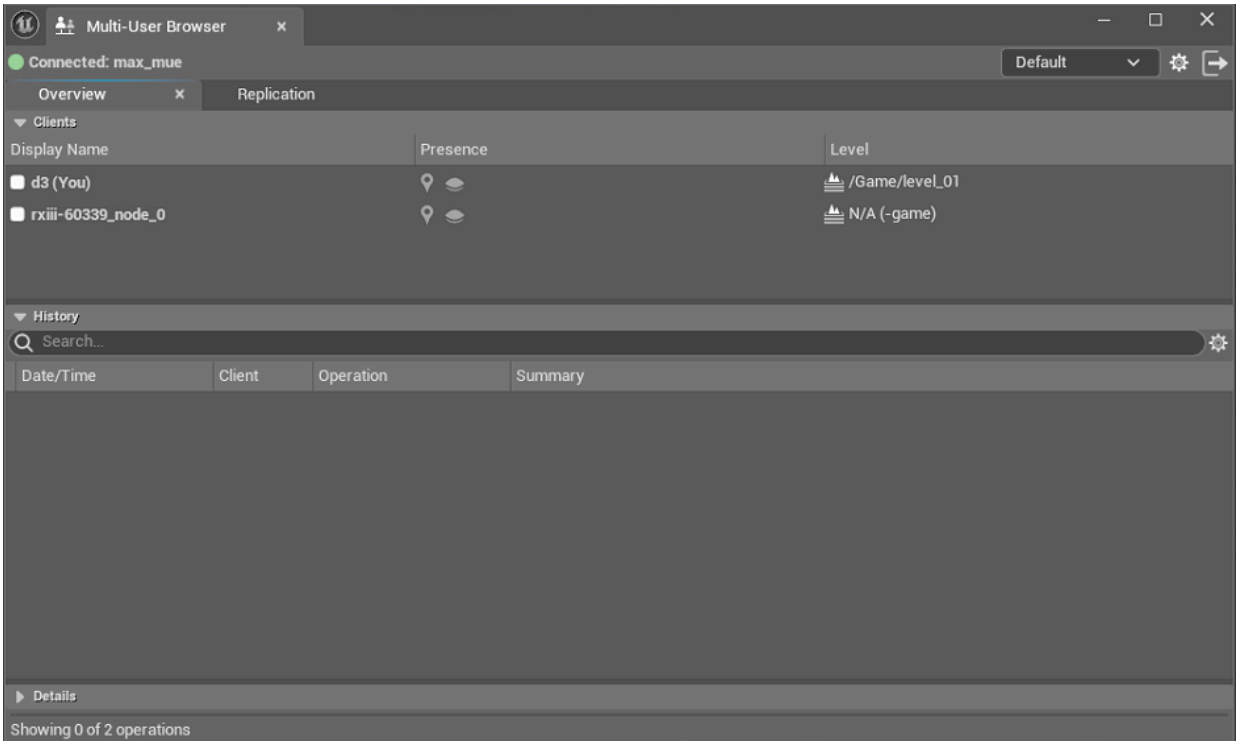
Multi-User Browser

2. If a Multi-User Server and Session have been launched, and you have correctly configured your UDP messaging settings, you will automatically see the available Sessions on your network. Double-click to join.



MUE Active sessions

3. All clients connected and in session are listed in the Multi-User Browser.



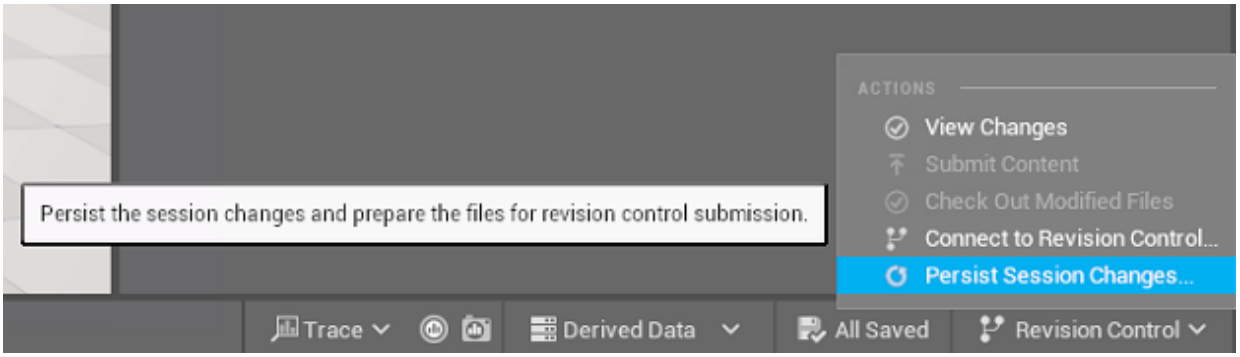
Connected clients in the MUE session

Saving changes in a session

Changes made in a Multi-user session are not persisted to the project file on the connected Render Server. If a Render Server leaves the session it will have it's original content and not the

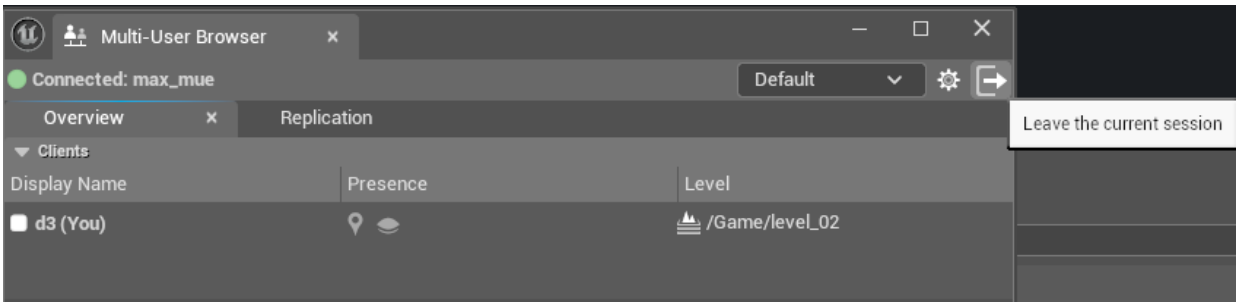
changes from the Multi-user session.

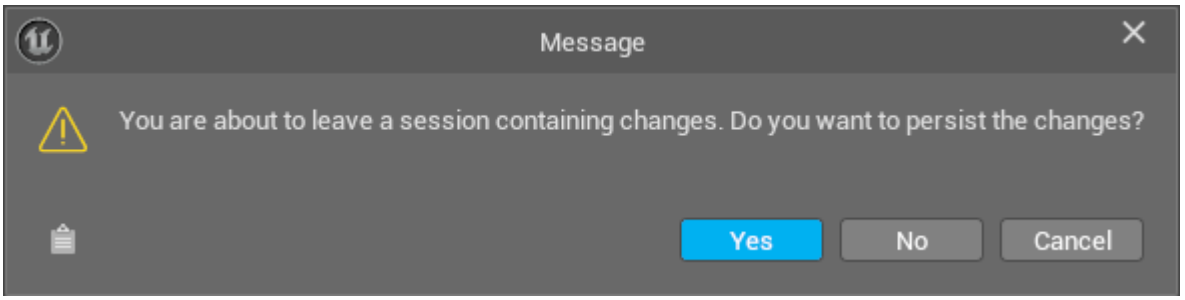
You can persist the files while editing using the **Revision Control** button in the bottom right and clicking **Persist Session Changes...**

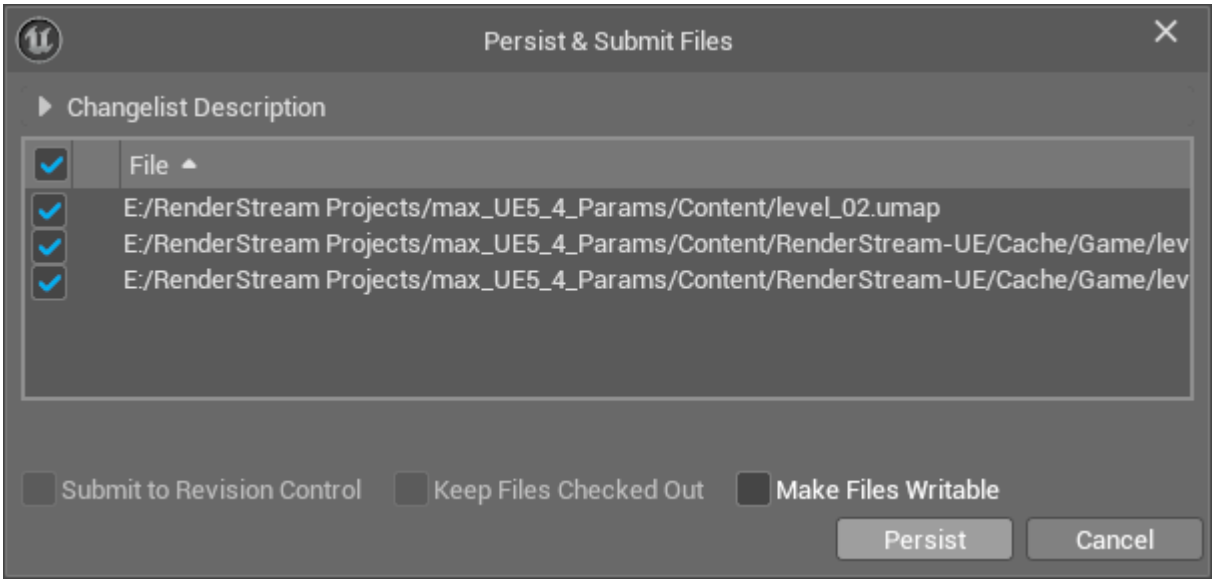


To persist the files once adjustments have been finalised, click the **Leave the current session** button, **Yes** on Do you want to persist the changes? and then **Persist** after reviewing the files to Persist from the session.

Then you can sync the project file to the Render Servers and rejoin the session.







Additional information

- Multi-User Editor will use the Preferred Synchronization Adaptor selected in the d3 project to send nDisplay communication messages, including MUE data. Therefore, all servers must be connected to a common network that can be designated as the Preferred Synchronisation Adaptor and must be able to communicate with each other via ping over this network.
- The MUE plugin must be enabled in UE and the server must be launched within UE. We've provided instructions for launching the session through the UE editor. If you prefer to use a console command, please refer to the Unreal Engine documentation.
- Adding Perforce can add a layer of complexity to troubleshooting issues, but does provide invaluable Source control.
  - When using Perforce, Source control must be enabled in the Editor and the content must be on the latest revision (ie. pull latest in Perforce).
  - It is very helpful to check out the **Engine.ini** file before making changes in Project Settings when using Perforce. You can find it under Root Project Folder > Saved > Config>WindowsEditor.
- When making large amounts of changes, make sure that you save and persist often. While persisting, it is helpful to un-toggle anything being persisted to the "UE-Renderstream-Cache" folder.

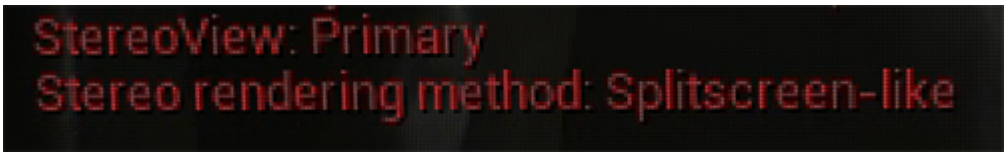
To learn more about Multi-User Editing (MUE) from Epic's official documentation: https://dev.epicgames.com/documentation/en-us/unreal-engine/multi-user-editing-in-unreal-engine?application_version=5.4

Unreal Engine - Troubleshooting

Troubleshooting RenderStream can be complex. This guide outlines common issues and provides best-practice solutions to help resolve them.

Remove `StereoView-Primary` Screen Message when visible in RenderStream

Below are instructions about how to remove the `StereoView-Primary Stereo rendering method: Splitscreen-like` message that appears on RenderStream using Unreal Engine 5.2 and above. Messages can appear when the **Pixel Streaming** plugin is enabled in the asset.

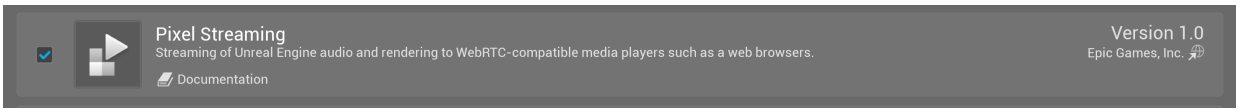


StereoView-Primary Stereo rendering method: Splitscreen-like

Disabling **Pixel Streaming** plugin will prevent this message from appearing. However, as the plugin is needed for a range of workflows, there is another solution to remove the screen message by updating the Level Blueprint in the Unreal project.

Remove the Pixel Streaming plugin

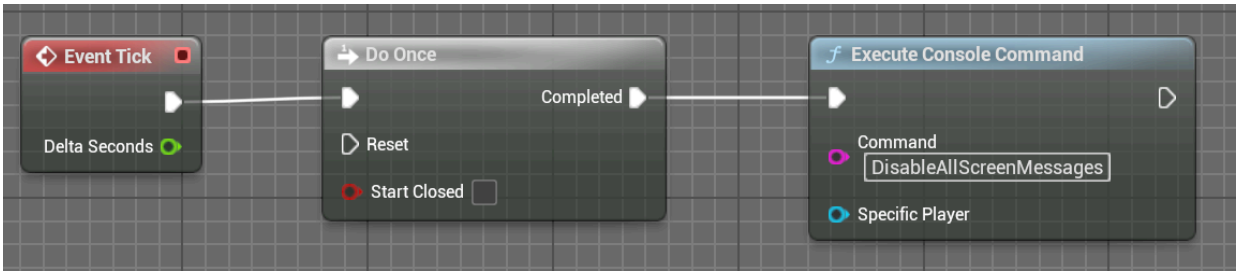
To disable the Pixel Streaming plugin, uncheck the checkbox.



Pixel Streaming plugin shown enabled

Disable all screen messages using a Blueprint

1. Open the Level Blueprint.
2. Add and connect the following nodes to the Event Graph:
   - **Event Tick**: This event fires every frame.
   - **Do Once**: This node executes its “Completed” output only once. It has a “Reset” input to allow it to execute again. This avoids the logs spamming commands lines.
   - **Execute Console Command**: This node executes the `DisableAllScreenMessages` console command.



DisableAllScreenMessages Level Blueprint

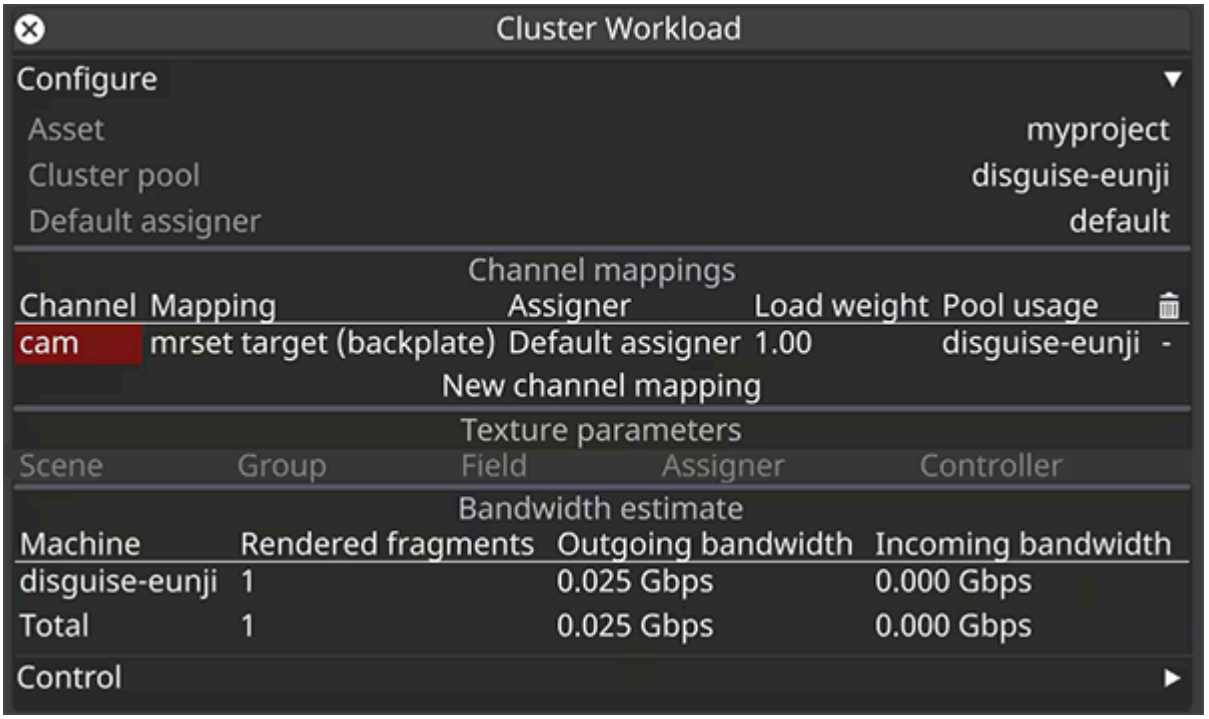
3. Compile, save and close the Unreal project.
4. Open the d3 project and run the RenderStream workload. All screen messages are disabled.

Unreal Engine Renderstream CineCameras not available for Channel mapping

When using the RenderStream plugin in an Unreal Engine project, CineCameras may not appear in the Cluster Workload Channel Mapping if the associated JSON file is not updated correctly.

After adding the plugin, restart the Unreal project.

If the JSON does not updated correctly with the cameras in the scene, it will not work properly and like the screenshot below, and the cine camera will not be available in the Cluster Workload Channel Mapping.



Regenerate the JSON file

1. Close Unreal Engine.
2. Delete the `Renderstream-UE` plugin from the Plugins folder.
3. Download and install the latest Renderstream plugin in the Plugins folder.
4. Enable the RenderStream plugin. After enabling the plugin, restart the project.

5. Open or create a level containing the CineCameras you want to use, and save the level.
6. Check the generated JSON file to ensure that the schema includes the CineCamera channel references. The file will be located at the root level of the UE project and will have a name following the format `rs_<project name>.json`

Scene Optimisation

This topic covers best practices when configuring scenes within an Unreal Engine project for use with RenderStream.

Here are several useful tips and best practices for optimizing your Unreal Engine scenes for use with RenderStream.

Alpha

Tips for using Alpha with Unreal Engine assets:

- Alpha content should be set to **Premultiplied Alpha** in Disguise.
- PostProcessingMaterials do not support alpha from Unreal Engine.
- Set post-processing and effects quality to high/cinematic/epic.
- Atmospheric fog will need to be disabled in channels rendering front plate alpha.
- When a monitor is connected to a render node, ensure that there is 100% scaling to the native resolution. A second monitor can be used as the review screen if the GUI must be scaled.
- Ensure there are no sky domes or full frame effects in the frontplate channels.
- `r.SceneColorFormat` must be set to 4.
- Turning on/off alpha or modifying the transport configuration in any other manner for Unreal assets will cause a shader recompilation and may take a considerable amount of time before the content is visible within Designer.
- Try hiding post-processing methods one by one within the RenderStream Layer as a troubleshooting test if you are not getting alpha through.

> ⚠ **Warning**
>
> Turning on alpha or modifying the transport configuration in any other manner for Unreal assets will cause a shader recompilation and may take a considerable amount of time before the content is visible within Designer. This process will be reflected within the Workload status widget.

Limitations

Limitations of using certain Unreal Engine effects with RenderStream.

There are a number of default options in Unreal Engine that are not suitable for use with any splitting strategy in a Cluster pool. In some cases this is because the whole image is required to be present in order to work; this can sometimes be mitigated with padding and overlap, but this must be evaluated on a case-by-case basis. In other cases, it is because additional buffering is required.

The RenderStream **Channel Definition** component settings contain flags allowing many of these to be enabled / disabled on a per-channel basis.

Below is a non-exhaustive list of such effects:

- Depth of Field
- Bloom
- Vignette
- Lens Flare
- Temporal Anti Aliasing (TAA)
- Screen Space Global Illumination (SSG)
- Screen Space Ambient Occlusion (SSAO)
- Screen Space Reflections (SSR)
- Raytracing (denoiser)
- Chromatic Aberrations
- Eye Adaption
- Motion Blur

> ⚠ **Warning**
>
> Many features within Unreal Engine will not work with nDisplay, specifically nodes that require determinism. Please consult Unreal Engine documentation for more information.

Post Process Volume

By default, Unreal Engine automatically sets the exposure of a map's **PostProcessVolume**. This can cause the exposure of split frames to be different, revealing a noticeable seam between frames. To correct this here are some suggested settings:

- Expand the Lens property of **PostProcessionVolume**.
- Expand Exposure.
- Change the Metering mode to **Manual**.
- Adjust the Exposure Compensation.
- Exposure Compensation can either be a user defined float value appropriate to the scene, or a Float Curve can be applied.

- Be aware there are exposure settings within camera and CineCamera objects; ensure they are set to manual exposure as well.

> **⚠ Warning**
>
> Many features within Unreal Engine will not work with nDisplay, specifically nodes that require determinism. Please consult Unreal Engine documentation for more information.

Verification checklist

This topic covers some recommended practice information for those working with RenderStream and Unreal Engine.

General

- The ValidationFramework plugin is enabled to check for any potential nDisplay / RenderStream issues.
  - Supported Versions
  - UE4.27+: Supported v1.0.1 Only: ✅
  - UE5.0: Testing: ❗
  - UE5.1: Supported v1.1.0+: ✅
  - UE5.2: Supported v1.2.0+: ✅
  - UE5.3: Supported v1.3.0+: ✅
- Verify all required plugins are in the project directory and enabled.

Channel Definitions

RenderStreamChannelDefinitions have been added to each Cine Camera Actor in the specific scene

Frontplate

- Frontplate cameras are set to Default Visibility Hidden.
- All components to be added to frontplate are added to Force Visible section.
- Frontplate cameras have the following disabled (unchecked)
  - Fog
  - Atmosphere

Preparation for Source Control Push

Before a push to source control, and especially after making any changes, the Unreal scene should be tested by launching RenderStream via d3. Check for frame drops:

- During operator camera movements
- When triggering exposed parameters
- Test ALL exposed parameters.

Trigger via a RenderStream layer inside d3 to ensure nDisplay / RenderStream is handling these parameters as the artists intended.

Test ALL frontplate objects to ensure they are rendering as the artist intended.

Sometimes these objects need to be built as Actor Blueprints in order to attach specific lighting to them and make sure the object maintains correct lighting conditions on both the backplate and frontplate. See below for example.

Step through all view modes to check for any glaring issues.

For more information on view modes, see Epic's Documentation .

The Viewmodes most relevant to optimization include:

- Light Complexity
- Lightmap Density
- Stationary Light Overlap
- Shader Complexity
- Stationary Light Overlap
- Shader Complexity & Quads
- Quad Overdraw

If any issues persist, use Unreal Insights to diagnose them.

Documentation on Unreal Insights (UE4.26) is available from Epic here.

Derived Data Cache

For information about configuring DDC with Disguise see the DDC page.

DDC PakS

If you download Unreal Engine from the Epic Games Store, the engine will come with a DDC Pak (.ddp). The DDC Pak contains derived data for all engine content, so you can start working without compiling shaders and so on. Similarly, some samples are shipped with a DDC Pak for the same reason.

- Engine DDC Pak

For Example: `EngineDir/DerivedDataCache/Compressed.ddp`

- Project DDC Pak

For Example: `ProjectDir/DerivedDataCache/Compressed.ddp`

Building a project

Considerations while building a project.

An example of an Actor Blueprint with lighting attached:

Actor Blueprint with Lighting attached

Optimization

Optimization is a complex subject influenced by multiple variables such as (but not limited to):

Mobility status for mesh actors and lighting actors

Render settings

Complexity of materials

Amount of Parent Materials vs. Material Instances used.

Complexity of meshes (and for UE5 - whether they are Nanite meshes or not)

Complexity of lighting actors / how many lighting actors are in the scene.

Optimization Recommendations

Use Oodle Texture compression.

When using Lumen, due to the nature of how the light is calculated based on the view of the rendering camera, this may cause seems to appear when using tiled rendering.

Screen-Space Global Illumination should be fine (change this setting under Project Settings > Global Illumination > Dynamic Global Illumination Method).

When using Post Processing Volumes, change this setting by ticking the check box under the PostProcessingVolume Actor under Global Illumination, and changing to Screen Space.

Consider GPU baking as many lights as possible and avoid using Ray Tracing.

Use as few lights as possible.

If you must use a Rect Light, don't use too many and make sure they're baked.

Turn off shadows under individual Light Actors when they're not needed.

Reflection captures don't need a resolution higher than 512.

Learn about Temporal Super Resolution and lower your screen percentage for when using it.

If you're using TAA, use Temporal Upsampling.

Here are some amazing tools to use DURING dev:

UE stat commands

A list of all stats commands for: https://docs.unrealengine.com/5.4/en-US/stat-commands-in-unreal-engine/

Optimise the amount of draw calls

> "UE[4] calculates scene visibility to cull objects that will not appear in the final image of the frame. However, if the post-culled scene still > > contains thousands of objects, then draw calls can become a performance issue. Even if we render meshes with low polygon count, if there are too many draw calls, it can become the primary performance bottleneck because of the CPU side cost associated with setting up each draw call for the GPU. Both UE4 and the GPU driver do work per draw call.

> However, reducing draw calls is a balancing act. If you decide to reduce draw calls by using few larger meshes instead of many small ones, you lose the culling granularity that you get from smaller models."

- Consider combining actors for less draw calls.
- One mesh with 50 mat IDs is better than 50 meshes with one mat ID each (51 vs 100).
- Use the console variable editor.

General Onsite Advice

- If the player can't see an object, don't bother rendering it.
- Trees can be the biggest challenge especially when filling a scene with them. They have an impact on draw calls. High poly assets with multiple materials slots are problematic.
- Using a visual LOD system can be one solution with better assets closer to the camera, and lower quality ones behind. While a treeline texture plane is useful for the background.
- Also, turning off all shadows and systematically turning them back on can create the desired effect and a lighter weight project. To compensate for highlights appearing flat as a result, the light function on the Directional Light can be altered to help fake slight foliage shadows.
- Replacing the Atmospheric Sky and Volumetric Clouds with a sphere that has an unlit shader is a good solution when the time of day does not change in a scene.

- Volumetric fog can be improved by:-
  - Tweaking the View Distance space
  - Using the following console commands:

    `r.VolumetricFoc.GridPixelSize`

    `r.VolumetricFog.DepthDistributionScale`

    `r.VolumetricFog.GridSizeZ`