PCA 3D ADVENTURE STUDIO

# 3D Models and Video in GUIs

## Build inspection screens, animate models and control embedded clips

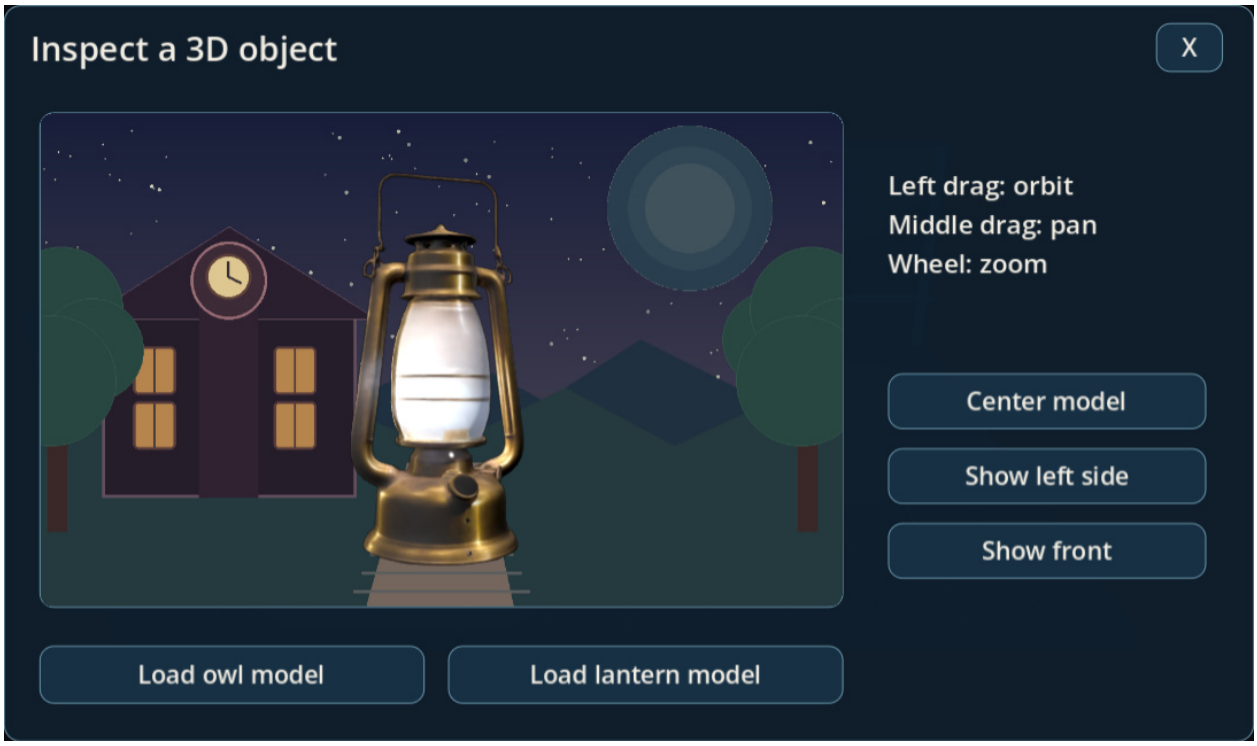


Figure 1. A 3D model displayed inside a GUI, with a background image and separate view controls.

A practical guide for Windows, macOS and Linux.

Editor reference: 0.86.3.4 | Companion to the GUI Designer tutorials

# Before you begin

This guide explains the GUI 3D Model and Video elements from setup through gameplay. The examples use a lantern, an animated character and a short video test clip, all displayed inside ordinary GUI windows.

You will learn which fields change the view, which controls the player can use and how to drive the elements with actions or scripts. The video chapters also explain behavior that depends on the decoder, so a setting’s label is not mistaken for a promise that every format behaves identically.

The screenshots are from the running editor and its game preview. The instructions are shared across desktop platforms. No practice project or extra practice files are supplied; you can follow along in a copy of an existing project.

| You should already know… | If you need a refresher… |
| --- | --- |
| How to create a GUI and add elements | Use the earlier GUI Designer guide. |
| How IDs differ from assigned names | A dropdown may show Lantern(lantern_model); scripts use lantern_model alone. |
| How to create and apply a script record | The examples use GDScript based on script_base.gd. |
| How to import assets | The asset importer guide covers the broader workflow. A model-specific shortcut is explained here. |

> Element property fields normally update live. Apply script changes in the script editor, then Save Project. Designer Preview tries local visuals and controls; a running game is needed for real events and scripts.

All behavior described here was checked against 0.86.3.4. Later versions may change the field layout or remove a limitation.

# Contents

Click a chapter title or use the PDF bookmarks. Continued pages stay with their chapter.

1. What these elements actually display 4
2. Prepare the assets and names 5
3. Add a GUI 3D Model element 6
4. Choose the initial view 7
5. Frame the model with the camera 8
6. Use Center & Fit instead of guessing 9
7. Allow the player to inspect it 10
8. Add a background and frame 12
9. Import a model from this inspector 13
10. Play an embedded model animation 15
11. Use model actions in a GUI event 18
12. Change the clip, view or model 20
13. When the model viewer looks wrong 21
14. Add a Video element to a GUI 22
15. Choose a video format that can load 23
16. Understand the playback settings 24
17. Set the audio deliberately 25
18. Lay out the content rectangle 26
19. Give a failed clip a useful fallback 28
20. Build a Play / Restart Button 29
21. Pause, resume and stop 30
22. Seek to a position in the clip 32
23. React when a finite clip finishes 33
24. Stop the movie when closing the GUI 34
25. Change the clip or its settings 35
26. Start playback when the window opens 36
27. Run the controls through action lists 37
28. Add captions, labels and a title card 38
29. Use Preview for the right checks 39
30. Keep playback and resource use tidy 40
31. Assemble a complete inspection screen 41
32. When embedded video does not work 42
33. A reference for the model viewer 43

# 1. What these elements actually display

| Element | Use it for | Keep this distinction in mind |
|---|---|---|
| 3D Model | An inspectable item, character showcase, trophy or rotating object display. | It loads a model asset into a separate viewer. It is not the actual room character. |
| Video | A clip inside a GUI rectangle: a briefing, monitor, illustrated menu or help screen. | It has its own player. Fullscreen movie actions use another playback path. |
| Image | A still illustration or a sequence of image frames. | It does not become a 3D model or video merely because the picture depicts one. |

The model viewer supplies its own small 3D world, camera and lights. Changing its view does not move the room camera, change the player’s position or assign a character’s gameplay animation states.

The Video element draws a stream within a GUI. It does not automatically add a play bar, pause button, subtitles or a completion action to your window. Build the controls you want around it.

> The shared Center GUI Model and Play GUI Model Animation actions target the GUI model viewer. The shared Play Video / Stop Video / Seek Video actions target the general movie player, not a particular embedded GUI Video element.

# 2. Prepare the assets and names

Begin with assets that you know load correctly. Use one static model, one model with embedded animation clips and one short video before adding a large collection.

| Example ID | Assigned name | Purpose |
|---|---|---|
| lantern_model | Lantern | Static inspection model. |
| owl_model | Owl | A second model for the replacement exercise. |
| professor_model | Professor | Animated model with embedded clips. |
| workshop_clip | Workshop Clip (Theora) | Short Ogg Theora video test. |
| mp4_clip | Workshop Clip (MP4) | A decoder-dependent comparison clip. |
| workshop_art | Moonlit Workshop | Background and fallback poster image. |

The model examples come from Clockmaker project assets. These IDs were chosen for the guide; a new project may use different IDs or need an import. Choose your matching assets in the importer, keep their IDs stable and use readable assigned names. A GUI element ID identifies the display; a model or video asset ID identifies the content loaded into it.

Check the model in the asset preview first. For animation, confirm that its file actually contains clips. For video, check the clip’s codec and duration; an extension by itself does not guarantee a usable stream.

> The clip in the screenshots is a short test pattern. A plain, reliable test clip is useful while setting up playback controls. Replace it with your story footage once the workflow is working.

# 3. Add a GUI 3D Model element

1. Create a GUI called 3D Object Viewer with ID model_lab.
2. Add a 3D Model element and give it ID object_viewer.
3. In General, set X 32, Y 96, Width 720 and Height 444 for this example.
4. Open Images. Set Media Asset to your Lantern model asset.
5. Open Type Settings. Leave Starting Camera Distance at 0 for automatic fitting.

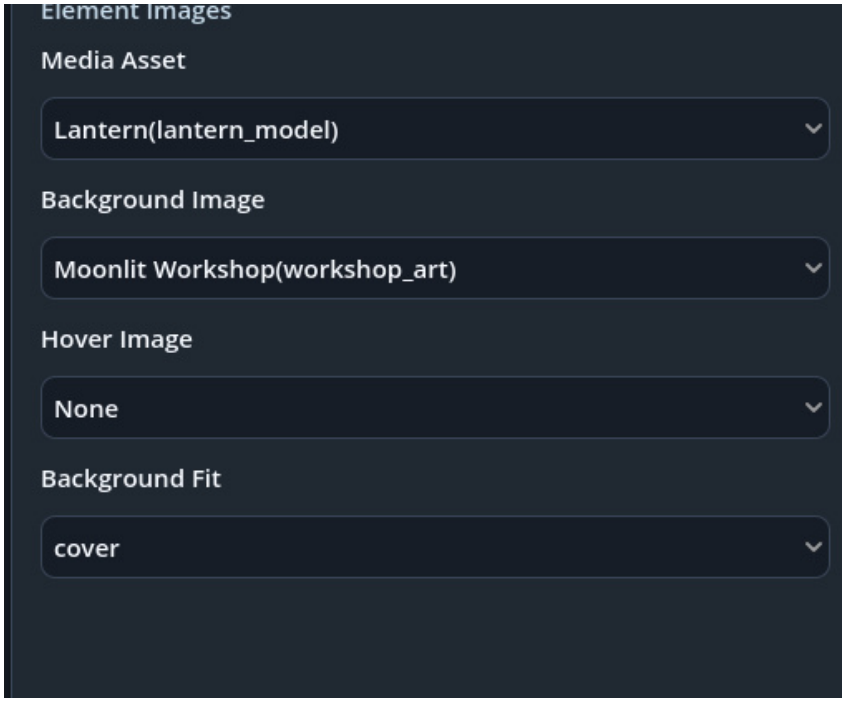


Figure 2. Images contains the model’s Media Asset and the separate 2D Background Image.

The model should appear inside the element rectangle. Add a separate Label for instructions and a Button to close the GUI. A model’s asset name is not automatically drawn as a caption inside the viewer.

> Choose a model asset, not a character definition ID or the ID of a room object. The viewer needs a loadable model resource.

# 4. Choose the initial view

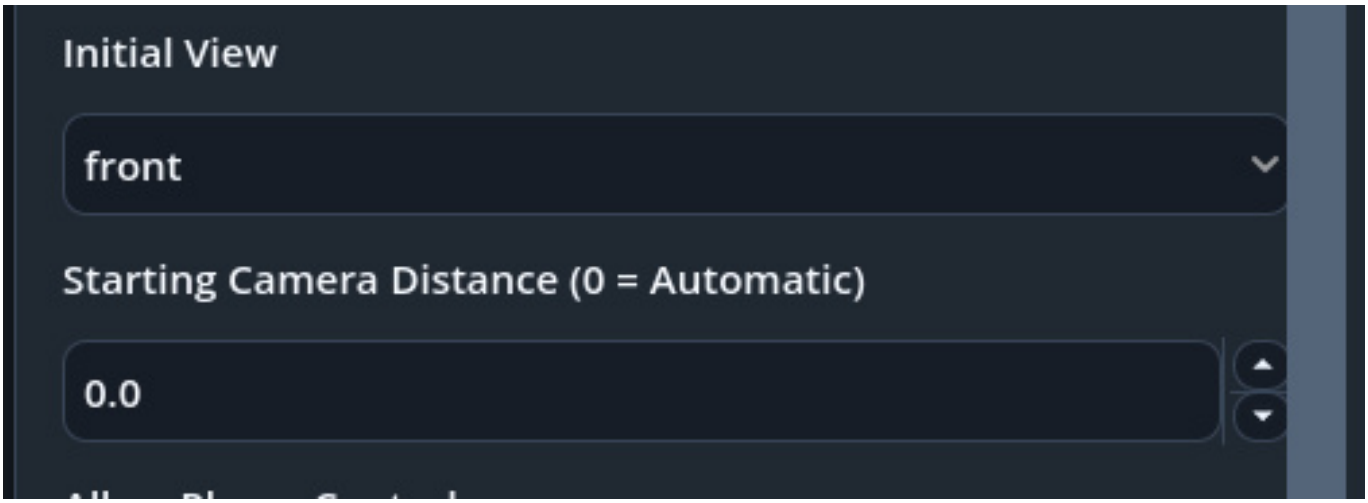


Figure 3. Initial View supplies a preset orientation. A distance of 0 uses automatic fitting.

| Initial View | What it changes |
|---|---|
| front | The model’s imported front orientation. |
| back | Turns the authored view around by 180 degrees. |
| left / right | Side views from opposite directions. |
| side | An alias for the right-side preset in this build. |
| top / bottom | Looks along the model’s vertical direction from opposite sides. |

“Front” depends on how the source model was made and imported. If the lantern or character faces sideways, the viewer has not necessarily lost its camera; its imported orientation may simply be different from the one you expected.

Use Rotation X, Rotation Y and Rotation Z for extra authored orientation. The values are degrees and are added to the chosen preset. A small Rotation Y, such as 20, can make a static item more interesting than a perfectly flat front view.

> Changing the preset or authored rotation reapplies the initial view. It can reset the player’s temporary orbit and pan. Set the starting presentation before handing control to the player.

# 5. Frame the model with the camera

| Field | What it means |
| --- | --- |
| Starting Camera Distance (0 = Automatic) | 0 calculates a distance from the model bounds and viewer shape. A positive value supplies a fixed starting distance. |
| View Offset X / View Offset Y | Shift the viewing target in the viewer’s own world. They do not edit the source mesh’s transform. |
| Field of View | The viewer camera’s vertical angle, from 5 to 120 degrees in the typed field. |
| Width / Height in General | The on-screen rectangle used by this viewer, not the model’s physical dimensions. |

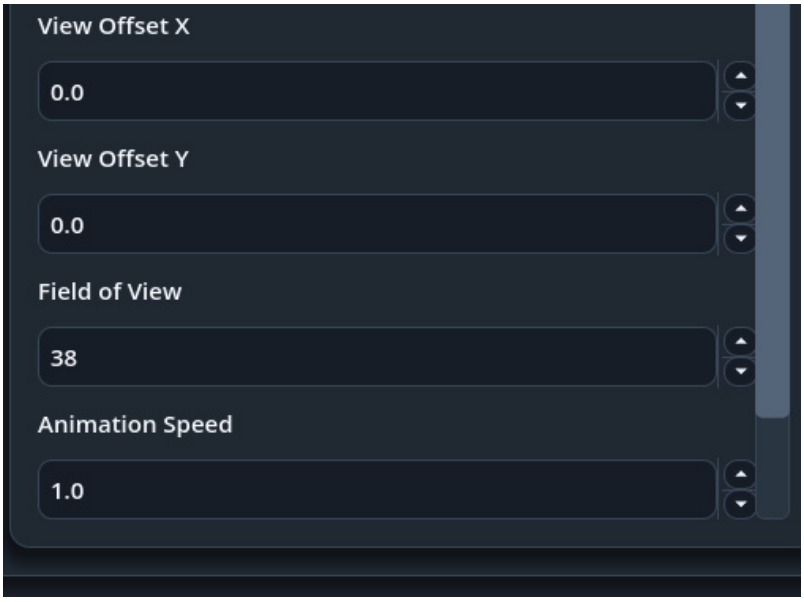


Figure 4. View offsets, Field of View and Animation Speed appear farther down Type Settings.

Start with automatic distance and a Field of View of 38. Enlarge the element if the object needs more screen space. Then adjust the field of view and framing only if the presentation needs it.

With automatic fitting, changing the viewer’s shape can recalculate the camera distance. With a positive fixed starting distance, you take responsibility for keeping the model inside the frame.

A field-of-view change can look subtle when automatic fitting keeps the whole object visible. Perspective and distance are related; neither is the same as resizing the GUI rectangle.

# 6. Use Center & Fit instead of guessing

1. Select the model element and open Type Settings.
2. Scroll to Center & Fit Model.
3. Click it to return Starting Camera Distance to automatic and clear the authored view offsets.
4. Check the view, then save the project when the starting presentation is right.

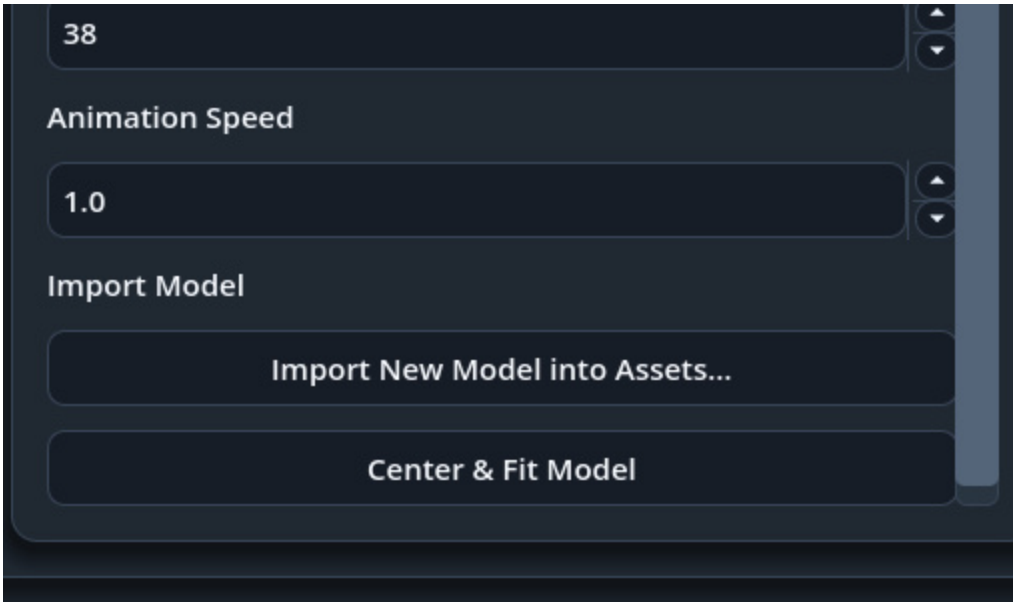


Figure 5. Center & Fit Model is a designer command. The nearby import shortcut adds a model asset.

Centering keeps the chosen Initial View and authored Rotation X/Y/Z. It is a framing reset, not a command to erase every orientation choice. If you want a plain front view, choose front and set the extra rotations deliberately.

In a running game, use the shared Center GUI Model action to give the player a recovery Button. It clears temporary orbit/pan and refits the display while preserving the authored orientation.

> Automatic bounds are measured from the loaded model. A clip with large movements, long attachments or root motion can still extend beyond those initial bounds. Use a larger frame or a more suitable camera distance for that animation.

# 7. Allow the player to inspect it




Figure 6. Allow Player Controls is the master switch. Orbit, Pan and Zoom remain separate choices.

| Switch | Player action |
|---|---|
| Allow Player Controls | Enables the viewer’s interaction system. |
| Orbit Model | Left-mouse drag, or a one-finger drag on supported touch input. |
| Pan Model | Middle-mouse drag, Shift + left drag, or supported two-finger movement. |
| Zoom Model | Mouse wheel over the viewer or supported pinch/magnify gestures. |

Enable only the controls your interface needs. A character showcase might allow orbit and zoom while keeping pan off. A decorative model can have all player controls disabled and still display or animate.

Dragging in designer Preview is a local test. It does not automatically save new Rotation fields, offsets or starting distance. Use the typed fields to author a repeatable starting view.

> Put a short instruction Label beside the model. Players should not have to guess that a picture is interactive or discover middle-drag panning by accident.

# Try orbit, pan and zoom in gameplay



Figure 7. The lantern after a left-drag orbit. Only this viewer’s camera has changed.



Figure 8. Wheel zoom changes the camera distance. It does not resize the element rectangle.

Open the GUI, orbit the model, pan it and zoom in. Release the drag outside the rectangle as well as inside it. Then use Center model to recover the original framing.

Test the intended input device. A touch screen has different gestures from a desktop mouse, and panning or zooming can feel different on a trackpad. Keep a clear recovery control available.

# 8. Add a background and frame

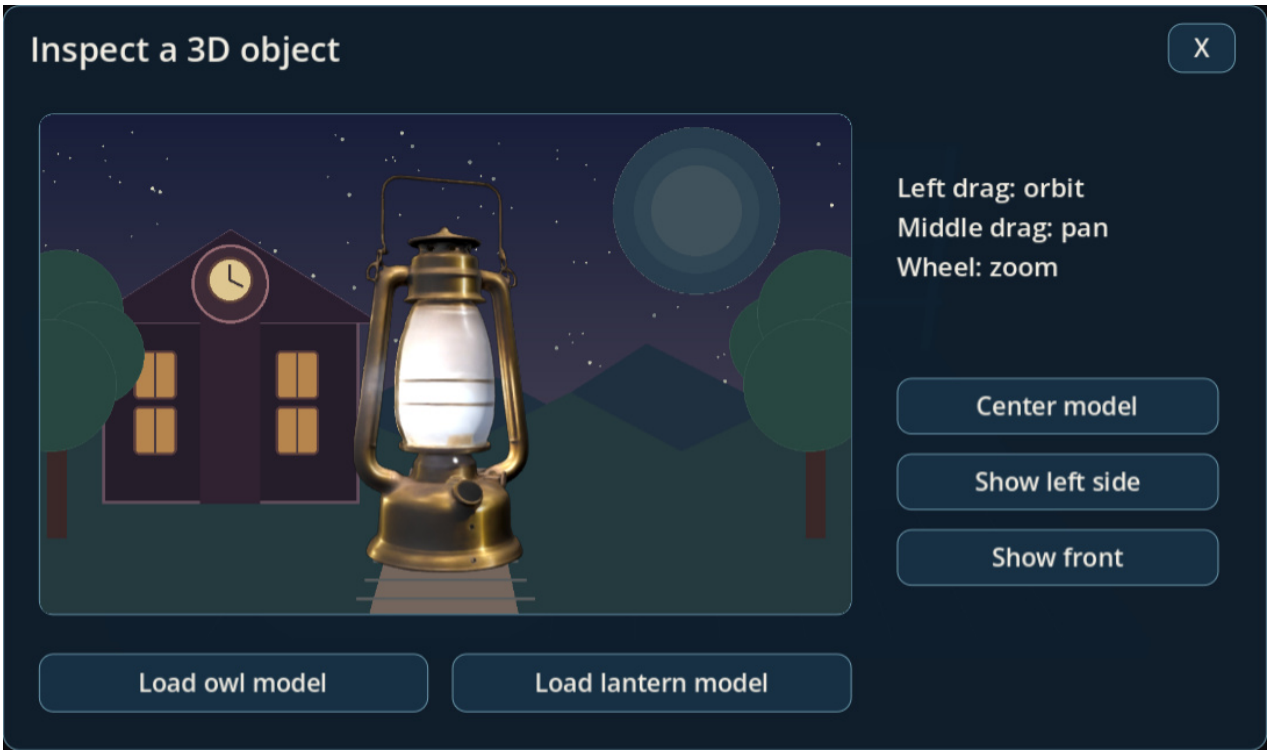


Figure 9. The model is rendered over a 2D background image. The right-hand Buttons are ordinary GUI elements.

In Images, choose Background Image separately from Media Asset. The background is a flat image behind the model; it is not an environment map or a reflective room around the object. Background Fit controls how that image fills the rectangle.

Use Visual Style for the surface color, opacity, border and corners. The main Border Radius supplies the common curve; each corner can inherit with -1, be square with 0 or use its own positive radius.

The viewer has its own lights. Room lights, fog and room camera settings are not a direct way to light this isolated display. A material that looks different here may be responding to that separate lighting setup.

> Rounded background or frame artwork does not guarantee that the entire native 3D viewport is clipped to the same curve. Check the corners with a large, orbited model rather than assuming every child renderer uses a rounded mask.

# 9. Import a model from this inspector

1. Select the 3D Model element.
2. Open Type Settings and find Import New Model into Assets.
3. Choose the source model file. Keep its relative textures, materials and companion files available.
4. Wait for the import or conversion to finish.
5. Check the selected Media Asset, framing and animation list before saving.

| Format in the shortcut | Useful expectation |
| --- | --- |
| GLB | A convenient starting point, especially when the mesh, materials and animations are packed together. |
| glTF | Keep referenced binary and texture files with the source file. |
| FBX / OBJ / DAE | Check geometry, materials and any expected animation after import. OBJ is usually a static-mesh workflow. |
| Blender .blend | Uses the Blender bridge to convert to GLB. A working Blender executable must be available. |

The shortcut imports into the project’s model assets and selects the resulting asset for this GUI element. It is not a link that leaves your finished game dependent on the original file sitting in an arbitrary downloads folder.

> The broader asset importer remains useful for organizing, naming and checking models. Use the shortcut for convenience, then verify the imported asset in the normal asset tools.

# What to check after model import

| Check | Why it matters |
| --- | --- |
| The mesh is visible | A scene or node hierarchy without a usable mesh can leave the viewer empty. |
| Textures and materials came across | Missing dependency files can produce untextured or unexpected surfaces. |
| Scale and orientation are sensible | Automatic fitting can hide a large scale difference, but animation and camera distances still depend on the model. |
| The expected clips are embedded | A character’s gameplay state names do not create clips inside this model asset. |
| The correct asset is selected | A similar editor name can hide a different ID or an older import. |

If a model appears as a simple shape or a proxy, check what the asset importer actually produced. Do not try to fix a missing mesh by increasing the GUI’s border width or changing a character’s Walk state.

For Blender sources, confirm the conversion succeeded before looking for clips in the viewer. If Blender cannot be found or the export fails, resolve that import result rather than expecting the GUI element to convert it during gameplay.

Test with a simple known model when diagnosing a complex one. If the lantern works and a new character does not, the basic GUI viewer is functioning; focus on that character’s imported content and dependencies.

> Do not overwrite an existing working asset merely to test a different file. Import it with a distinct ID, compare the results, then choose the intended asset.

# 10. Play an embedded model animation

1. Add another 3D Model element for an animated asset, or use a separate GUI called animation_lab.
2. Set its ID to avatar_viewer and choose the Professor model asset.
3. In Type Settings, enable Play Model Animation.
4. Choose an actual imported clip from Animation. The example uses PCA_Nod.
5. Enable Loop Model Animation for a repeating showcase and leave Animation Speed at 1.




Figure 10. Play Model Animation and the Animation selector control the clip in this viewer.

The Animation dropdown is populated from AnimationPlayer clips found inside the loaded model. Wait for loading to finish before concluding that the list is empty.

> Automatic means the first available non-RESET clip for each animation player. It does not mean the character’s gameplay Idle state. Select an explicit clip when the exact presentation matters.

# Choose clips by their real names

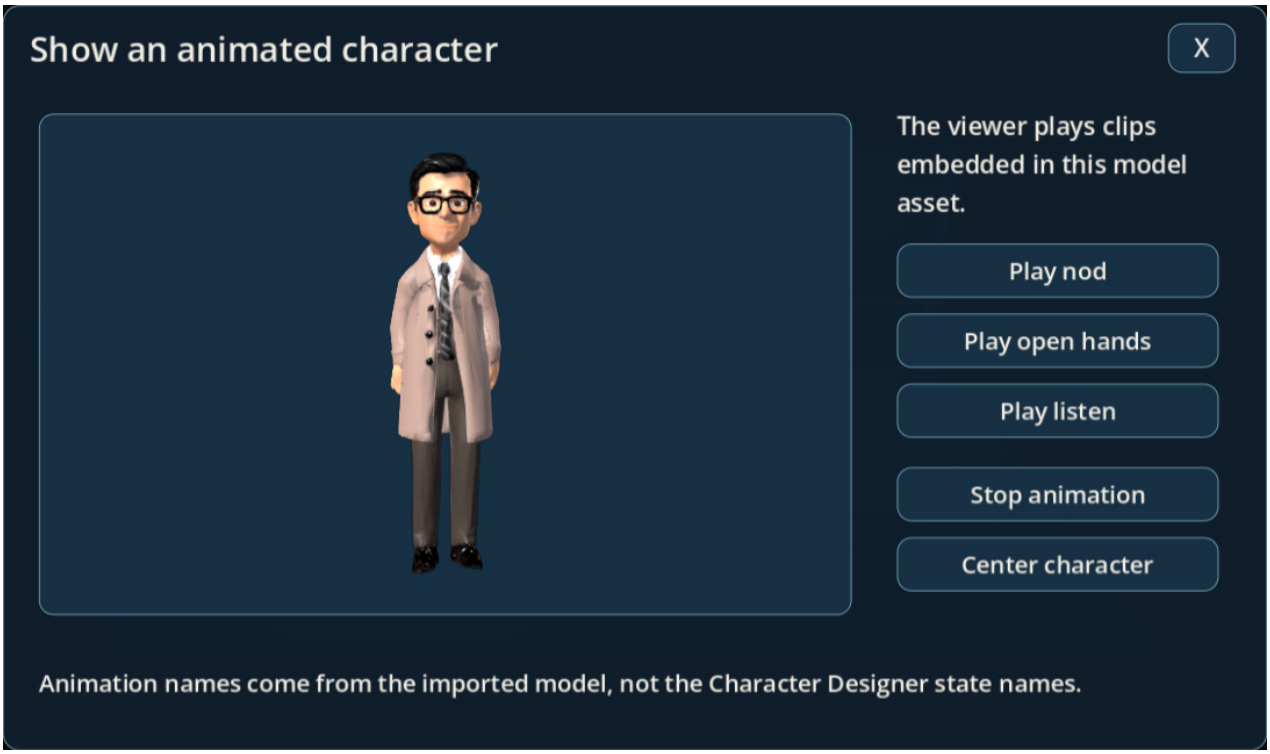


Figure 11. The character model is displayed inside a GUI. It is a separate instance from any character in the room.

Use the clip names the importer actually supplied, including capitalization and any prefixes. A friendly gameplay state called Talk may point to a clip with a very different name elsewhere in the character setup.

| What you have | What to do |
|---|---|
| Several embedded clips | Select the one you want from the viewer’s Animation dropdown. |
| A model with no embedded clips | Use it as a static display, or import a model that includes the animation. |
| A separate animation file | Do not assume it is automatically merged into this GUI viewer. Prepare/import the model with the needed clips. |
| An explicit clip name that does not exist | The viewer stops rather than silently playing an unrelated clip. |

For this example, compare PCA_Nod, PCA_OpenHands and PCA_Listen. Your own model may have entirely different names. The example names are not special commands built into every model.

# Looping, speed and moving beyond the frame

| Setting | Behavior |
| --- | --- |
| Loop Model Animation on | Repeats the selected clip. |
| Loop Model Animation off | Plays without the authored repeat. Do not assume a special “animation finished” GUI event is created. |
| Animation Speed 1 | Normal clip speed. |
| Animation Speed 0.5 / 2 | Half-speed / double-speed playback. |
| Play Model Animation off | Stops the viewer’s authored animation playback. This is not a player-character state change. |



Figure 12. A different embedded gesture playing in the same viewer.

Use an in-place clip for a stable showcase where possible. An animation with root motion can move the model away from its fitted center. A wide hand gesture can also extend beyond bounds measured from the initial pose.

Make the viewer large enough for the whole movement, or choose a fixed starting distance that leaves room. A Center command refits the model’s bounds; it is not an animation-follow camera that continuously tracks every extremity.

> Stopping or changing a clip can affect the displayed pose. If a precise static pose is important, verify it directly rather than assuming every model resets to the same pose.

# 11. Use model actions in a GUI event

1. Add a Button beside object_viewer and give it the caption Center model.
2. Open Events and set Click to Center GUI Model.
3. Choose GUI model_lab and 3D Model object_viewer.
4. Run the game, move the view and press the Button.

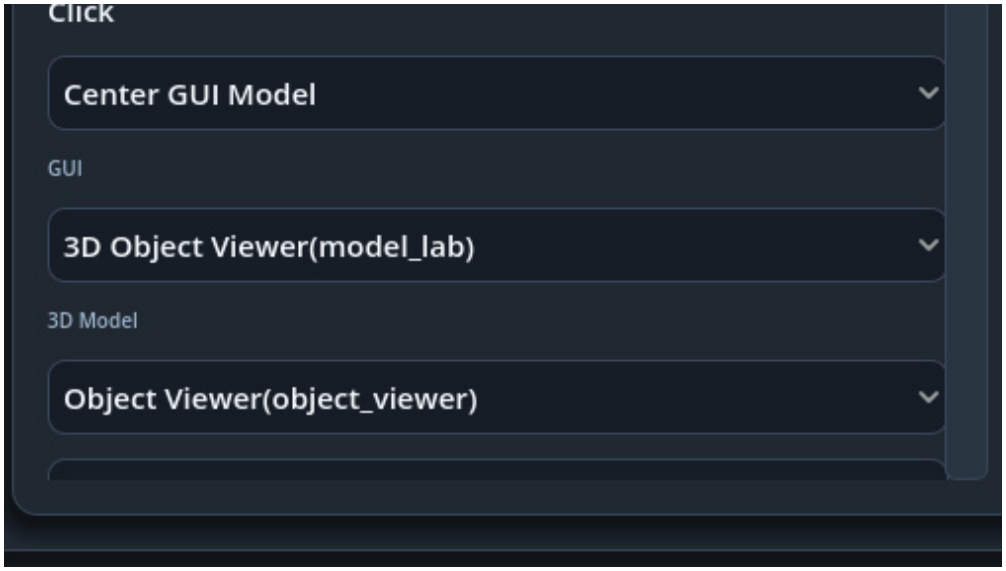


Figure 13. Center GUI Model targets the owning GUI and the 3D Model element.

| Shared action | Fields |
|---|---|
| Center GUI Model | GUI and 3D Model. |
| Play GUI Model Animation | GUI, 3D Model, Animation, Loop and Speed. |
| Change GUI Element Property | For authored viewer fields such as assetId, modelInitialView or modelAnimated. |

These actions are also available in action lists and visual scripting. Use a separate Button for a story action when dragging should remain a harmless inspection gesture. A model’s generic mouse Click event can otherwise fire at the start of an orbit drag.

# Use the model actions from a script

```
extends "res://scripts/runtime/script_base.gd"

func run(_context: Dictionary = {}) -> Variant:
    await CenterGUIModel({
        "guiId": "model_lab",
        "elementId": "object_viewer"
    })
    return true
```

Create and apply this GDScript record, then run it from the intended Button or gameplay interaction. The IDs are the GUI and its element, not the source model asset ID.

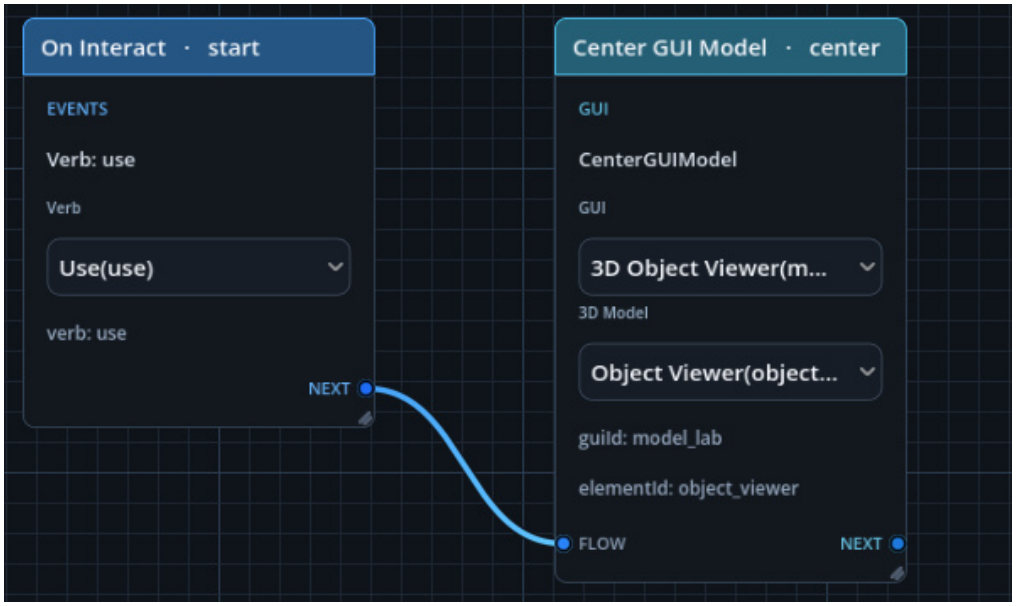


Figure 14. The visual-script node uses the same two target IDs.

For a visual graph, connect the intended entry node to Center GUI Model. For an action list, add the same action and fill in the two selectors. You do not need a custom camera script just to recover the framing.

# 12. Change the clip, view or model

Play a particular embedded clip with the shared action. This example is a complete small GDScript record:

```
extends "res://scripts/runtime/script_base.gd"

func run(_context: Dictionary = {}) -> Variant:
    await PlayGUIModelAnimation({
        "guiId": "animation_lab",
        "elementId": "avatar_viewer",
        "animation": "PCA_OpenHands",
        "loop": true,
        "speed": 1.0
    })
    return true
```

To stop the viewer animation, use Change GUI Element Property with property modelAnimated and value false. To change an authored view, use a property such as modelInitialView with the value left.

```
await ChangeGUIElementProperty({
    "guiId": "model_lab",
    "elementId": "object_viewer",
    "property": "assetId",
    "value": "owl_model"
})
```

The second example belongs inside a run function. It swaps the loaded model asset while keeping the element ID. After swapping, check framing and clip selection: an animation name from the old model may not exist in the new one.

Changing view fields reapplies the starting view and can clear temporary player orbit/pan. Use those changes deliberately, not every frame while the player is inspecting the object.

# 13. When the model viewer looks wrong

| Problem | Check this first |
|---|---|
| Nothing appears. | Choose a model asset in Images → Media Asset. Check the importer produced a usable mesh and the file resolves. |
| It looks tiny or far away. | Use Center & Fit and a camera distance of 0. Check source bounds, attachments and imported scale. |
| It faces the wrong way. | Check Initial View and Rotation X/Y/Z. Imported front orientation varies by model. |
| Dragging does nothing. | Enable Preview or run the game. Check Allow Player Controls and the individual Orbit/Pan/Zoom switches. |
| The view changes do not save after Preview. | Interactive orbit is temporary. Author repeatable values in the typed view fields. |
| There are no animation choices. | Wait for loading, then check the model contains AnimationPlayer clips. Gameplay state labels are separate. |
| It moves out of frame while animating. | Check root motion and gesture size. Use an in-place clip or leave more room. |
| Room lights or fog have no effect. | The GUI viewer uses its own world, camera and lights. |
| The backdrop does not reflect in the model. | Background Image is a flat 2D layer, not a reflection environment. |
| Center does not erase a tilted angle. | It preserves the authored preset and extra rotation. Change those fields explicitly. |

Restart the game preview after a structural asset change so you test a fresh loaded instance. Keep a known working model available for comparison.

# 14. Add a Video element to a GUI

1. Create an Embedded Video GUI with ID video_lab.
2. Add a Video element with ID briefing_video.
3. For the example, use X 32, Y 94, Width 736 and Height 421.
4. In Images, choose the imported video as Media Asset.
5. In Type Settings, set each content margin to 8. Leave Autoplay off while you build the playback Buttons.

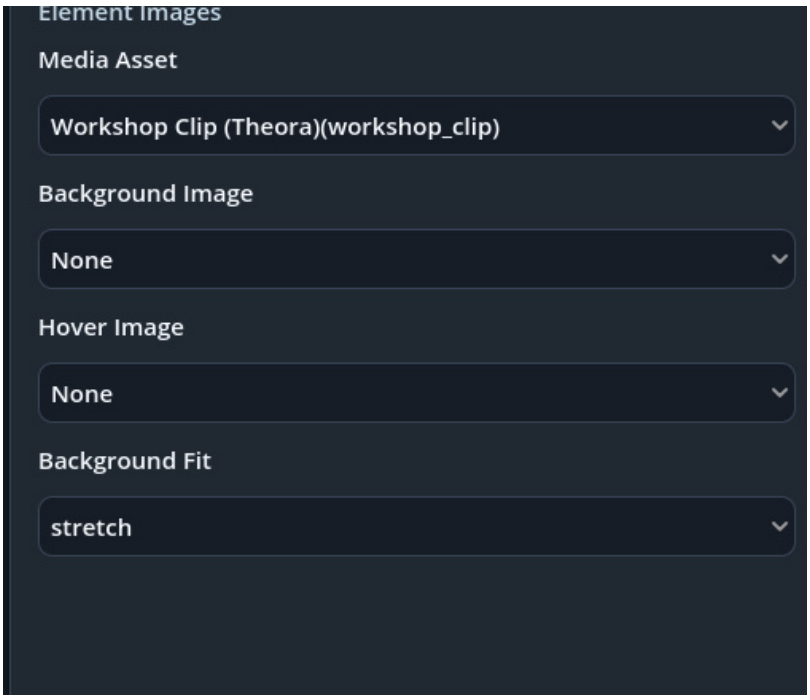


Figure 15. The Video element’s Media Asset selector uses video assets. Background Image is a separate image layer.

The outer rectangle is 736 by 421. The four 8-pixel margins leave a 720 by 405 content rectangle, which matches a 16:9 clip. A correctly shaped content area is important for this version’s embedded player.

Add separate Buttons for Play / Restart, Pause, Resume and Stop. The next chapters explain their scripts; these controls are not generated automatically by the element.

> A valid clip with Autoplay off can remain blank until it is played. Poster Image is not a general pre-play cover in this build.

# 15. Choose a video format that can load

| Format / decoder | What this build provides |
| --- | --- |
| Ogg Theora video, usually .ogv | Uses the native Theora playback path. This is the straightforward shared desktop test format. |
| MP4, MOV, WebM and other listed formats | Use the FFmpeg video decoder when it is present and usable. The file’s codec must also be supported. |
| Bundled FFmpeg platform check | The source enables this bundled decoder path for Windows and Linux on x86_64. Do not assume the same package is active on macOS or another architecture. |

The sample Theora clip and an MP4 comparison both loaded in the desktop environment used for this guide. That does not prove every codec inside an MP4 or every target build will work. Test the files in the package you plan to distribute.

A filename extension is a container hint. Renaming a file to .ogv does not turn it into Theora video, and an Ogg audio file is not a video just because the loader recognizes .ogg as a possible video extension.

1. Import a short known clip first.
2. Assign it as the Video element’s Media Asset.
3. Play it in a running game preview.
4. Test the actual exported game on the intended desktop platform.

> A missing decoder can lead to a poster or Load Failure Text. That fallback is useful feedback, but it is not successful movie playback.

# 16. Understand the playback settings

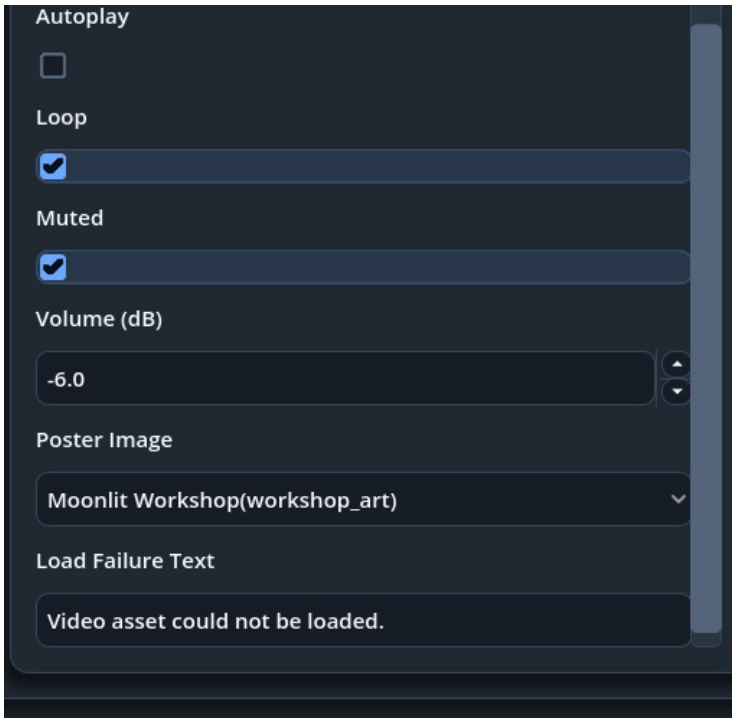


Figure 16. Video Playback contains Autoplay, Loop, Muted, Volume, Poster Image and Load Failure Text.

| Field | Meaning |
|---|---|
| Autoplay | Requests playback when the video definition is applied. It is not a dedicated “GUI opened” event. |
| Loop | The authored repeat behavior. Turn it off for a finite completion exercise. |
| Muted | Silences this embedded player using a very low player volume. |
| Volume (dB) | Audio level for this player when not muted. |
| Poster Image | Load-failure image fallback in this build. |
| Load Failure Text | Message used when no stream and no usable poster can be shown. |

For a controlled briefing, leave Autoplay off and call the play script after opening the GUI. For a repeating decorative clip, test Autoplay and Loop together in the actual opening flow.

> Changing a Video property through Change GUI Element Property reapplies its definition and reloads the stream in this version. Apply configuration before starting playback rather than treating every property edit as a harmless live adjustment.

# 17. Set the audio deliberately

Muted and Volume (dB) belong to this Video element. They do not automatically pause scene music, stop other sounds or change the whole game’s audio preference.

| Volume setting | Practical use |
| --- | --- |
| 0 dB | No attenuation at this player’s volume control. |
| A negative value, such as -6 or -12 | Reduce this clip’s level. Judge the result with an audible clip. |
| Muted on | Silence this player even if its saved Volume field is higher. |
| A positive value | Raises the level. Check the resulting sound rather than assuming louder is better. |

The test-pattern clip shown in this guide is silent. Use a clip with speech or a clear sound to check audio. If music covers a spoken briefing, coordinate the music through its own actions or settings.

For a live player-level adjustment from a script, obtain the VideoStreamPlayer as shown in the playback chapters, then change its volume_db. This changes that player directly without reloading the element definition.

```
player.volume_db = -12.0
```

The line above assumes player is a valid VideoStreamPlayer already obtained by your script. It is a decibel value, not a 0-to-100 percentage.

> If you change the authored muted property through Change GUI Element Property, expect a stream reload. A paused clip or one with Autoplay off may need an explicit playback command afterward.

# 18. Lay out the content rectangle

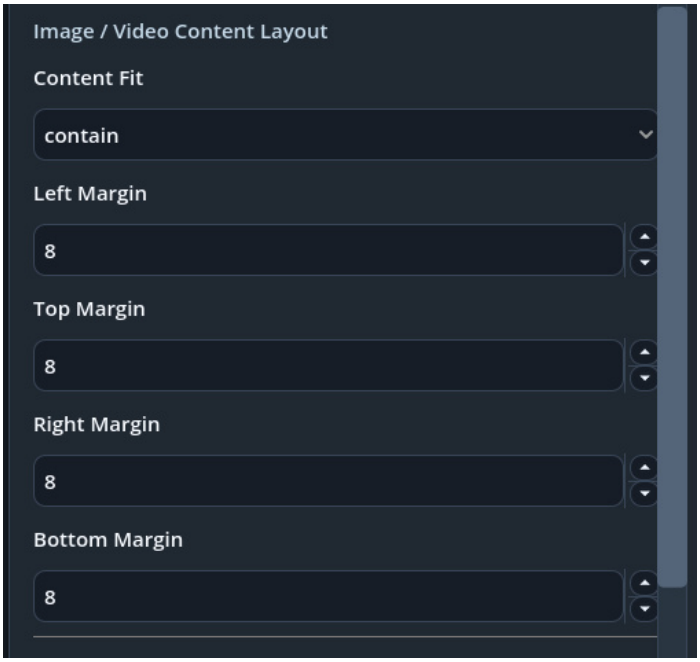


Figure 17. The four margins inset the video content. The section is shared with Image controls.

The visible content rectangle is the element’s Width and Height minus its left/right and top/bottom margins. Leave room around it for the frame and the control Buttons.

| Source shape | Example content rectangle |
|---|---|
| 16:9 | 720 by 405, or 640 by 360. |
| 4:3 | 640 by 480. |
| Square | 400 by 400. |

To put a 16:9 clip inside a larger square panel without distortion, use a wide Video element inside that panel and leave the unused area to the surrounding GUI. Do not stretch the element to a square and assume it will always letterbox automatically.

> Animation Frame Asset IDs and Image Animation FPS may be visible in the shared layout section. They are Image settings; they do not set the Video element’s decoded frame rate or create a video animation sequence.

# Content Fit: check the current limitation

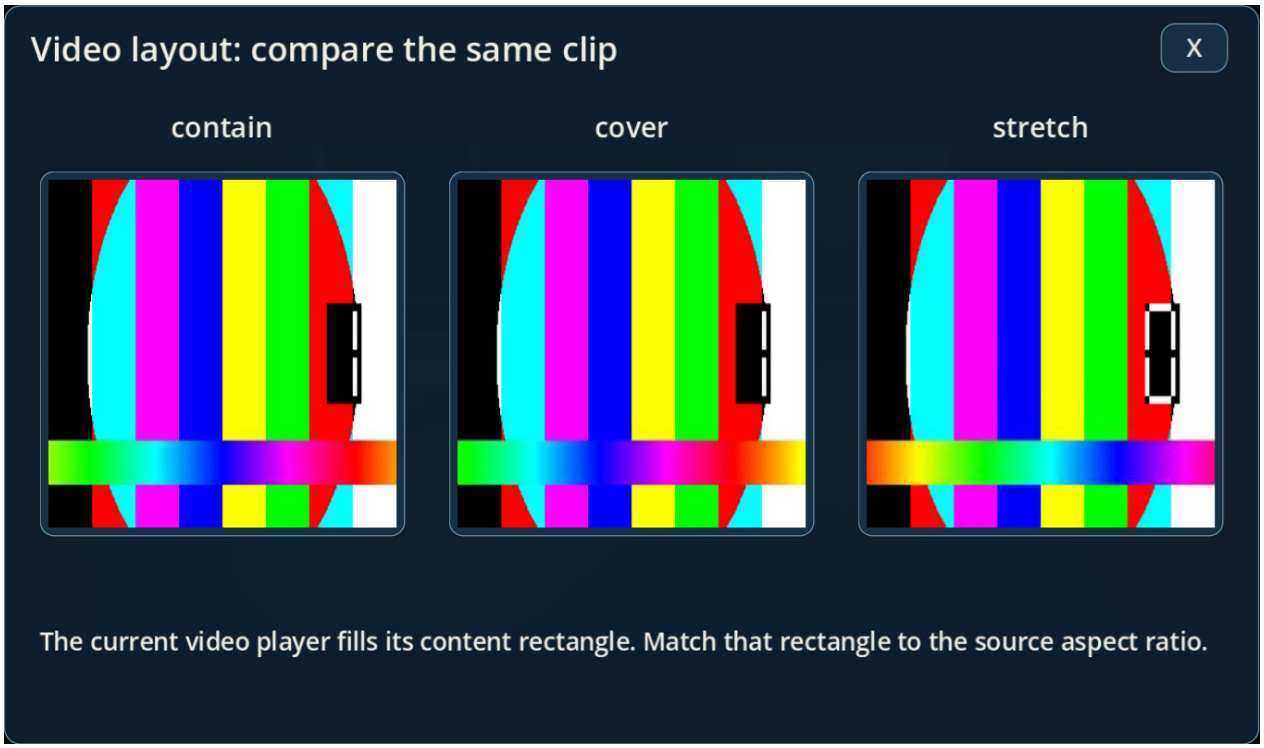


Figure 18. The same clip in square content rectangles. In this build the decoded video fills its rectangle across these settings.

The current GUI video implementation does not apply the full Image element’s contain/cover/stretch layout logic to decoded movie frames. It gives the player the authored content rectangle. Content Fit does control how a fallback poster texture is fitted.

For predictable movie proportions, match the actual content rectangle to the source aspect ratio. If a 16:9 picture is forced into a square rectangle, changing this field alone is not a dependable way to recover the original proportions in 0.86.3.4.

The frame and background can use rounded artwork. The native player is clipped by the element rectangle; do not assume a rounded border also supplies a rounded movie mask.

> Check these details after an engine update. The fields may remain in the same place even when the implementation changes, so compare the actual playback rather than relying only on the setting name.

# 19. Give a failed clip a useful fallback




Figure 19. The left Video has a poster; the right has readable Load Failure Text. Neither has a usable stream assigned.

1. Select a Video element and choose a clear Poster Image.
2. Enter a short Load Failure Text message as well.
3. Test with no video assigned, then with a valid clip.
4. Keep a working Close or Back control available in both cases.

With no stream, the poster is shown if it can load. If there is no usable poster, the text message is shown instead. When a stream loads successfully, the poster is hidden even when Autoplay is off.

Use a separate Image element for a deliberate title card before playback. Hide that Image when your Play Button starts the movie. The Poster Image field alone does not provide that opening sequence.

> Use a normal Label for dynamic or translated surrounding instructions. Do not assume the Video element’s Load Failure Text runs the same wildcard-expansion path as a normal gameplay Label.

# 20. Build a Play / Restart Button

The embedded Video element does not have dedicated shared Play GUI Video / Pause GUI Video action types in this build. A small GDScript wrapper can control its actual player. Create a script record with ID play_video:

```
extends "res://scripts/runtime/script_base.gd"

func run(_context: Dictionary = {}) -> Variant:
    var player := GetGUIElementProperty(
        "video_lab", "briefing_video", "video_player", null
    ) as VideoStreamPlayer
    if player == null or player.stream == null:
        return false
    player.paused = false
    player.play()
    return true
```

1. Apply Changes to the script.
2. Add a Button with the caption Play / Restart.
3. In Events, set Click to run-visual-script and Click Target to play_video.
4. Start a running game preview, open the GUI and press the Button.

The behavior name run-visual-script can run this GDScript record too. You can also use the shared script-running action that offers the project’s script records.

player.play() starts playback from the beginning. Resuming a paused movie uses a different operation, explained next.

# 21. Pause, resume and stop

Create a separate Pause script record. It obtains the same embedded player and changes its paused flag:

```
extends "res://scripts/runtime/script_base.gd"

func run(_context: Dictionary = {}) -> Variant:
    var player := GetGUIElementProperty(
        "video_lab", "briefing_video", "video_player", null
    ) as VideoStreamPlayer
    if player == null or player.stream == null:
        return false
    player.paused = true
    return true
```

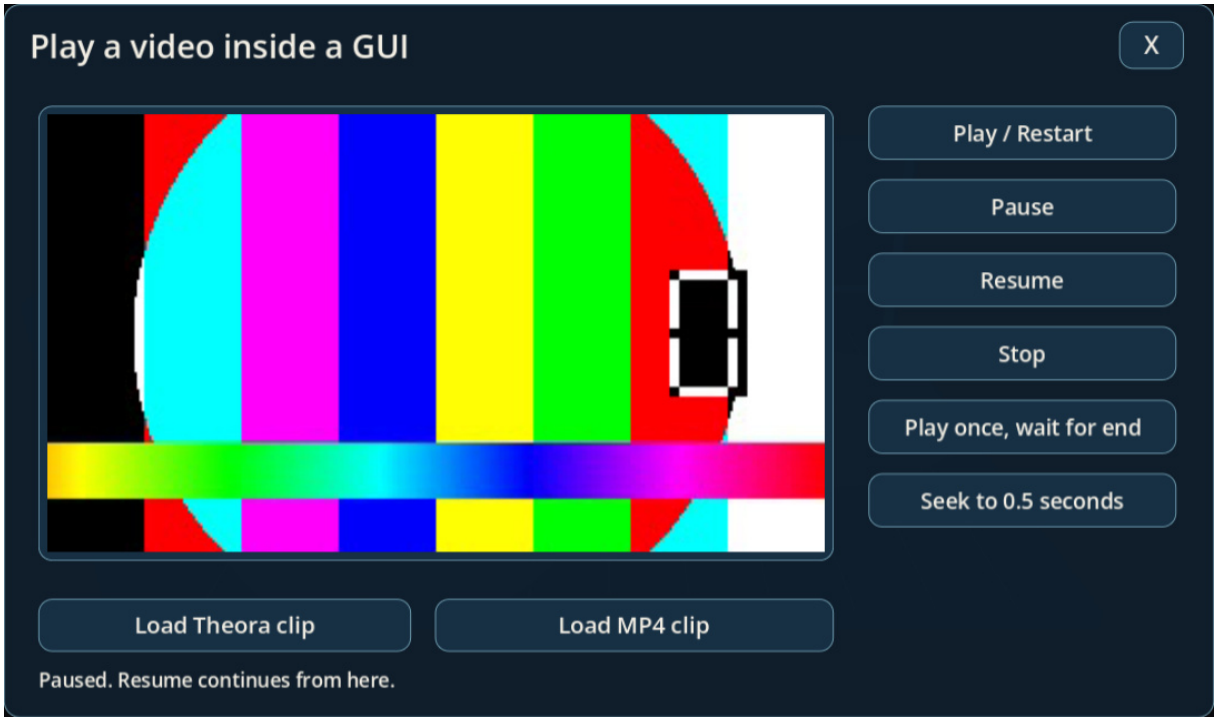


Figure 20. The paused frame stays visible. Resume should continue the existing playback rather than restart it.

Assign this record to the Pause Button through run-visual-script and its script ID. A paused player can still report is_playing() as true; paused and stopped are different states.

# Resume continues; Stop ends the current play

For Resume, copy the Pause script and change only the final assignment:

```
player.paused = false
```

Do not add player.play() to a simple Resume command; that restarts the clip. Resume is meaningful after a pause, while Play / Restart is the appropriate command after a stop.

Here is a complete Stop script:

```
extends "res://scripts/runtime/script_base.gd"

func run(_context: Dictionary = {}) -> Variant:
    var player := GetGUIElementProperty(
        "video_lab", "briefing_video", "video_player", null
    ) as VideoStreamPlayer
    if player == null:
        return false
    player.stop()
    player.paused = false
    return true
```

| Button | Intended result |
| --- | --- |
| Play / Restart | Start again from the beginning. |
| Pause | Keep the current playback suspended. |
| Resume | Clear the pause without restarting. |
| Stop | End the current playback; use Play / Restart to begin again. |

Do not depend on Stop preserving a particular still frame or revealing the fallback poster. Use a separate Image if your design requires a specific stopped display.

# 22. Seek to a position in the clip

The embedded player exposes stream_position in seconds. This example requests a seek to half a second:

```
extends "res://scripts/runtime/script_base.gd"

func run(_context: Dictionary = {}) -> Variant:
    var player := GetGUIElementProperty(
        "video_lab", "briefing_video", "video_player", null
    ) as VideoStreamPlayer
    if player == null or player.stream == null:
        return false
    player.stream_position = 0.5
    return true
```

The sample Theora and MP4 clips accepted the 0.5-second seek test in this build. Seeking still depends on the stream implementation and the file. Check the actual frame, position and behavior on the target platform.

1. Play the clip, pause it and note the current frame.
2. Run the seek script, using a position inside the known clip duration.
3. Check the displayed frame and the player’s reported position.
4. Resume and confirm that playback continues from the intended point.

Use seconds, not frame numbers or a percentage. Stay within the clip’s duration and test backward as well as forward seeking before building a scrubber around it.

> The general Seek Video action is for the movie manager’s player. It does not select briefing_video inside this GUI. Use this wrapper when you intend to control this particular embedded element.

# 23. React when a finite clip finishes

The typed GUI Events tab has pointer events, not a dedicated On Video Finished action slot. A script can wait for the embedded player’s natural finished signal. Create a Boolean variable clip_done, starting false:

```
extends "res://scripts/runtime/script_base.gd"

func run(_context: Dictionary = {}) -> Variant:
    var player := GetGUIElementProperty(
        "video_lab", "briefing_video", "video_player", null
    ) as VideoStreamPlayer
    if player == null or player.stream == null:
        return false
    await SetVariable({"name": "clip_done", "value": false})
    player.loop = false
    player.paused = false
    player.play()
    await player.finished
    await SetVariable({"name": "clip_done", "value": true})
    return true
```

Run this from a Play once Button and let the short clip reach its end. You can then place the intended story action after the wait, or test clip_done elsewhere.

Loop is deliberately turned off so the exercise has a natural end. A stop or close is not the same event as reaching that end. If the player can cancel, give the cancel path its own cleanup and result rather than assuming finished ran.

> Do not start several copies of the waiting script for one playback. In a finished interface, guard the launch or disable repeated activation while that sequence is active.

# 24. Stop the movie when closing the GUI

Hiding the GUI did not stop the embedded player in this guide’s runtime test. Use an explicit close script when the movie and its audio should stop as the window closes:

```
extends "res://scripts/runtime/script_base.gd"

func run(_context: Dictionary = {}) -> Variant:
    var player := GetGUIElementProperty(
        "video_lab", "briefing_video", "video_player", null
    ) as VideoStreamPlayer
    if player != null:
        player.stop()
        player.paused = false
    await HideGUI({"guiId": "video_lab"})
    await ShowGUI({"guiId": "media_hud"})
    return true
```

1. Add your own Close Button, or assign this script to the existing GUI close element.
2. Use run-visual-script and the script’s ID as Click Target.
3. Play a clip and close the GUI while it is running.
4. Reopen it and check the behavior you intended: explicit restart, deliberate resume or an idle screen.

The null check still allows the GUI to close when a player or stream is unavailable. A broken video should not trap the user inside the window.

media_hud is the example’s small opening menu. Replace that ID with your own HUD or omit the ShowGUI line if your game does not need it.

# 25. Change the clip or its settings

```
extends "res://scripts/runtime/script_base.gd"

func run(_context: Dictionary = {}) -> Variant:
    await ChangeGUIElementProperty({
        "guiId": "video_lab",
        "elementId": "briefing_video",
        "property": "assetId",
        "value": "workshop_clip"
    })
    var player := GetGUIElementProperty(
        "video_lab", "briefing_video", "video_player", null
    ) as VideoStreamPlayer
    if player == null or player.stream == null:
        return false
    player.paused = false
    player.play()
    return true
```

Changing assetId applies the new definition and loads the new stream. Obtain the player and check its stream before issuing the next playback command.

Other Video property changes also reapply the definition in this build. That can stop the current clip, restore authored Loop and Muted values and request Autoplay again. Do not use repeated property edits as a way to animate a movie’s volume or progress every frame.

| Change | Useful approach |
| --- | --- |
| A different clip | Change assetId, check the new stream, then play deliberately. |
| Live player volume | Use the obtained player’s volume_db. |
| Pause / resume | Use the player’s paused flag. |
| Background image | Set up the appearance before playback when possible. |

# 26. Start playback when the window opens

Autoplay is a construction/configuration request. It is not a new action that runs each time you call Show GUI. Hidden or already constructed elements make an explicit opening sequence easier to reason about.

```
extends "res://scripts/runtime/script_base.gd"

func run(_context: Dictionary = {}) -> Variant:
    await ShowGUI({"guiId": "video_lab"})
    var player := GetGUIElementProperty(
        "video_lab", "briefing_video", "video_player", null
    ) as VideoStreamPlayer
    if player == null or player.stream == null:
        return false
    player.paused = false
    player.play()
    return true
```

Assign this record to the Button or story event that opens the briefing. Keep Autoplay off when this script is responsible for starting it. Decide separately whether the opening HUD should be hidden.

If a particular GUI uses an opening transition, make sure the start time matches the intended presentation. Starting the movie immediately can advance frames while a long transition is still revealing the window.

1. Open the GUI for the first time and check the first visible frame.
2. Close it using the cleanup script.
3. Open it again and confirm the movie restarts because this script calls play.
4. Test a missing stream too; the GUI should still offer its poster/message and a way back.

For deliberate resume-on-open behavior, preserve the paused player and clear paused instead. That is a different design from stopping on close and restarting on open.

# 27. Run the controls through action lists

The model commands already exist as shared actions. Embedded video commands in this guide are small script records. Both can take part in the normal GUI event and action-list workflow.

| Task | How to arrange it |
| --- | --- |
| Center an inspection model | Add Center GUI Model directly to the Button event or action list. |
| Change its embedded animation | Add Play GUI Model Animation with its clip, Loop and Speed. |
| Start a GUI video | Run the prepared Play script, targeting the embedded element. |
| Close a video screen | Run a close wrapper that stops the player before hiding the GUI. |
| Use a visual graph | Connect the intended entry to the shared model action, or use the shared script-running node for the prepared video wrapper. |

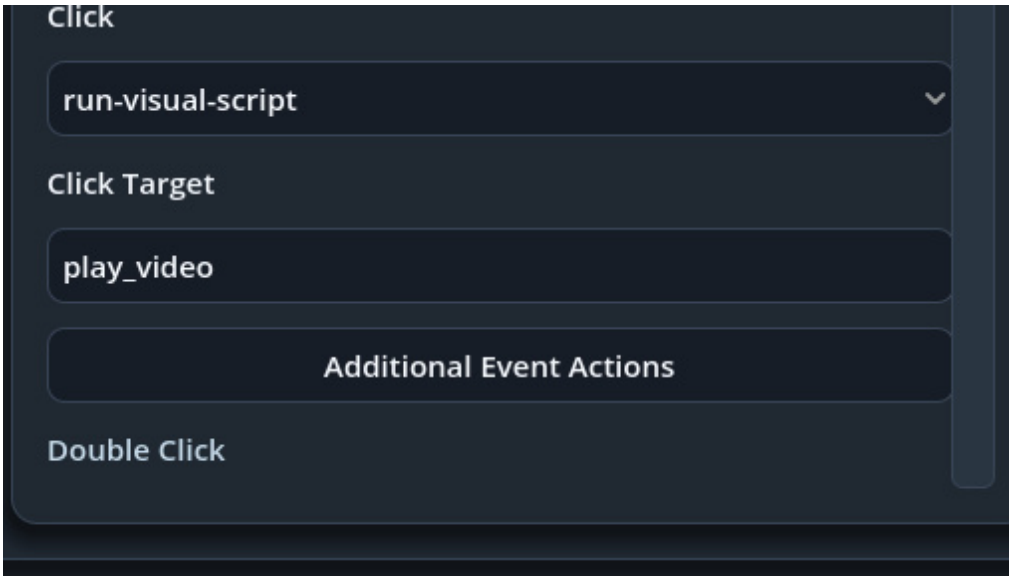


Figure 21. The Video Play Button runs a script record. Its target is the wrapper script, which then finds the embedded player.

Keep the wrapper script IDs different from the video asset ID. A Button that runs play_video is choosing code; that code targets briefing_video, which loads workshop_clip.

> The shared Play Video action can be useful elsewhere in your game, but using it here does not mean it is controlling the Video element you placed in this GUI.

# 28. Add captions, labels and a title card

Use ordinary Labels for the model’s name, inspection notes, video instructions and status. Place them beside the display, or layer them deliberately over it with a higher Z Index.

| What you want | Useful setup |
| --- | --- |
| A model title | A Label that names the displayed object. Update it when swapping assets. |
| Live status text | A String variable and a normal Label such as $$media_status. |
| A video title card before playback | A separate Image above the movie. Hide it when the Play script begins. |
| Captions visible throughout a clip | A normal Label, or captions already included in the video picture. |
| Timed subtitle cues | A deliberate timing script or the appropriate movie workflow; do not assume the GUI element adds them automatically. |

The general movie manager has its own subtitle handling. The embedded GUI Video element does not automatically create that same subtitle overlay from the video asset’s cue data.

A normal Label can use gameplay text wildcards, such as @PlayerName or $$media_status. An asset ID field is a literal reference, not a place to type a Label wildcard.

Keep labels and title-card images out of the model’s drag area when they do not need to cover it. A decorative overlay that intercepts pointer input can make an otherwise working viewer feel unresponsive.

> A click-to-start title card is a small interface sequence: show the Image, start the player, then hide the Image. The Poster Image field alone is not that sequence.

# 29. Use Preview for the right checks

| Check | Use |
| --- | --- |
| Model appearance, asset choice and initial framing | The designer, then a running game preview. |
| Local model orbit/pan/zoom | Designer Preview with player controls enabled. |
| Model action Buttons and story scripts | A running game. |
| Video frames in the designer | Preview mode with the intended autoplay setup; ordinary editing does not start story events. |
| Video wrapper Buttons | A running game preview. |
| Codecs and playback in the final package | The exported game on the target machine. |

Turn Preview off before selecting and resizing elements. When testing an initial model view, restart or reapply the authored setup so you are not judging a temporary orbit left over from a previous local test.

In designer editing, the Video preview is constructed without starting autoplay. With Preview enabled, an enabled Video whose authored Autoplay is on can be started for local preview. That is still not execution of your Play Button’s script.

Use a short known clip and a known model to isolate problems. A missing codec, a missing model dependency and an unassigned Button event can all look like “nothing happened,” but they require different fixes.

> The validator in this release may report that model3D or video uses generic preview rendering. These elements were rendered by their dedicated viewers in this guide. Read the exact warning alongside the actual preview; it does not by itself prove the asset is unsupported.

# 30. Keep playback and resource use tidy

A GUI model viewer renders a real model in its own 3D viewport. A Video element decodes a real stream. Keep the number and size of these live displays sensible for the hardware your players use.

| Choice | Practical effect |
|---|---|
| A few appropriately sized model viewers | Easier to test and less demanding than a page full of large animated models. |
| A model prepared for display | Avoids unnecessary meshes, very large textures and movements that need extreme framing. |
| A clip sized for its display | A small monitor inside a GUI rarely needs a very large source movie. |
| Explicit video close cleanup | Stops unwanted playback instead of assuming hidden means stopped. |
| Separate saved data and temporary view state | Keeps the game’s meaning independent from the last drag or decoded frame. |

Save Project keeps the authored GUI and scripts. A temporary model orbit or current video frame is not automatically the permanent game record. Store meaningful story values, then reconstruct the intended display when opening or loading.

For video resume across a save/load, store the clip’s stable asset ID, a valid time and the intended playback state, then test restoration with that decoder. A stream reload or format change can invalidate an assumption about the current player.

> Test the exported game as well as the editor. Use the target computer’s actual renderer, decoder package, display scale and input device before treating the interface as finished.

# 31. Assemble a complete inspection screen

Here is a practical order for building a screen that combines the two elements without trying to solve everything at once.

1. Create a trigger-controlled GUI with a clear title and a working Close Button.
2. Add the 3D Model element, choose a known model and use automatic fitting.
3. Set its initial orientation and background. Test the model before enabling animation.
4. Enable the player controls you want and add a Center GUI Model Button.
5. Choose an embedded clip if the model should animate. Verify it loops and stays inside the frame.
6. Add a Video element with a content rectangle matching the source ratio. Keep Autoplay off during the first setup.
7. Create Play, Pause, Resume and Stop wrappers. Assign them to ordinary Buttons and test each one.
8. Add poster/message fallbacks and a close wrapper that stops playback.
9. Add the intended opening sequence, labels and any natural-completion logic.
10. Save, restart, test missing content and check the exported game on the intended platform.

For an adventure, the model might be an artifact the player is inspecting while the video shows a short explanation or remembered scene. The GUI supplies the presentation; your actions and story variables decide what the player learns or unlocks.

> Keep “inspect” and “use” deliberate. A drag that turns an artifact should not accidentally award an item or start a conversation just because it began with a left press.

# 32. When embedded video does not work

| Problem | Check this first |
|---|---|
| The clip is blank before Play. | A valid stream with Autoplay off may be idle. Poster Image is a load-failure fallback, not a general opening card. |
| The poster appears instead of the clip. | Check the video asset, resolved file and decoder availability. A poster is not successful video playback. |
| Only the failure message appears. | Neither a stream nor a usable poster loaded. Keep the Close control working and fix the content path or codec. |
| Pause works, but Resume restarts. | Resume should clear paused. Calling play starts again. |
| A hidden clip still runs or makes sound. | Stop or pause it explicitly before hiding the GUI. |
| Changing volume or Loop restarts the movie. | Changing the element definition reloads the stream. Use the live player where a continuous adjustment is intended. |
| Content Fit does not letterbox the movie. | Match the content rectangle to the clip’s source ratio in this build. Poster fitting is a separate path. |
| A fullscreen movie opens instead. | The general Play Video action uses the movie manager. Use the embedded-player wrapper for this GUI element. |
| My completion action never runs. | Disable looping, let the clip reach its natural end and distinguish cancellation from completion. |
| Seeking behaves differently in another build. | Test that decoder and file on the target platform. The requested position is in seconds. |

# 33. A reference for the model viewer

| Property / command | Use |
|---|---|
| assetId | The model asset loaded by this GUI element. |
| modelInitialView | front, back, left, right, side, top or bottom. |
| modelStartDistance | 0 for automatic framing; a positive starting distance for manual control. |
| modelRotationX / Y / Z | Additional authored rotation in degrees. |
| modelPanX / modelPanY | Authored viewing-target offsets. |
| modelFov | Viewer camera field of view. |
| modelInteractive | Master player-control switch. |
| modelMouseRotate / Pan / Resize | Orbit, pan and zoom permissions. |
| modelAnimated / modelAnimation | Playback enable and embedded clip name. |
| modelAnimationLoop / Speed | Clip repeat and speed. |
| CenterGUIModel | Shared action: guiId, elementId. |
| PlayGUIModelAnimation | Shared action: guiId, elementId, animation, loop, speed. |

The designer uses readable labels for these fields. Scripts and property actions use the exact property keys. For example, the Zoom Model switch corresponds to modelMouseResize.

Temporary orbit, pan and zoom are viewer interaction state. They are not the same values as the authored Rotation or View Offset fields.

> When swapping a model, verify its clip names and framing again. The element ID can remain stable while the asset behind it changes.

# A reference for the embedded player

| Element property | Meaning |
|---|---|
| assetId | The video asset to load. |
| autoplay / loop / muted | Authored playback configuration. |
| volumeDb | Authored player level in decibels. |
| posterAssetId / emptyText | Load-failure image and text. |
| contentMarginLeft / Top / Right / Bottom | Insets that define the content rectangle. |
| contentFit | Poster fitting; see the decoded-video limitation in Chapter 18. |

## Live VideoStreamPlayer controls

| Read or command | Purpose |
|---|---|
| GetGUIElementProperty(..., "video_player", null) | Obtain this element’s live player in GDScript. |
| player.play() | Start again. |
| player.paused = true / false | Pause / resume the existing playback. |
| player.stop() | Stop the current playback. |
| player.stream_position | Current/requested position in seconds. |
| player.volume_db | Live player audio level. |
| player.finished | Natural completion signal for the finite-playback exercise. |

Screenshots and runtime checks: PCA 3D Adventure Studio 0.86.3.4. Models and background artwork shown are from existing engine project assets. The sample video is a short test clip. No practice files are included.