PCA 3D ADVENTURE STUDIO

GDScript

Programming and Godot

A practical tutorial and complete PCA scripting reference

Build adventure logic, work with the designers, and use Godot nodes, signals, resources and custom classes.

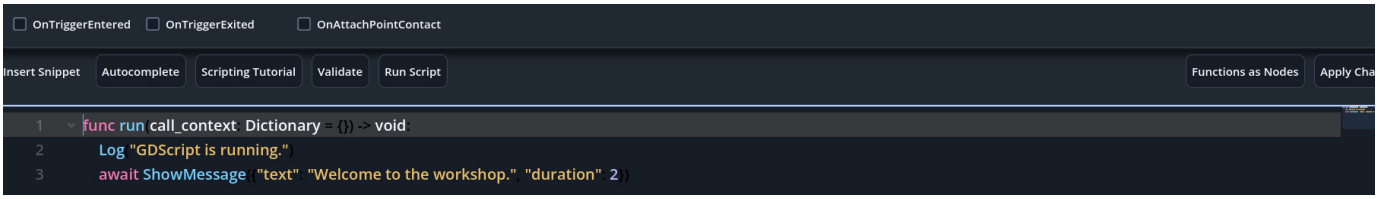

Figure 1. Native GDScript in the PCA editor.

PCA 3D 0.86.3.4 • Godot 4.7.2

Windows · macOS · Linux

About the examples

This book follows the same workshop names as the PCA Script guide. You do not need a practice project. Replace the example IDs with records in your own test game.

Example ID	Meaning
lab / archive	Two rooms
ada / elias	Player and NPC definitions
archive_door / key_pickup / welcome_trigger	A placed door, pickup and trigger
key / chime / wide	Inventory item, audio asset and room camera
hud / name_entry	Gameplay interface and input panel
utility_gd / reward_gd	Reusable native function records

Tutorial blocks are complete native script records unless the text says they are a fragment or a file for an editable Godot project. No extends line is needed for normal PCA records; PCA supplies its script base.

The language examples were compiled using the installed runtime. Pure calculations, containers, signals, file operations, timers and node examples were exercised where applicable. The input panel, native UI, native cube and cross-language routes were tested in the running game.

> A compiled call pattern is not a complete authored game. Gameplay functions still need valid records, assets, navigation and the correct active room. The reference marks runtime/configuration boundaries rather than assuming that every field is live.

Contents

About the examples 2

1. GDScript inside PCA 3D 9

2. Create and run your first GDScript 10

3. Indentation, lines and comments 11

4. Quotes, escapes and multiline text 12

5. Values, types and conversions 13

6. Variables, constants and enums 14

7. Operators and arithmetic 15

8. If, elif, else and conditional expressions 16

9. Match and named states 17

10. For loops, while loops and range 18

11. Arrays and typed arrays 19

12. Dictionaries and defaults 20

13. Functions, defaults and return values 21

14. Strings, formatting and regular expressions 22

15. Math, vectors and random choices 23

16. Read player input from an authored GUI 24

17. Waits, timers and elapsed time 26

18. Native file reading, writing and appending 27

19. JSON, configuration and file operations 28

20. Dates and calendar values 29

21. Errors, assertions and recovery 30

22. Your run function and its caller 31

23. Self, this and the player 32

24. Find records through registries 33

25. Read and change designer properties 34

26. Gameplay actions and await 35

27. Build a one-time key-and-door interaction 36

28. Subscribe to PCA events 37

29. Custom events and Godot signals 38

30. Speech, portraits, lip-sync and typewriter text 39

31. Cameras, scene changes and effects 40

32. Save state, instance state and project settings 41

33. How to access Godot functionality 42

34. Keep the PCA base when extending a script 43

35. Startup order and script lifetime 44

36. Nodes, parents, paths and ownership 45

37. Godot signals and Callables 46

38. A native Godot button inside a PCA game 47

39. Processing and a frame-based countdown 48

40. Godot input and PCA controls 49

41. Tweens and temporary visual changes 50

42. Add a native 3D object to the room 51

43. Reach an existing model without guessing a node path 52

44. Materials, shaders, animation and resource sharing 53

45. Physics, areas and ray queries 54

46. Resources, load and preload 55

47. Your own classes and helper objects 56

48. Inheritance, annotations and editor code 57

49. External files, libraries and packaging 58

50. Include a PCA script library 59

51. Call PCA Script, Lua, BASIC and visual graphs 60

52. Expose a native function as a visual node 61

53. Debugging in PCA and in Godot 63

54. Build a clean native extension 64

55. Test the complete workflow 65

PCA helper functions: the full native reference 66

Execution, calls and project access 66

Guarded calls and loop helpers 71

Game queries and interfaces 73

Time, files and text 78

Math, randomness and data conversion 83

Gameplay actions: all 206 native wrappers 88

Gameplay: Variables, rewards and reusable logic 89

Gameplay: Inventory and two-object interactions 94

Gameplay: Speech, messages, voice and dialogue 98

Gameplay: Characters, movement and animation 111

Gameplay: Objects, descriptions and scene life 126

Gameplay: Cameras, transitions and screen effects 136

Gameplay: Room changes, 2D controls and environmental state 148

Gameplay: Continuous object effects and attachments 161

Gameplay: GUIs, element state and data-driven widgets 174

Gameplay: Control profiles, verbs and pointer behavior 189

Gameplay: Audio, video, FMV and recorded gameplay 198

Gameplay: Quests and objectives 210

Gameplay: Saving, language and project settings 218

Gameplay: Cutscene control and timeline-specific actions 223

Registries, proxies and native nodes 224

Registry and script-call methods 225

Record proxy methods: the complete authored surface 226

Nested properties and resource ownership 229

Designer data: reading the property reference 230

Designer: Character definitions 231

Designer: Objects and character instances 237

Designer: Triggers 246

Designer: Rooms 253

Designer: Inventory definitions 258

Designer: GUIs 260

Designer: Dialogue graphs 263

Designer: Quests 264

Designer: Objectives 265

Designer: Assets 266

Designer: Cameras 267

Designer: Spawn points 269

Designer: Cursors 270

Designer: Verbs 271

Designer: Fonts 272

Designer: PCA Script records 273

Designer: Lua script records 274

Designer: BASIC script records 275

Designer: Visual script records 276

Variables, native scripts, project data and cutscenes 277

Character-property action keys 278

Game setting keys 279

GUI element property keys 283

Whole-GUI property keys 287

Object-property action keys 289

Project property paths 290

Quest-objective action keys 292

Quest-property action keys 293

Quest-stage action keys 294

PCA events: all shared hooks 295

Godot global functions: the checked runtime 302

GDScript-specific globals 303

Shared global functions 305

Godot class map and API lookup 320

Godot resources in an exported PCA project 326

Official Godot references 327

GDScript quick reference 328

PCA Script to GDScript: important differences 329

Troubleshooting native scripts 330

Reference basis and practical limits 331

Alphabetical API and event index 332

1. GDScript inside PCA 3D

GDScript is Godot’s own programming language. In PCA 3D, you can use it both for ordinary adventure logic and for work that needs Godot classes directly. A short script can unlock a door; a larger one can create nodes, connect signals, load resources or manage a custom interface.

This guide uses PCA 3D Adventure Studio 0.86.3.4 running Godot 4.7.2. It applies on Windows, macOS and Linux. The editor screenshots are actual captures. The source examples use Godot 4 syntax, which differs from older Godot 3 tutorials.

There are two related APIs to learn. GDScript supplies the language and Godot classes. PCA’s script base adds record registries, gameplay actions and helpers. Function names are case-sensitive: Godot’s `print()` and PCA’s `Log()` are different functions.

Goal	Where to begin
Learn the language	Chapters 2-21
Write adventure logic	Chapters 22-32
Use Godot functionality	Chapters 33-49
Share functions and distribute a game	Chapters 50-55
Look up a PCA action, helper or record	The reference sections and alphabetical index

> The book covers every normal PCA gameplay wrapper and public adventure helper in this build. Godot itself has a much larger class library. The Godot chapters teach the access patterns and common classes, with links to the official class reference for the rest.

2. Create and run your first GDScript

1. Open a project. In the Adventure Database, add a native Script record. Select it to open the GDScript workspace.
2. Use an ID such as `hello_gd` and a readable name such as First GDScript. Keep script IDs unique across languages.
3. Enter the source below. Click Validate, then Apply Changes. Save the project.
4. Start a Test Scene or Play session. Use Run Script to execute the record against that runtime. A gameplay call needs a running game.

```
func run(call_context: Dictionary = {}) -> void:
    Log("GDScript is running.")
    await ShowMessage({"text": "Welcome to the workshop.", "duration": 2})
```

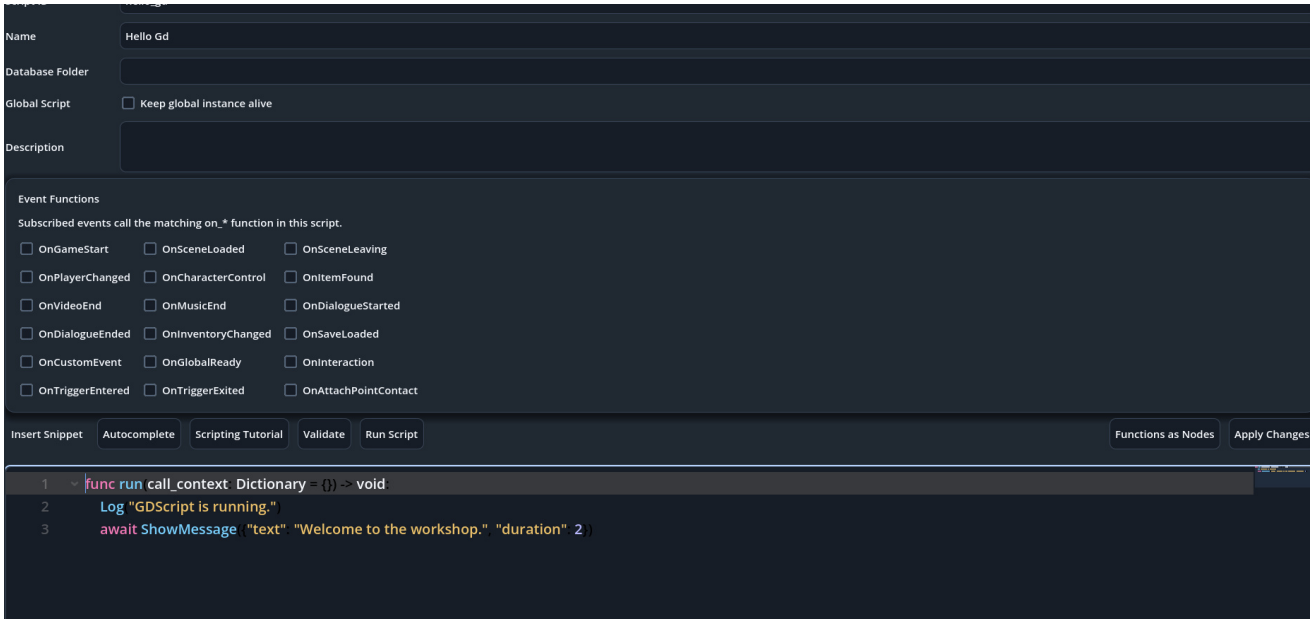

Figure 2. The native GDScript workspace shows source, event subscriptions, the global-instance option and the script toolbar.

You can omit an extends line in a PCA script record. PCA then adds its PCAScriptBase, which inherits Godot’s Node class. You still have GDScript and Godot functionality, with the PCA helpers available as inherited methods.

3. Indentation, lines and comments

```
# A whole-line comment.
func run() -> int:
    var points: int = 5  # A comment after an instruction.
    if points > 0:
        print("There are points to award.")
    return points
```

A colon opens a function, branch, loop or class block. Indentation tells Godot which lines belong to that block. There is no `end` keyword. Use one consistent indentation style; the editor's Tab key is a sensible default.

`#` begins a comment. `##` is used for documentation comments in Godot's documentation workflow. `//` is not a GDScript comment marker. Triple-quoted text is a string, not a separate block-comment system.

```
func run() -> void:
    await ShowMessage({
        "text": "The workshop is quiet.",
        "duration": 2,
        "blockInput": false
    })
```

Unlike PCA Script, GDScript allows a call, array or dictionary to continue across lines inside brackets. Use this to keep options readable. A line with only `pass` is a valid empty function body while you are building the rest.

4. Quotes, escapes and multiline text

```
func run() -> Array:
    var line_a: String = "It's locked."
    var line_b: String = 'The label says "Clockmaker".'
    var line_c: String = "He said \"come in\"."
    var two_lines: String = "First line\nSecond line"
    return [line_a, line_b, line_c, two_lines]
```

In source	Result
`\n`	New line
`\t`	Tab
`\r`	Carriage return
`\\`	Backslash
`\"`	Double quote
`\'`	Single quote

Single and double quotes both create String values. Triple quotes can hold multiline text. For an ID with spaces or punctuation, use a quoted string through a registry's Get method: `Objects.Get("archive-door")`. GDScript does not use PCA Script's backtick record syntax.

```
func run() -> String:
    var inscription: String = """Turn the wheel.
Then ring the bell."""
    return inscription
```

A source line containing `"$$score"` is ordinary text. GUI wildcard replacement happens in the GUI's text presentation, not automatically in every GDScript string.

5. Values, types and conversions

Type	Example	Typical use
bool	`true`, `false`	Puzzle flags
int	`3`	Counts, indices
float	`2.5`	Seconds, speeds
String	`"Archive"`	Speech, IDs
StringName	`&"idle"`	Frequently repeated names
Vector2 / Vector3	`Vector3(2, 0, -1)`	Positions and directions
Color	`Color("88bbff")`	Colors
Array / Dictionary	`[1, 2]`, `{"open": true}`	Containers
Object / Node / Resource	An instance of a class	Godot objects
Variant / null	Any value / no value	Dynamic values and missing results

```
func run() -> Array:
    var count: int = 3
    var seconds: float = 0.5
    var title := "Parts tray"
    var converted: int = int("12")
    return [count, seconds, title, converted]
```

`var title := "Parts tray"` infers a String type. `var title: String = ...` states the type directly. A variable without a type can hold different kinds of values. When a helper returns Variant, use an explicit Variant or a deliberate cast rather than forcing unsafe inference.

GDScript names are case-sensitive. `points`, `Points` and `POINTS` are three names. Text-to-number conversions need validation when input can be malformed. A nonempty string is not a reliable way to parse a player’s yes/no answer.

6. Variables, constants and enums

```
const KEY_REWARD: int = 5
enum LockState { LOCKED, OPEN, JAMMED }
var call_count: int = 0

func run() -> Array:
    var state: LockState = LockState.LOCKED
    call_count += 1
    return [KEY_REWARD, state, call_count]
```

A local variable is declared inside a function and belongs to that call. A member variable is declared outside functions and belongs to the script instance. A constant cannot be reassigned. An enum gives meaningful names to a small set of integer values.

In PCA, an ordinary native record gets a fresh instance for each call. Its member values start again. Turn on Keep global instance alive when a script must keep members, timers or connected callbacks between calls. That makes the instance persistent for the session; it does not put those members in save games.

For saved puzzle facts, use the Variable Designer and the Variable registry or variable actions. For example, `Variable.door_open.value` is game state, while a member named `door_open` is only that instance's data.

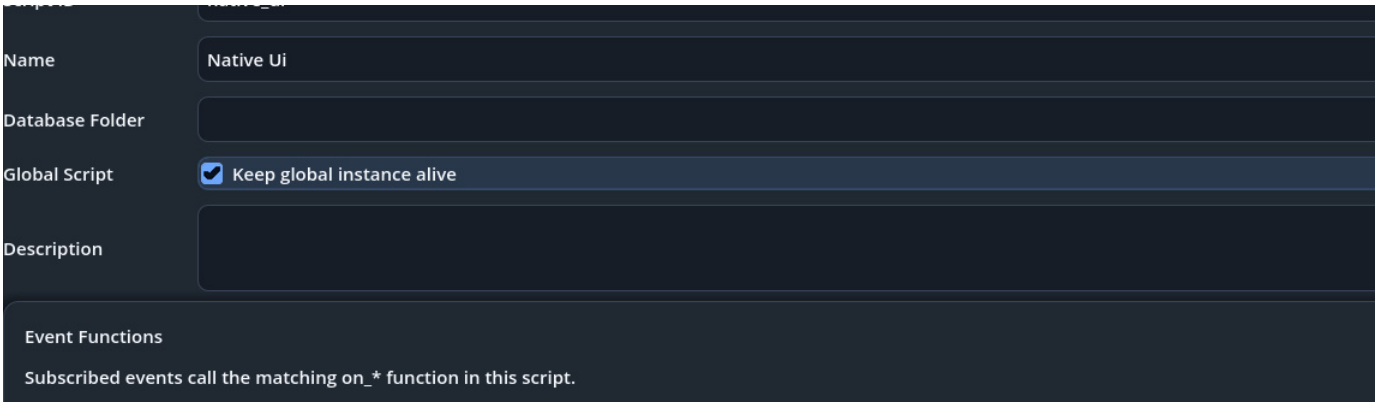

Figure 3. Keep global instance alive controls the lifetime of the authored script. Saved game variables remain a separate store.

7. Operators and arithmetic

```
func run() -> Array:
    var points: int = 10
    points += 5
    points *= 2
    var average: float = float(points) / 4.0
    var remainder: int = 11 % 3
    return [points, average, remainder, 2 ** 3]
```

Operator family	Meaning
`+ - * / % **`	Arithmetic, integer remainder, exponentiation
`= += -= *= /=`	Assignment and update
`== != < <= > >=`	Comparison
`and or not`	Boolean logic; and/or short-circuit
`in`, `not in`	Membership
`is`, `as`	Type test and object cast
`& | ^ ~ << >>`	Integer bit operations; useful for flags and masks

Dividing two integers performs integer division. Use a float operand or explicit conversion when you need a fractional result. `=` assigns; `==` compares. GDScript has no `++` or `--`: use `+= 1` or `-= 1`.

Parentheses make mixed expressions clear. For a condition involving both and and or, group the part you intend to combine first. Do not use bit operators as substitutes for ordinary Boolean tests.

8. If, elif, else and conditional expressions

```
func describe_charge(charge: float) -> String:
    if charge >= 80.0:
        return "Fully charged."
    elif charge >= 30.0:
        return "Enough for one attempt."
    else:
        return "The battery is flat."

func run() -> String:
    return describe_charge(45.0)
```

The first true branch runs. Place specific conditions before broad ones. A return can end a function early, which is useful for rejecting an impossible interaction before starting the successful sequence.

```
func run() -> String:
    var opened: bool = true
    return "Open" if opened else "Locked"
```

A conditional expression chooses one value. Keep it short; use normal branches when a choice needs several actions or a useful explanation to the player.

9. Match and named states

```
func describe_lock(state: String) -> String:
    match state:
        "locked":
            return "I need a key."
        "open", "unlatched":
            return "The way is clear."
        _:
            return "The mechanism is damaged."

func run() -> String:
    return describe_lock("open")
```

match compares a value with patterns. The underscore is the fallback. Comma-separated patterns share a response. A matched branch does not fall through into the next branch.

Use match for a clearly defined state or command type. Use if/elif when your tests are separate conditions such as inventory possession, a room visit and a distance measurement.

10. For loops, while loops and range

```
func run() -> Array:
    var total: int = 0
    for amount in [10, 20, 30]:
        total += amount
    var attempts: int = 0
    while attempts < 3:
        attempts += 1
    var odds: Array = []
    for number in range(5, 0, -2):
        odds.append(number)
    return [total, attempts, odds]
```

A for-loop visits a collection or a range. A dictionary loop visits its keys. range(stop) starts at 0 and excludes stop. range(start, stop, step) adds a start and a step; use a negative step to count down.

A while-loop repeats while its condition is true. Change the tested state inside it. Ordinary GDScript does not get PCA Script’s automatic loop guard. A loop that waits on player input must yield to the engine or it can block the frame indefinitely.

```
func run() -> Array:
    var kept: Array = []
    for number in range(10):
        if number == 2:
            continue
        if number == 5:
            break
        kept.append(number)
    return kept
```

There is no native do-while statement. A while true loop with a final test and break expresses the same pattern. PCA also supplies callable-based loop helpers, documented later.

11. Arrays and typed arrays

```
func run() -> Array:
    var clues: Array[String] = ["key", "note"]
    clues.append("watch")
    clues[1] = "torn_note"
    clues.erase("key")
    return [clues, clues.size(), clues.has("watch")]
```

The first index is 0. Check size before reading an index. `Array[String]` restricts entries to strings. append adds at the end; insert adds at an index; erase removes a matching value; remove_at removes by index.

```
func run() -> Array:
    var original: Array = [{"id": "key"}]
    var independent: Array = original.duplicate(true)
    independent[0]["id"] = "note"
    return [original[0]["id"], independent[0]["id"]]
```

Arrays are shared by reference. Assigning an array to a second variable does not create an independent list. duplicate(true) copies nested arrays/dictionaries too; contained Godot objects and resources still need their own deliberate ownership/copy policy.

Other useful methods include find, sort, reverse, shuffle, slice, filter, map and reduce. Avoid removing entries from the same array you are walking forward through; collect removals, loop backwards, or build a filtered result.

12. Dictionaries and defaults

```
func run() -> Array:
    var facts: Dictionary = {"unlocked": false, "attempts": 0}
    facts["attempts"] += 1
    facts["unlocked"] = true
    var missing: Variant = facts.get("owner", "nobody")
    return [facts["attempts"], missing, facts.has("unlocked")]
```

A Dictionary stores values by key. Use get(key, fallback) when a key can be missing. has checks whether the key exists, even if its value is false or null. keys and values return arrays. erase removes a key/value pair.

A dictionary of action settings must keep the exact field spelling. `characterId` is not `character_id`. Native GDScript identifiers are case-sensitive, and dictionary keys are data; a plausible spelling does not automatically become a supported setting.

Typed dictionaries are available in current Godot 4 releases, but ordinary Dictionary is the clearest boundary for PCA’s JSON-style records and cross-language calls. Keep transferred values simple and document the expected keys.

13. Functions, defaults and return values

```
func total_price(price: float, count: int = 1) -> float:
    return price * count

func run() -> Array:
    return [total_price(2.5), total_price(2.5, 3)]
```

A function has a name, a parameter list and an optional return type. Default arguments make trailing parameters optional. `-> void` means the function does not return a value. A bare return leaves it without a result.

Use ordinary positional arguments when calling GDScript functions. An options dictionary is still one argument, even when it contains many named fields. Required parameters must come before optional ones.

PCA normally calls run when the record is run directly. If run accepts at least one parameter and no arguments were supplied, PCA passes the caller context as its first argument. The same rule applies to on_* handlers. A named utility function receives the argument array supplied by its caller.

A local called call_context or event avoids shadowing the inherited context member. Do not override an inherited helper such as Action or CallScript with an incompatible signature.

14. Strings, formatting and regular expressions

```
func run() -> Array:
    var answer: String = "  Open The Door  ".strip_edges().to_lower()
    var words: PackedStringArray = answer.split(" ", false)
    var tidied: String = " ".join(words)
    var caption: String = "%s: %d" % ["Score", 25]
    return [tidied, caption, answer.contains("door")]
```

String methods include to_lower, to_upper, strip_edges, contains, begins_with, ends_with, replace, split, join, substr, left, right and length. Use substr(start, length) to extract a portion. A negative length means to the end.

```
func run() -> String:
    var expression := RegEx.new()
    var error_code: Error = expression.compile("([A-Z]+)-(\\d+)")
    if error_code != OK:
        return "Bad pattern"
    var found: RegExMatch = expression.search("KEY-204")
    return found.get_string(2) if found != null else "No match"
```

Native RegEx returns RegExMatch objects. PCA’s native RegexSearch helper also returns RegExMatch, which differs from PCA Script’s dictionary-shaped regex result. Use get_string, get_start and get_end on a match.

15. Math, vectors and random choices

```
func run() -> Array:
    var health: float = clampf(135.0, 0.0, 100.0)
    var halfway: float = lerpf(10.0, 20.0, 0.5)
    var distance: float = Vector3.ZERO.distance_to(Vector3(3, 0, 4))
    return [health, halfway, distance, sqrt(81.0), pow(2.0, 3.0)]
```

Godot supplies trigonometry, powers, square roots, rounding, interpolation and vector operations directly. Use deg_to_rad/rad_to_deg when an API expects radians but your designer value is in degrees. Use is_equal_approx for floating-point comparisons.

```
func run() -> int:
    var generator := RandomNumberGenerator.new()
    generator.seed = 1234
    return generator.randi_range(1, 6)
```

A separate RandomNumberGenerator gives your puzzle its own sequence. A fixed seed repeats a test sequence. Global randi_range/randf_range and PCA’s random helpers use shared randomness. Godot initializes the global generator; do not reseed it every time you roll.

Vector3 has x/y/z, length, normalized, dot, cross, distance_to and lerp. A position and a direction are different quantities. Normalizing a direction removes its original length.

16. Read player input from an authored GUI

```
func run() -> bool:
    var entered: String = str(GetGUIElementProperty(
        "name_entry", "name_box", "text", ""
    )).strip_edges()
    if entered.is_empty():
        SetGUIElementProperty("name_entry", "feedback", "text", "Enter a name.")
        return false
    Variable.player_name.value = entered
    SetGUIElementProperty("name_entry", "feedback", "text", "Hello, " + entered + ".")
    return true
```

1. Create a GUI called name_entry with a Text Box named name_box, a Label named feedback, and a Confirm button.
2. Create a text variable player_name. Put the example in a native script record called accept_name.
3. On Confirm → Events → Click, choose Run Visual Script / Run Script and target accept_name. Leave the event's Verb blank.
4. Validate and apply the script, save the GUI, show it during play, type a name and click Confirm.

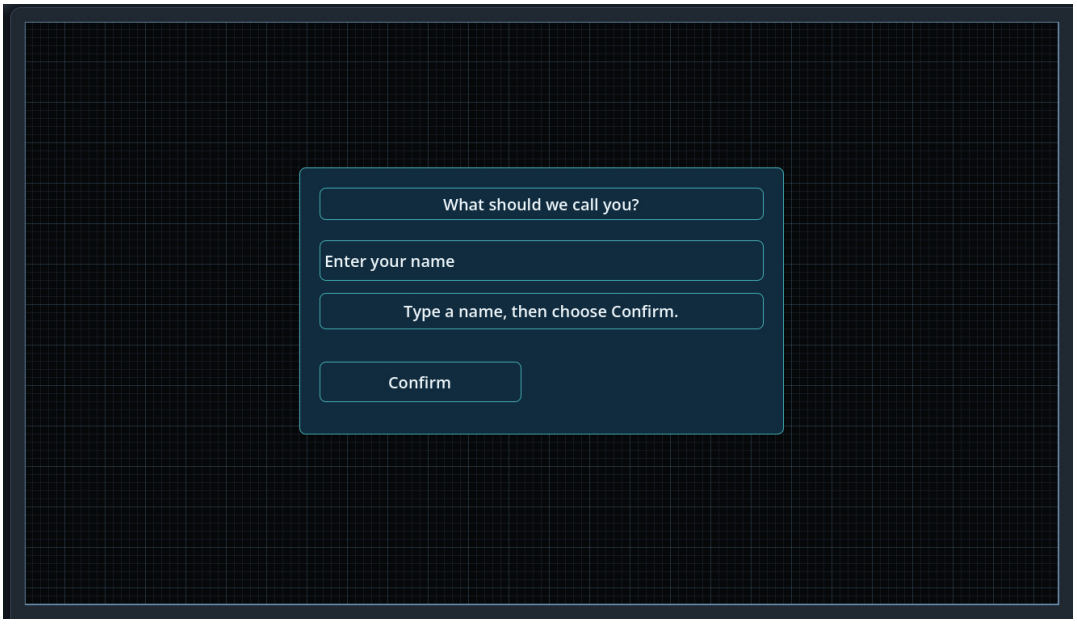

Figure 4. The authored interface collects the name; its Confirm event calls the native GDScript record.

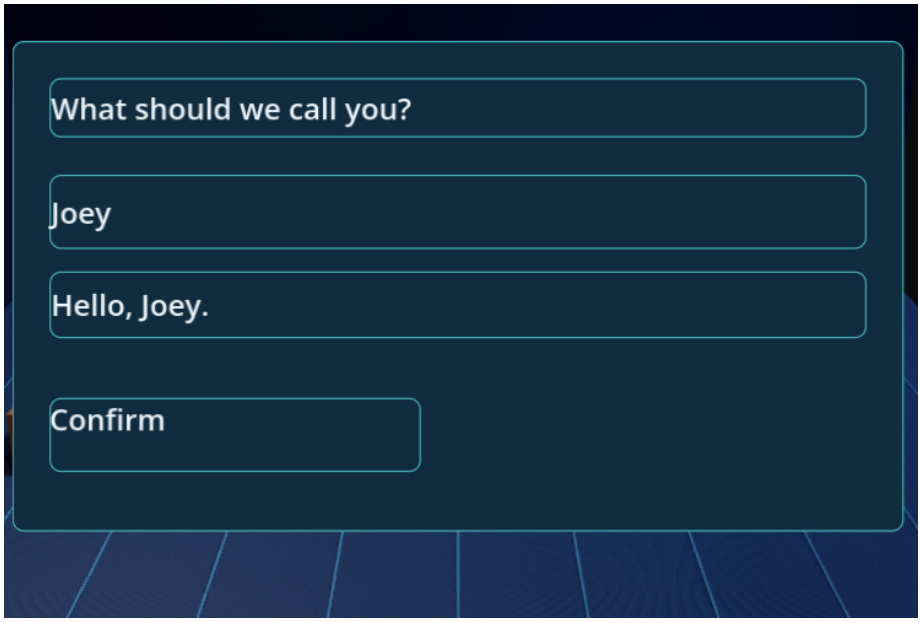

Figure 5. The running game reads the live text box and changes its feedback label.

There is no automatic terminal input box attached to a GDScript record. GUI controls and the text parser are the normal player-facing routes. Sliders and checkboxes can also bind to game variables.

17. Waits, timers and elapsed time

```
func run() -> String:
    await get_tree().create_timer(0.01).timeout
    return "Finished waiting"
```

await suspends this function until the signal or coroutine completes. It yields control to the engine; it does not freeze the whole game. A caller that needs the result must await this function too.

```
var ticker: Timer
var ticks: int = 0

func on_global_ready(event: Dictionary = {}) -> void:
    ticker = Timer.new()
    ticker.wait_time = 1.0
    add_child(ticker)
    ticker.timeout.connect(_on_tick)
    ticker.start()

func _on_tick() -> void:
    ticks += 1
    Log("Timer ticks: " + str(ticks))
```

Turn on Keep global instance alive for this repeating-timer record. on_global_ready is called after PCA binds the runtime. The Timer is owned by the script, so it is freed with that instance. stop pauses future timeout emissions; start begins it again.

For measuring durations, Time.get_ticks_msec/usec is monotonic. Time.get_unix_time_from_system is the calendar clock and can change. A default SceneTree timer processes while the tree is paused; pass process_always=false if it should pause with the tree.

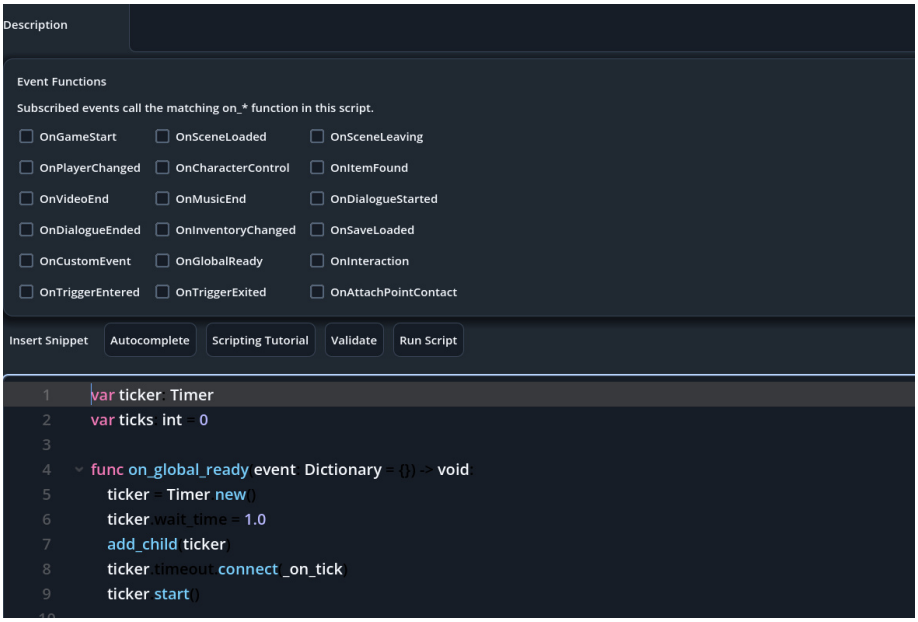

Figure 6. A persistent timer script keeps its counter between ticks.

18. Native file reading, writing and appending

```
func run() -> String:
    var path: String = "user://gdscript_tutorial_notes.txt"
    var output: FileAccess = FileAccess.open(path, FileAccess.WRITE)
    if output == null:
        return error_string(FileAccess.get_open_error())
    output.store_string("Visited the archive.\n")
    output.close()
    var input: FileAccess = FileAccess.open(path, FileAccess.READ)
    if input == null:
        return "Read failed"
    return input.get_as_text()
```

user:// is the writable data directory for the running application. res:// is its packaged Godot resource root. FileAccess.WRITE replaces the file. READ_WRITE opens an existing file for both directions; WRITE_READ creates/truncates one.

```
func append_line(path: String, line: String) -> Error:
    var mode: FileAccess.ModeFlags = FileAccess.WRITE_READ
    if FileAccess.file_exists(path):
        mode = FileAccess.READ_WRITE
    var file: FileAccess = FileAccess.open(path, mode)
    if file == null:
        return FileAccess.get_open_error()
    file.seek_end()
    file.store_line(line)
    file.close()
    return OK
```

To edit a specific line in a text file, read it, split it into lines, change the String array and rewrite the complete text. File cursor positions are bytes; do not assume an arbitrary byte offset is a safe character boundary in UTF-8 text.

> PCA’s WriteTextFile/WriteJSONFile helpers accept user:// only. Native FileAccess is a Godot API with the platform’s filesystem permissions; it is not the same restricted helper. Use user:// for portable player data.

19. JSON, configuration and file operations

```
func run() -> Array:
    var text: String = JSON.stringify({"music": 0.5, "name": "Ada"})
    var parser := JSON.new()
    if parser.parse(text) != OK:
        return []
    var data: Dictionary = parser.data
    return [data["music"], data["name"]]
```

JSON.parse_string is convenient; JSON.new().parse gives an error code and diagnostics. JSON does not preserve every Godot value type. Encode vectors as arrays or dictionaries, and reconstruct them explicitly when loading.

```
func run() -> float:
    var configuration := ConfigFile.new()
    configuration.set_value("audio", "music", 0.5)
    var saved: Error = configuration.save("user://gdscript_tutorial.cfg")
    if saved != OK:
        return -1.0
    var loaded := ConfigFile.new()
    if loaded.load("user://gdscript_tutorial.cfg") != OK:
        return -1.0
    return float(loaded.get_value("audio", "music", 0.7))
```

DirAccess can create directories, list entries, rename and remove files. Check Error results. Perform test edits inside your own user-data tutorial folder. Deleting a custom note file is different from deleting a PCA save slot; use the save-widget/save-system route for game saves.

For an imported image, model or other engine resource, ResourceLoader is usually the correct export-aware reader. An original source file may be converted during export and unavailable as plain bytes.

20. Dates and calendar values

```
func run() -> Dictionary:
    var date: Dictionary = Time.get_datetime_dict_from_unix_time(0)
    return {"year": date["year"], "month": date["month"], "day": date["day"]}
```

Time.get_datetime_dict_from_system returns the current calendar components; an optional UTC flag selects UTC rather than local system time. Time.get_datetime_string_from_system creates a display string. Unix timestamps represent seconds since the epoch.

PCA’s NowUnix, DateTimeNow, DateTimeString and UnixToDateTime are convenience wrappers. UnixToDateTime retains a compatibility UTC parameter, but the conversion itself uses Godot’s UTC timestamp dictionary. Do not use the system calendar clock as an unquestioned gameplay stopwatch.

21. Errors, assertions and recovery

```
func parse_number(entered: String) -> Dictionary:
    if not entered.is_valid_int():
        return {"ok": false, "error": "Enter a whole number."}
    return {"ok": true, "value": entered.to_int()}

func run() -> int:
    var result: Dictionary = parse_number("42")
    return int(result.get("value", 0)) if result["ok"] else 0
```

GDScript does not provide try/catch exceptions. Use guards, Error codes, null checks and explicit result dictionaries. A Godot runtime error is not something PCA's TryCall can reliably catch after it has happened.

TryCall checks whether a Callable is valid, then calls it and packages a result. It is not an exception trap. PCAAssert reports a failed condition and returns a Boolean. Native assert is a development check and is ignored in non-debug builds; do not use it as your player-facing error handling.

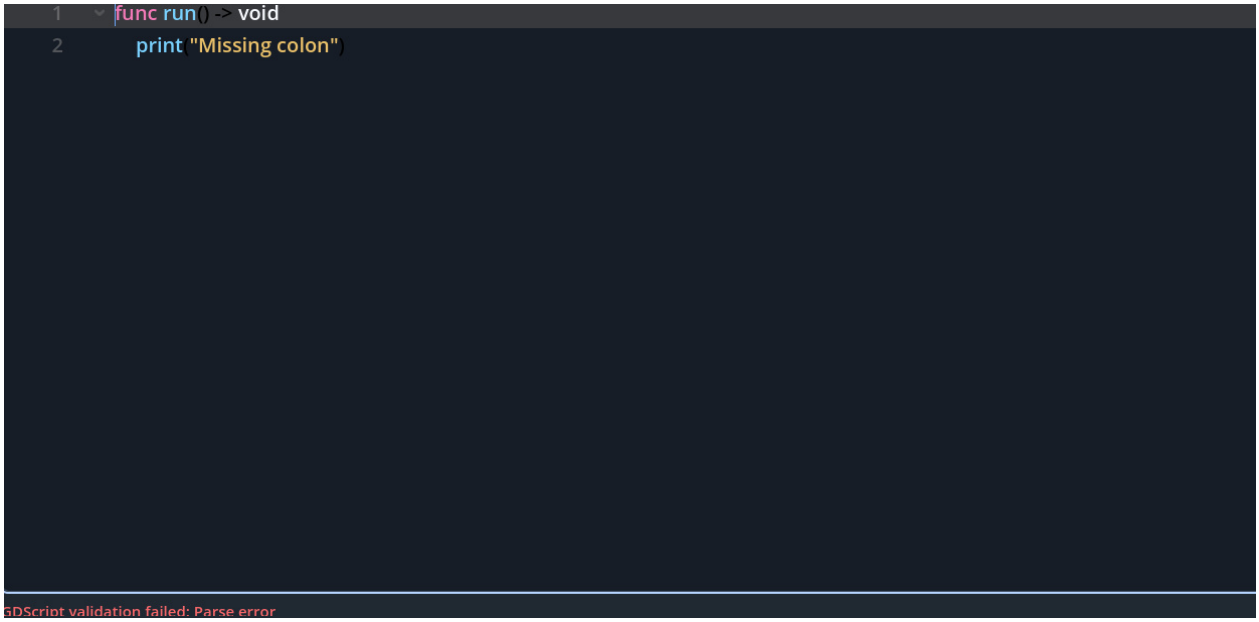

Figure 7. Validate catches parser problems before the record is run. Runtime logs and the Godot debugger address a different stage.

Use is_instance_valid on a Node you saved earlier, not merely node != null. A reference can still exist after the underlying object was freed.

22. Your run function and its caller

```
func run(call_context: Dictionary = {}) -> String:
    var verb_id: String = str(call_context.get("verb", ""))
    var source_id: String = str(call_context.get("entityId", ""))
    Log("Verb: " + verb_id + "; source: " + source_id)
    return source_id
```

The caller context explains why the script ran. An object response, a GUI button, a parser command and a manual Run Script call can provide different keys. Read an optional key with get and a fallback.

The inherited context member is refreshed for the call, and PCA’s helpers pass it on to nested actions/scripts. The parameter shown above is a local copy/reference to the supplied dictionary; naming it call_context avoids hiding the base member.

For initial inspection, Log(ToJSON(call_context)) shows ordinary data. Do not assume every context contains entityId, itemId, guiId, elementId or a parser command. Discover the relevant payload in a small test, then depend only on the keys that route supplies.

23. Self, this and the player

```
func run() -> bool:
    if this == null:
        Warn("This interaction needs a caller.")
        return false
    Log("Caller ID: " + str(this.Get("id", "")))
    return true
```

self is the Godot script instance. this is PCA’s caller proxy when a caller is available. player is the active character’s PCA record proxy, refreshed at the start of a call. None of these automatically means the selected editor object.

A proxy describes a PCA record and routes supported changes. It is not the rendered Node3D. Adding a Timer to self creates a Godot child of the script. Changing this.visible addresses the caller’s record/runtime route instead.

A nested call normally preserves caller context. A direct Run or a global event may have no caller, so reusable interaction code must tolerate null. If the active player can change during an awaited sequence, capture the intended character ID before waiting.

24. Find records through registries

```
func run() -> String:
    var door: Variant = Objects.Get("archive_door")
    if door == null or not door.Exists():
        return "Door is missing"
    var title: String = str(door.Get("name", "Door"))
    Log(title)
    return title
```

The main registries are Character, Scene, Objects, Asset, Inventory, Cursor, Verb, GUI, Dialogue, Quest, Variable, Fonts and Cutscene. Script registries cover the supported languages. Project and Settings provide project-level data; Global provides script-call proxies.

Use Get when the ID is in a string, contains punctuation or might collide with an inherited method. `Objects.archive_door` is a convenient direct spelling for a simple ID. Registry IDs and display names serve different purposes; use stable IDs in calls.

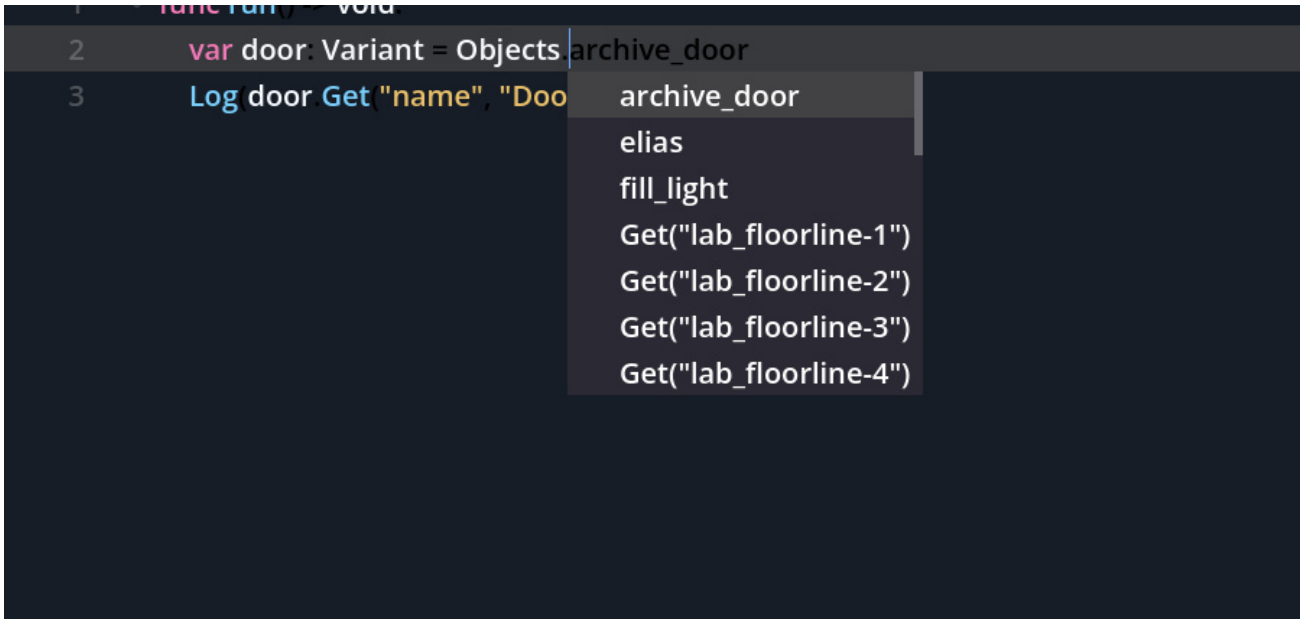

Figure 8. Registry-aware completion helps choose a record rather than guessing a display label.

PCA’s registries are separate from Godot’s SceneTree. Scene.lab is room data. get_tree().current_scene is the loaded top-level Godot scene, which can be the PCA editor or runner.

25. Read and change designer properties

```
func run() -> bool:
    var door: Variant = Objects.Get("archive_door")
    if door == null or not door.Exists():
        return false
    door.Set("visible", true)
    door.Set("transform.position", Vector3(2, 0, -3))
    return bool(door.Get("visible", false))
```

Get and Set accept a property path. Direct property syntax is convenient too, but Get can provide a fallback and can return an actual Dictionary instead of a nested proxy. Use that when you need to duplicate or edit container data.

Definition changes and live changes are different. A model or animation-state definition may need a loaded actor to rebuild. A live transform/visibility update can affect the current object immediately. The property reference marks that distinction.

Use record.Custom or a custom-only proxy for project-defined properties. Do not rename a record by assigning its ID. Edit an array copy and write the complete array back, or use a dedicated runtime action; a nested array mutation is not a reliable live notification.

> The native Get/Set string-path route does not use PCA Script’s dotted-name canonicalizer. Use the full stored path for shader/particle fields rather than transplanting PCA Script’s special workaround unchanged.

26. Gameplay actions and await

```
func run() -> void:
    await AddInventory({"itemId": "key", "count": 1})
    await SetObjectVisible({"entityId": "key_pickup", "visible": false})
    await PlaySound({"assetId": "chime", "channel": "puzzle"})
    await ShowMessage({"text": "The key is in my bag.", "duration": 2})
```

All 206 normal shared actions are PascalCase methods on PCAScriptBase. They accept one options dictionary. Field keys keep their original spelling. You can also use `await Action("ActionID", options)`.

Awaiting an action waits for its runtime routine. It does not promise that every visible effect has finished. Starting music or most imported animations returns while playback continues. A camera blend or fade may wait, and some actions have a wait option.

The result depends on the action. It can be null, a Boolean, query data or a status dictionary. Native action calls do not gain PCA Script’s try/catch semantics. Check meaningful results and the resulting state when a failure needs recovery.

27. Build a one-time key-and-door interaction

```
func run() -> bool:
    if bool(Variable.door_open.value):
        await ShowMessage({"text": "It is already unlocked.", "duration": 1.5})
        return true
    if not HasInventory("key"):
        await ShowMessage({"text": "I need the archive key.", "duration": 1.5})
        return false
    Variable.door_open.value = true
    await AddScore({"amount": 5})
    await PlaySound({"assetId": "chime", "channel": "puzzle"})
    await ShowMessage({"text": "The archive is unlocked.", "duration": 2})
    return true
```

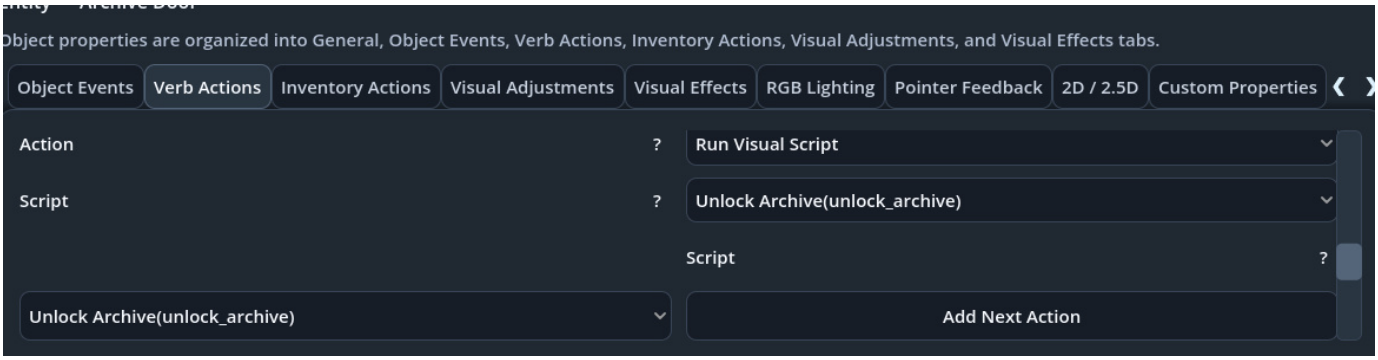

Figure 9. The native interaction response calls a script; the object still owns its hotspot, verb and approach settings.

1. Create the key inventory definition, a Boolean door_open variable initially false, and a numeric score variable initially 0.
2. Create the native script unlock_archive. Validate, apply and save it.
3. In the door’s Use response, add Run Visual Script / Run Script and choose unlock_archive.
4. Test without the key, with the key and with a second click. Restore the door’s appearance from door_open whenever the room loads.

Set a one-time flag before long speech or movement so repeated clicks cannot award another reward. If completion can fail after the state changes, decide deliberately whether to keep the fact or roll it back.

28. Subscribe to PCA events

```
func on_scene_loaded(event: Dictionary = {}) -> void:
    if str(event.get("sceneId", "")) != "lab":
        return
    Log("The workshop is ready.")

func on_trigger_entered(event: Dictionary = {}) -> void:
    if str(event.get("triggerId", "")) == "welcome_trigger":
        Log("The player crossed the welcome area.")
```

Write the matching on_* function, then check the corresponding Event Functions box in the GDScript editor. Writing a handler name alone is not a subscription to every runtime broadcast.

The event argument is a Dictionary. OnSceneLoaded provides the room ID; trigger events provide triggerId and related context. Filter a broad listener before doing local work. A local door response and a global interaction listener can both run, so guard one-time rewards.

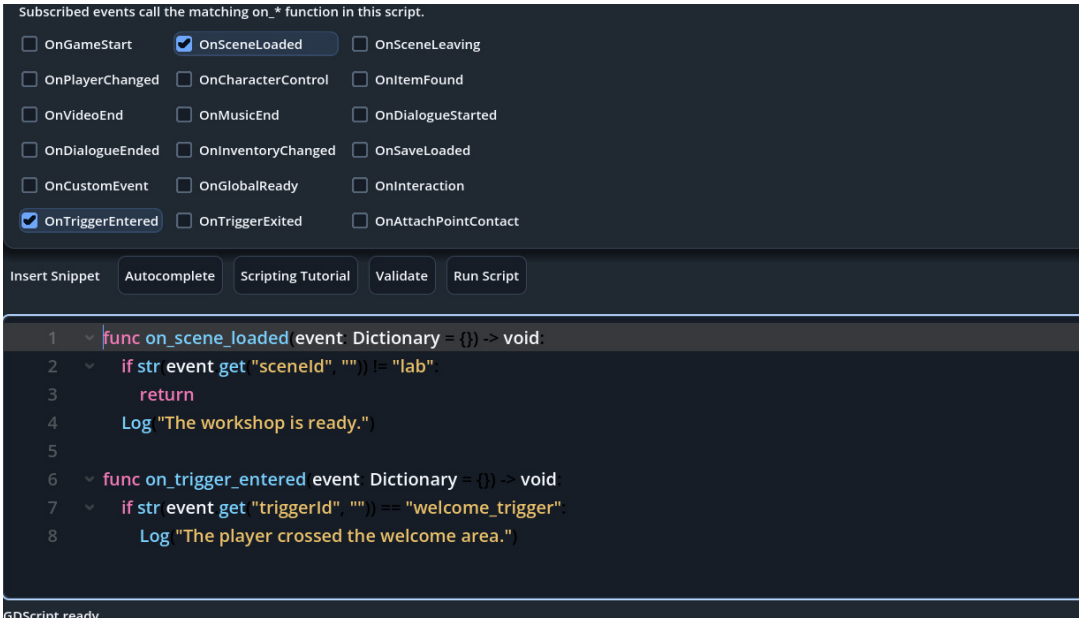

Figure 10. The subscriptions tell PCA which event functions to dispatch to this record.

on_global_ready is a special startup call for a global native record that defines it. You do not need a second ordinary subscription merely to initialize that global instance.

29. Custom events and Godot signals

```
func on_custom_event(event: Dictionary = {}) -> void:
    if str(event.get("reason", "")) == "signal_powered":
        Log("The lamp has power.")

func run() -> void:
    await EmitEvent("OnCustomEvent", {"reason": "signal_powered"})
```

This uses the standard OnCustomEvent route and a reason field. A custom exact event name is also possible, but a native handler needs a route that maps to a real method. For a shared native listener, the standard hook plus a discriminator is straightforward.

Godot signals are a different mechanism: a particular object emits a typed signal to connected Callables. Use PCA events for adventure-wide notifications and Godot signals for relationships between your own nodes. The Godot signal chapter shows the full connection pattern.

Do not broadcast the same notification from its own handler without a guard. Set saved state before emitting a completion event, and avoid circular chains of listeners.

30. Speech, portraits, lip-sync and typewriter text

```
func run() -> void:
    await Say({
        "speakerId": "elias",
        "text": "Someone has been here before us.",
        "typewriter": true,
        "typewriterSpeed": 35,
        "typewriterSound": "default"
    })
```

Say uses the speaker’s configured speech/portrait route. ShowMessage is a narrator/message route. StartDialogue runs an authored conversation graph. A voice clip, portrait mapping and lip-sync track come from their configured assets; a plain string does not create those assets.

Typing sound can be default, none or custom. For custom, set typewriterSoundAssetId. For voiced speech, use voiceAssetId and the appropriate voice key/portrait settings shown in the action reference. Test both short and long lines, and check whether the line waits for continuation.

Use the game’s translation workflow for player-facing text. String.format and PCA’s Format build text; they do not automatically register a new translation entry.

31. Cameras, scene changes and effects

```
func run() -> void:
    await ActivateCamera({"cameraId": "wide", "duration": 0.8, "wait": true})
    await ZoomBlur({
        "strength": 0.2,
        "duration": 0.5,
        "includeGUI": false,
        "wait": true
    })
```

Use PCA camera actions for authored camera behavior. They know about the game’s camera definitions, blending and follow configuration. Directly replacing a camera node can leave PCA’s camera state pointing somewhere else.

Use ChangeScene for an adventure room change. Godot’s SceneTree.change_scene_to_file replaces the top-level Godot scene and can discard the PCA editor/runner structure; it is not the normal way to enter another PCA room.

Short zoom blur, fades, overlays, screen shader styles and local effects use different scopes. Restore temporary input locks, pointer state and effects after a sequence. Test the target renderer: a supported data field does not make every GPU effect available on every render path.

32. Save state, instance state and project settings

```
var visit_counter: int = 0

func run() -> int:
    visit_counter += 1
    Log("Calls this session: " + str(visit_counter))
    Variable.powered.value = true
    return visit_counter
```

Store	Lifetime / purpose
Local variable	One function call
Member on ordinary native record	One temporary script instance/call
Member on global native record	Current session until reload, invalidation or shutdown
Variable registry / game variables	Game state managed by PCA save/load
Included-variable store	Router-owned shared session/cache state
user:// custom file	Your explicit persistence format
Runtime project/record data	Configuration and live state; persistence depends on the specific route

GetSetting/SetSetting and GetProjectProperty/SetProjectProperty address the running project. They do not save the editor document to disk. For a live feature with a dedicated action, use that action when immediate reconfiguration matters.

33. How to access Godot functionality

```
func run() -> Dictionary:
    var information: Dictionary = Engine.get_version_info()
    var elapsed_ms: int = Time.get_ticks_msec()
    var position: Vector3 = Vector3(2, 0, -1)
    return {
        "godot": information.get("string", ""),
        "elapsed_ms": elapsed_ms,
        "distance": position.length()
    }
```

No extra import is needed for built-in Godot classes and singletons. You can call Time, Engine, Input, FileAccess, ResourceLoader, ClassDB and the normal GDScript functions directly. Class names and method names keep Godot’s case and spelling.

You need…	Typical Godot route
A new node	Timer.new(), Label.new(), Node3D.new()
A delay	await get_tree().create_timer(seconds).timeout
A signal callback	node.signal_name.connect(callable)
An asset resource	load / ResourceLoader.load after checking the path
A new class from a file	preload or load a .gd Script, then new()
A native scene	load a PackedScene, then instantiate()
A runtime check	is_instance_valid, is, has_method, ClassDB queries

Access to the API is not automatic integration with PCA’s saved records. A native cube you create yourself is not automatically an adventure hotspot, a saved object or a navigation obstacle. Decide which system owns it.

34. Keep the PCA base when extending a script

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

func run() -> String:
    Log("The PCA helpers are still inherited.")
    return get_class()
```

This explicit base path is equivalent to keeping PCA’s script base. Omitting extends also lets PCA add it. Writing `extends Node` creates a plain Node script; it does not inherit PCAScriptBase, so bare Say, Action, Variable and other inherited helpers are no longer supplied.

A PCA record must instantiate as a Node. A standalone Resource or RefCounted helper is useful, but it belongs inside a helper class/resource file or as an inner class, not as the root class of a PCA native record.

To add spatial behavior while keeping the adventure base, create a Node3D child or a helper class instance. GDScript has single inheritance. You cannot extend PCAScriptBase and Node3D as two separate parents.

> Do not redeclare Runtime, context, player or this on a script that already inherits the PCA base. On a deliberately plain Node extension, only explicitly present Runtime/context properties can be injected by the router.

35. Startup order and script lifetime

Hook	What is available
_init	The instance is being constructed; no PCA runtime binding yet.
_enter_tree / _ready	Godot tree lifecycle. PCA adds the node before binding Runtime and record proxies.
on_global_ready	PCA’s deferred startup hook after binding a global record.
run / on_* event	PCA has refreshed caller context for that routed call.
_process / _physics_process	Only while the instance exists and processing is active.
_exit_tree	Cleanup as the instance leaves the tree.

```
var ready_before_binding: bool = false

func _ready() -> void:
    ready_before_binding = Runtime == null

func run() -> bool:
    return ready_before_binding
```

Put PCA-dependent startup work in on_global_ready or a routed function/event. Pure local initialization can happen in _init/_ready. If you are creating something inside the current room, wait for the relevant room-load event or a direct call after the room exists.

A temporary record is queued for deletion after the awaited call completes. A Timer added to it cannot keep a repeating job alive after the function returns. For a long-lived controller, use the global checkbox and deliberate cleanup. Editing/invalidation can replace that instance.

36. Nodes, parents, paths and ownership

```
func run() -> bool:
    var marker := Node.new()
    marker.name = "TutorialMarker"
    add_child(marker)
    var found: Node = get_node_or_null("TutorialMarker")
    var correct: bool = found == marker
    marker.queue_free()
    return correct
```

A Node belongs to one parent. add_child places it in the tree; queue_free schedules its deletion. Use get_node_or_null when absence is acceptable. A leading slash is an absolute path from the SceneTree root; relative paths begin at the calling node.

A PCA script is parented under the script engine, not under the door that called it. `$Mesh` means a child of the script node. It does not search the rendered door model. `%UniqueName` depends on Godot scene ownership and a unique-name setup, not a PCA record ID.

Node.owner controls what can be included when packing a Godot scene. It is not the same as parent, PCA’s this, inventory ownership or the active player. For runtime-only nodes, a correct parent/lifetime is usually the first concern.

37. Godot signals and Callables

```
signal reward_requested(points: int)
var received_points: int = 0

func run() -> int:
    if not reward_requested.is_connected(_receive_reward):
        reward_requested.connect(_receive_reward)
    reward_requested.emit(5)
    return received_points

func _receive_reward(points: int) -> void:
    received_points += points
```

A signal is declared on an object. connect tells it which Callable to run. emit supplies the arguments. Keep the receiver’s parameter list compatible with the signal. is_connected prevents accidentally wiring the same callback twice.

A Callable can be a named method, Callable(object, method_name), or a lambda. bind adds fixed arguments after the emitted arguments. disconnect removes a connection. CONNECT_ONE_SHOT can make a connection remove itself after one emission.

```
func add_points(base: int, bonus: int) -> int:
    return base + bonus

func run() -> int:
    var add_bonus: Callable = add_points.bind(5)
    return int(add_bonus.call(10))
```

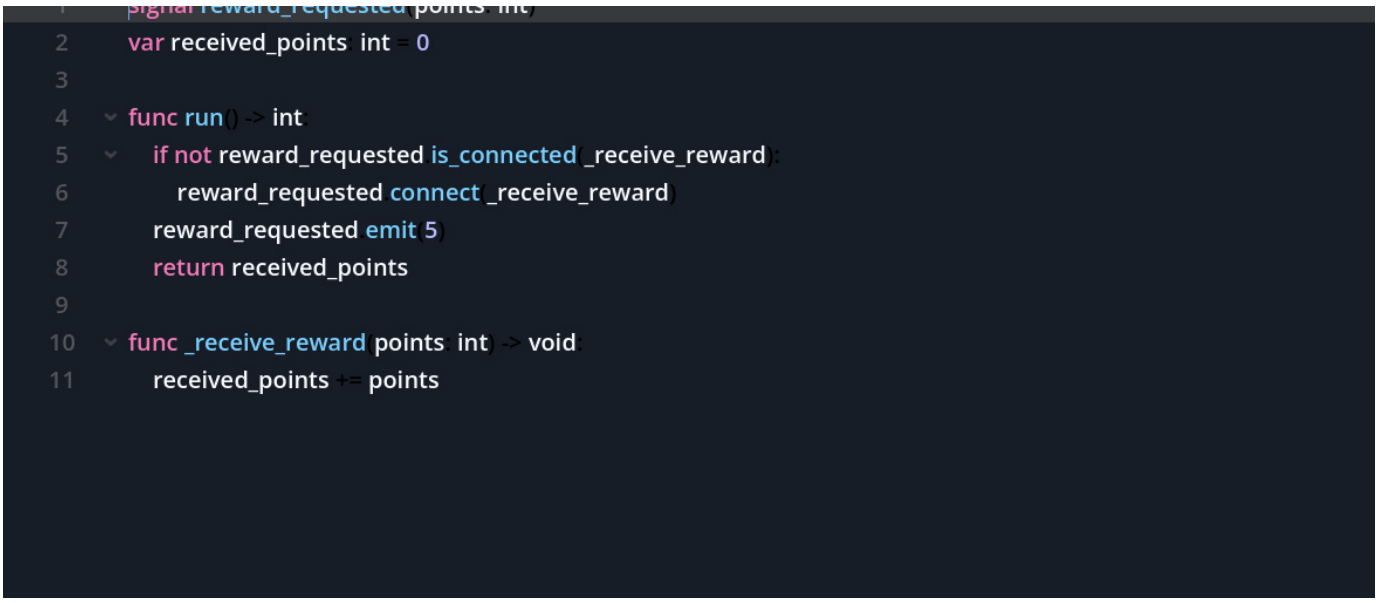

Figure 11. A typed signal and its receiver are normal GDScript, even inside a PCA record.

38. A native Godot button inside a PCA game

```
var overlay: CanvasLayer
var native_status: Label
var click_count: int = 0

func run() -> void:
    if is_instance_valid(overlay):
        return
    overlay = CanvasLayer.new()
    overlay.layer = 120
    add_child(overlay)
    var panel := VBoxContainer.new()
    panel.position = Vector2(35, 100)
    overlay.add_child(panel)
    native_status = Label.new()
    native_status.text = "Native Godot control: 0 clicks"
    panel.add_child(native_status)
    var button := Button.new()
    button.text = "Test Godot signal"
    panel.add_child(button)
    button.pressed.connect(_on_native_pressed)

func _on_native_pressed() -> void:
    click_count += 1
    native_status.text = "Native Godot control: %d clicks" % click_count
```

Make this record global, then run it during play. It creates a CanvasLayer, a layout container, a label and a button. The pressed signal changes the label. All nodes are children of the script, so they go away with that instance.

This is a useful small custom-tool pattern. For interfaces you want to author, translate, theme and manage through PCA, the GUI Designer remains the easier route. A native Control is not automatically registered as a PCA GUI element.

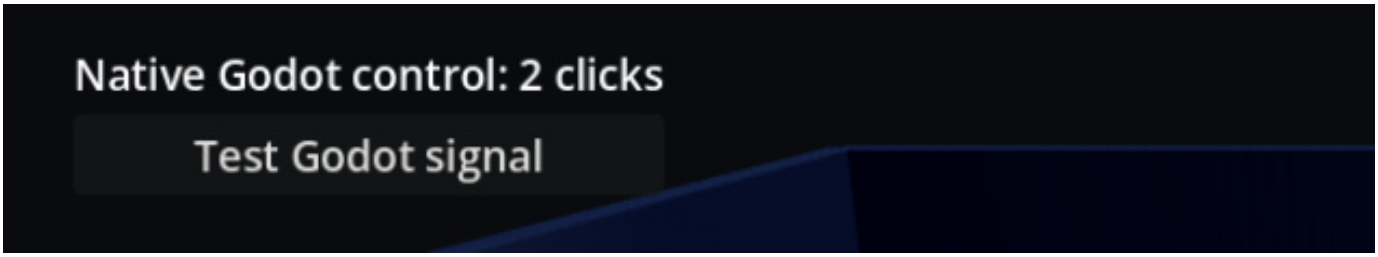

Figure 12. A real runtime test of the native Godot button and its pressed signal.

39. Processing and a frame-based countdown

```
var countdown: float = 0.0
var timer_running: bool = false

func run() -> void:
    countdown = 10.0
    timer_running = true

func _process(delta: float) -> void:
    if not timer_running:
        return
    countdown = maxf(0.0, countdown - delta)
    if countdown <= 0.0:
        timer_running = false
        Log("Countdown finished.")
```

Keep the instance alive. delta is elapsed seconds for this frame, so the countdown follows time rather than frame count. Use _physics_process for fixed-step physics work, not simply because a script changes a number.

A process callback should return quickly. Do not start an awaited long routine every frame; it can create many overlapping routines. Set a state flag before starting a job, or use a dedicated Timer/signal.

PCA can pause/suspend preview workspaces and runtime processing. Time scale and pause settings matter. A global checkbox keeps the instance; it does not force every callback to run while its tree/parent is disabled.

40. Godot input and PCA controls

```
func run() -> bool:
    var action_name: StringName = &"inspect_close"
    if not InputMap.has_action(action_name):
        return false
    return Input.is_action_pressed(action_name)
```

Input actions need to exist in Godot’s InputMap. A PCA keybinding record and a Godot InputMap action are not automatically the same name/route. Check that the action exists; configure it in your Godot project or add it deliberately at runtime.

```
func _unhandled_input(event: InputEvent) -> void:
    if event is InputEventKey:
        var key_event := event as InputEventKey
        if key_event.pressed and not key_event.echo and key_event.keycode == KEY_F6:
            Log("F6 reached the native script.")
            get_viewport().set_input_as_handled()
```

_unhandled_input is useful after GUI controls have had their chance. _input sees events earlier. Consume an event only when your feature owns it, or ordinary game controls can stop receiving it. Test focus, GUI typing, touch and controller routes as appropriate.

> Direct Input polling can bypass adventure interaction rules. Use PCA’s movement/input actions to change the game’s control profile instead of simultaneously driving the same actor from another movement loop.

41. Tweens and temporary visual changes

```
func run() -> bool:
    var label := Label.new()
    label.text = "Fading notice"
    add_child(label)
    var animation: Tween = create_tween()
    animation.tween_property(label, "modulate:a", 0.0, 0.02)
    await animation.finished
    var finished: bool = is_zero_approx(label.modulate.a)
    label.queue_free()
    return finished
```

create_tween binds the Tween to this node’s lifetime. tween_property animates a property. By default successive steps run in sequence; set_parallel or parallel changes grouping. await finished lets the caller wait for the complete sequence.

Only tween properties the target actually has. A Label has modulate; a plain Node does not. A PCA record proxy is not automatically a Tween target for a rendered mesh. Get the native node deliberately or use the appropriate PCA action.

Avoid two systems writing the same property each frame. A Tween cannot reliably win a tug of war with PCA’s active camera controller, animation system or GUI state updater. Restore temporary presentation from saved game state when needed.

42. Add a native 3D object to the room

```
var cube: MeshInstance3D

func run() -> bool:
    var room: Node3D = Runtime.get("active_room_root") as Node3D
    if room == null or not is_instance_valid(room):
        return false
    if is_instance_valid(cube):
        cube.queue_free()
    cube = MeshInstance3D.new()
    var mesh := BoxMesh.new()
    mesh.size = Vector3(1.2, 1.2, 1.2)
    cube.mesh = mesh
    var material := StandardMaterial3D.new()
    material.albedo_color = Color("3fb9b1")
    cube.material_override = material
    cube.position = Vector3(0, 1.2, 0)
    room.add_child(cube)
    return true

func _exit_tree() -> void:
    if is_instance_valid(cube):
        cube.queue_free()
```

Run this after a 3D room has loaded. active_room_root is a version-specific runtime member, so the cast and validity check matter. The cube is a native MeshInstance3D in the current room, with its own BoxMesh and material.

It is not automatically a PCA object definition, hotspot, blocking volume or saved item. Use Create Primitive/Add Asset to Scene when you want the adventure engine to own those behaviors. Use native nodes when your extension genuinely owns their rendering and lifecycle.

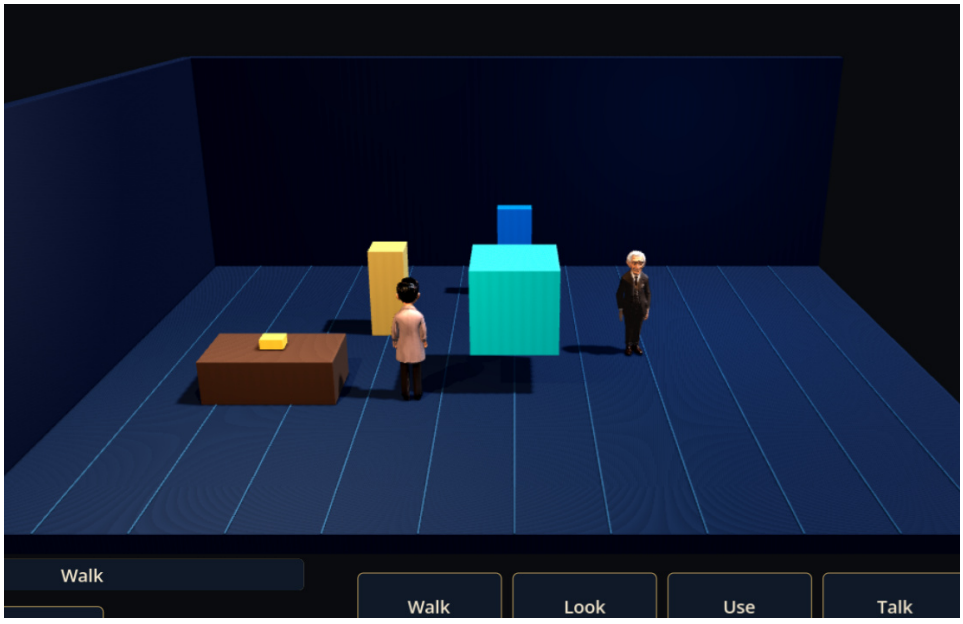

Figure 13. A native Godot MeshInstance3D created by a global GDScript record in the running PCA room.

43. Reach an existing model without guessing a node path

```
func run() -> bool:
    var nodes: Dictionary = Runtime.get("entity_nodes")
    var object_node: Node3D = nodes.get("archive_door") as Node3D
    if object_node == null or not is_instance_valid(object_node):
        return false
    Log("Rendered node class: " + object_node.get_class())
    return true
```

This is an intentional lower-level bridge. The entity_nodes table is runtime implementation data in this build, not a general stable registry API. Guard it and retest extensions when updating PCA. A character can be replaced/rebuilt, so a cached Node reference can become invalid.

The returned node may be a wrapper with an imported model below it. Use known child names from your own scene, or a constrained find_child/find_children search. Do not assume that every asset has the same hierarchy.

For ordinary visibility, animation, position and inventory changes, use the PCA action/proxy. Native node manipulation is for work outside that higher-level surface, with ownership and restoration handled deliberately.



Figure 14. The running preview confirms that the rendered archive_door node is a Node3D; its model is below that wrapper.

44. Materials, shaders, animation and resource sharing

```
func make_material() -> StandardMaterial3D:
    var material := StandardMaterial3D.new()
    material.albedo_color = Color("caa46d")
    material.roughness = 0.2
    material.metallic = 0.8
    return material

func run() -> float:
    return make_material().metallic
```

A loaded material/resource can be shared by multiple instances. Duplicate it before a local edit when other objects must keep their original appearance. material_override applies to the whole MeshInstance3D; per-surface overrides target a particular surface.

For ShaderMaterial, use set_shader_parameter with the exact uniform name from the shader. For AnimationPlayer, check has_animation before play; animation_finished is useful for a finite clip. A looping animation does not naturally emit a final completion you can wait for.

PCA can manage actor animation state, lip-sync, glow and materials itself. Direct writes can be overwritten by that system. Choose the PCA route for ordinary actor behavior; reserve direct Godot edits for a feature that owns that portion of the model.

Metallic/low roughness makes a reflective material response. A recognizable live character reflection needs the relevant rendering/reflection system; a material number alone does not construct a mirror camera.

45. Physics, areas and ray queries

```
func make_blocker() -> StaticBody3D:
    var body := StaticBody3D.new()
    var collision := CollisionShape3D.new()
    var box := BoxShape3D.new()
    box.size = Vector3.ONE
    collision.shape = box
    body.add_child(collision)
    return body

func run() -> String:
    var body: StaticBody3D = make_blocker()
    var result: String = body.get_class()
    body.free()
    return result
```

Godot physics needs CollisionObject/PhysicsBody/Area nodes, shapes and matching collision layers/masks. A visible mesh alone is not a collision body. Perform physics-space queries at the appropriate physics stage, and defer collision changes when a physics callback requires it.

A ray query uses PhysicsRayQueryParameters3D and the active World3D's direct_space_state.intersect_ray. Camera screen coordinates must belong to that camera's viewport. PCA can render a room into an internal viewport; raw window coordinates are not automatically correct.

PCA navigation obstacles, hotspot picking and event volumes are not automatically identical to Godot physics bodies. Use PCA trigger events for normal adventure actors unless your extension has supplied/tested real physics bodies for that interaction.

46. Resources, load and preload

```
const ADVENTURE_BASE = preload("res://scripts/runtime/script_base.gd")

func run() -> bool:
    var script_resource: Resource = load("res://scripts/runtime/script_base.gd")
    return script_resource == ADVENTURE_BASE
```

preload requires a constant path and loads while the script is prepared. load/ResourceLoader.load can use a runtime path. A .gd resource is a Script; call new to make its instance. A PackedScene uses instantiate instead.

res:// points to the resource pack/project of the application currently running. In PCA's editor that is the PCA application; in a built game it is the runner and mounted game content. A script record ID is not a file at res://scripts/your_id.gd.

ResourceLoader.exists can check a resource path before load. After loading, check the actual type. For raw packaged text/data, include the file through the export workflow; imported resources may have a converted format.



Figure 15. Imported asset metadata and Godot resource paths are different from the visible asset name.

47. Your own classes and helper objects

```
class RewardData:
    var item_id: String
    var points: int

    func _init(new_item: String, new_points: int) -> void:
        item_id = new_item
        points = new_points

func run() -> Array:
    var reward := RewardData.new("key", 5)
    return [reward.item_id, reward.points]
```

An inner class is useful when a small helper belongs with one script record. A separate .gd file can extend RefCounted for a data/service helper, Node for a scene component, or Resource for serialized editor data.

class_name registers a file-backed Godot class in the Godot project. PCA compiles authored records dynamically; do not depend on a record's class_name turning it into a universally registered file-backed class. Use CallScript/IncludeScript for record-to-record libraries.

In a Godot source project you can create a file such as res://custom/reward_data.gd and preload it. That file must actually be present in the project/export. Writing its path in a string does not create or package it.

48. Inheritance, annotations and editor code

```
class Charge:
    var amount: int = 10
    func description() -> String:
        return "Charge: " + str(amount)

class FullCharge extends Charge:
    func description() -> String:
        return super.description() + " (ready)"

func run() -> String:
    return FullCharge.new().description()
```

A subclass can override a parent method. super calls the parent implementation. Keep an override’s signature compatible with the inherited method; replacing a base helper with a different parameter list can cause parser errors.

Godot annotations include @export, @onready, @tool, @warning_ignore and @rpc. @export creates a Godot Inspector property for a file-backed scene script; it does not automatically add a field to PCA’s custom record inspector. @onready runs at Godot ready time, which is still before PCA binds a newly added record instance.

@tool runs in the Godot editor context. The standalone PCA application is itself a running Godot application, not automatically a Godot EditorPlugin host. Custom PCA designer tools require an actual integration, not merely an @tool line.

49. External files, libraries and packaging

GDScript does not use C-style headers or textual #include. Use extends for inheritance, preload/load for actual Godot files, and PCA's script router for authored records. These are distinct forms of reuse.

You have…	Use…
A PCA native/PCA/Lua/BASIC/visual record	CallScript or IncludeScript with its ID
A .gd file already in the Godot project	preload/load, then new()
A .tscn/.scn PackedScene	load, type-check, instantiate, add_child
A custom Resource	load/ResourceLoader or ResourceSaver with a supported path
An extra addon/native extension	Install/configure in the editable Godot project and export for the target

The normal desktop runner is precompiled. Its game pack includes the documented game data and asset tree; it is not a promise to collect arbitrary res://custom files mentioned in source. For new loose scripts, scenes, addons or native extensions, export an editable Godot project, add the files there, and build through Godot.

Keep extensions in a separate custom folder. Exporting a fresh PCA source project later does not automatically merge your previous custom Godot edits. Copy/reapply them deliberately, and test the exported game on a clean location.

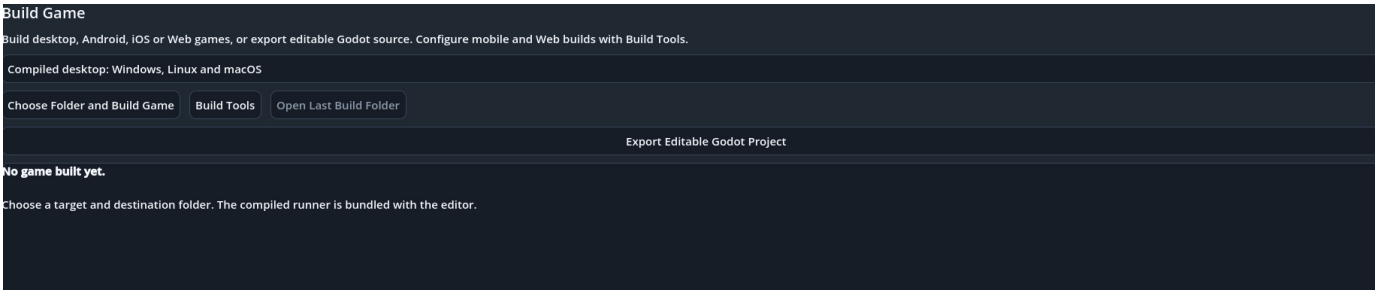

Figure 16. Build Game offers Export Editable Godot Project for work that needs additional Godot files or project configuration.

50. Include a PCA script library

```
func add_points(base: int, bonus: int = 1) -> int:
    return base + bonus

func run() -> int:
    return add_points(10, 5)
```

Put the example in a separate native record utility_gd. An IncludeScript call returns a script-call proxy, not pasted source and not the other record’s actual Node instance.

```
func run() -> int:
    var library: Variant = IncludeScript("utility_gd")
    var total: Variant = await library.Call("add_points", [10, 5])
    library.Set("last_total", total)
    return int(library.Get("last_total", 0))
```

Call invokes a named function; Run invokes run. Get/Set use the included-variable store, separate from the library’s member variables. A one-argument IncludeScript does not execute the library function by itself.

Global.Get(id) provides the same sort of call proxy. Use the public wrapper to inherit caller context, and use unique script IDs. Global is not a native Godot autoload singleton you can assume exists in an unrelated .gd file.

51. Call PCA Script, Lua, BASIC and visual graphs

```
func run() -> Array:
    var pca_result: Variant = await CallScript("utility_pca", "add_points", [10, 5])
    var lua_result: Variant = await CallScript("math_lua", "add_points", [10, 5])
    var basic_result: Variant = await CallScript("math_basic", "AddPoints", [10, 5])
    return [pca_result, lua_result, basic_result]
```

CallScript routes the ID to the correct supported language. The argument array is positional. The optional context dictionary merges with the caller context. Returned plain arrays/dictionaries are the clearest interchange format.

Use the matching language syntax in the helper record. A PCA helper uses function/end; Lua returns a module table; BASIC uses FUNCTION/END FUNCTION. The same helper names from the PCA Script guide can be reused without changing their language.

For a saved visual script, call its record ID through the router/shared run-script action. RunVisualGraph instead accepts the graph data dictionary. It is not a shorthand for an ID lookup.

Native Godot Node/Resource objects are not a portable return contract for every language or save format. Pass IDs and ordinary data unless the receiving route explicitly supports the object type.

52. Expose a native function as a visual node

```
func award(points: int, message: String = "Well done.") -> int:
    var updated: int = int(Variable.score.value) + points
    Variable.score.value = updated
    Log(message)
    return updated
```

1. Create a native script reward_gd with this function. Validate and Apply Changes.
2. Open Functions as Nodes. Add a function definition named award and a useful display name.
3. Add points as int, then message as string, in the same order as the source. Enter defaults that match those types. Check Expose as Action and Visual Node.
4. Click Use Definitions to accept the metadata, then Apply Changes to save the script.
5. In a visual script or supported action list, add Call Script Function, choose the published function, fill in the parameters and optionally select a result variable.

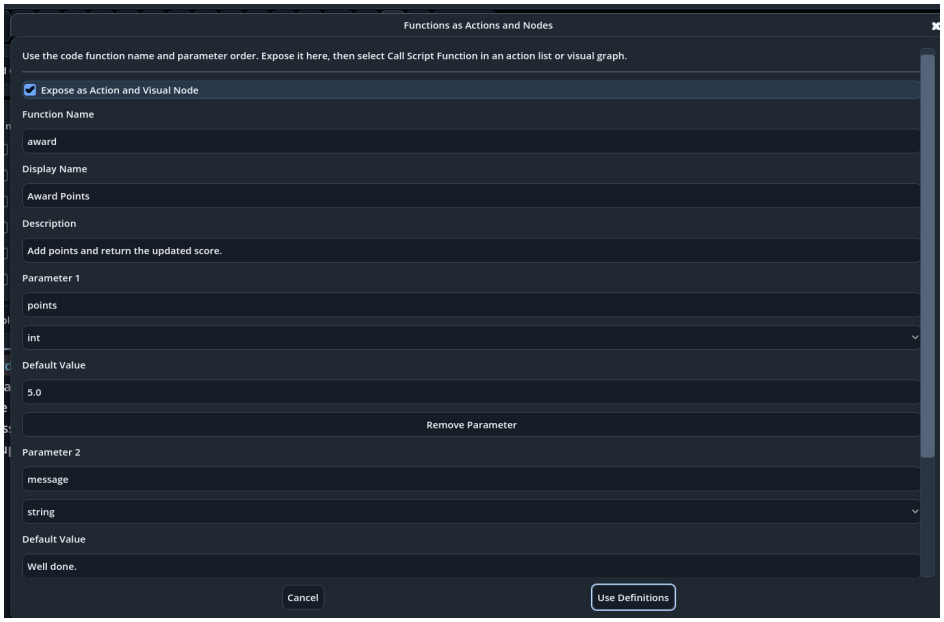

Figure 17. The exposed function definition describes the source function’s parameter contract.



Figure 18. The visual call exposes native GDScript parameters as editable fields.

Use Definitions accepts the definitions you entered. It does not discover functions, create the source body or execute the function. If no definitions were added, there is nothing for the visual picker to display.

53. Debugging in PCA and in Godot

Validate checks the prepared script, including its inherited base and generated record bindings. It is a parser check, not a complete test of every runtime path. A missing record, invalid node or wrong signal argument can still fail during play.

Use Log/print to inspect branches and values. Native breakpoint and the Godot debugger work in an actual Godot debugging session; do not expect PCA Script’s VM watch/step panel to become a full native GDScript debugger.

1. Reproduce the smallest failing interaction. Read the first useful error, including script/function and line information.
2. Log IDs and types before a dynamic call. Check is_instance_valid for nodes and Exists for record proxies.
3. For full native stepping, export/open the editable Godot project and run it under Godot’s debugger. Breakpoints, locals, remote scene tree and the profiler then describe the native execution.
4. Retest after removing diagnostics. Test unsuccessful paths, repeated calls, room changes and save/load.

Generated inheritance/binding lines can affect prepared-source line numbers. Avoid debugging by guessing which invisible line was added; inspect the source preparation/actual debugger location when needed.

> A local error should not be “fixed” by replacing every value with Variant. Use the correct type and lifetime, then keep the dynamic boundary explicit where the API actually returns Variant.

54. Build a clean native extension

A reliable extension has four pieces: a clear entry point, owned native nodes, a public adventure interface, and cleanup. The native button and cube examples show the pattern without changing the engine’s source.

Part	Example decision
Entry	A GUI button calls run, or a subscribed room event starts the feature.
State	Saved puzzle state stays in Variable; transient node references stay in a global script.
Ownership	Native UI is a script child; room visuals are attached to the room and cleaned up on replacement.
Communication	Use PCA actions/events for adventure work and signals for local nodes.
Cleanup	Stop timers, remove temporary nodes, and tolerate room changes invalidating references.
Distribution	Include actual resource files and test the exported game, not only the editor.

Keep the GDScript record small enough to understand. When the project needs a larger file-backed component, put that component in a custom folder in the editable Godot project and leave a small PCA bridge around it.

A native extension is most useful when it adds something concrete: a special inspection view, a custom control, an external data reader or a visual effect that the authored actions do not already cover.

55. Test the complete workflow

Check	Try this
Language	Validate source; check quoted strings, indentation and return types.
Lifetime	Run twice as ordinary and global records; confirm intended persistence.
Events	Check subscription boxes and filter actual payload IDs.
Input	Empty text, whitespace, long text, focus, pointer and controller behavior.
Navigation	Correct room, reachable target, repeated clicks and cancellation.
Native nodes	Stop/restart, room reload, stale references and shared resources.
Saved state	Save before/after a change and reload both.
Export	Run from a clean location with the actual packaged files and renderer.

The rest of this book is a reference: public PCA helpers, all gameplay actions, record/proxy APIs, designer data, shared events and a Godot API map. Use it alongside the small examples rather than memorizing every function before writing an interaction.

> No practice project is required. The screenshots use a private workshop to keep IDs consistent. Adapt the examples to your project’s IDs and choose the designer route whenever it already does the job you need.

PCA helper functions: the full native reference

PCAScriptBase supplies 78 public non-action helpers in this build. They sit alongside Godot’s native functions and the 206 gameplay wrappers in the next section. Signatures below come from the actual base script; names and parameter order are case-sensitive.

A call prefixed with await can suspend until its coroutine/signal work finishes. A synchronous helper can be called normally. Native GDScript has no PCA Script-style when/end syntax or try/catch block.

> The examples name records in the tutorial workshop. Replace them with your own IDs. A helper returning a proxy is not returning the underlying Node, and a diagnostic helper is not an exception system.

Execution, calls and project access

EndGame()

`EndGame(message: String = "", ask_first: bool = false) -> bool`

Request PCA’s end-game route.

Returns: Boolean result from the runtime.

```
await EndGame("Thanks for playing.", true)
```

This is a player-facing game operation, not a GDScript return statement.

QuitToMenu()

`QuitToMenu(message: String = "", ask_first: bool = false) -> bool`

Request a return to the game’s menu route.

Returns: Boolean result.

```
await QuitToMenu("Return to the menu?", true)
```

CallScript()

```
CallScript(script_id: String, function_name: String = "run", arguments: Array = [],
execution_context: Dictionary = {}) -> Variant
```

Call a function through the shared language router, merging additional context with the current caller.

Returns: The called function’s return value.

```
await CallScript("utility_gd", "add_points", [10, 5])
```

Argument array order must match the function. Use unique script IDs across languages.

IncludeScript()

```
IncludeScript(script_id: String) -> RefCounted
```

Obtain a script-call proxy through Global.Get.

Returns: RefCounted call proxy, or null if the registry is unavailable.

```
IncludeScript("utility_gd")
```

This does not paste source or return the other script’s actual Node. Use the proxy’s Call/Run/Get/Set/Emit methods.

GetIncludedVariable()

```
GetIncludedVariable(script_id: String, variable_name: String, fallback: Variant = null) ->
Variant
```

Read router-owned per-script shared data.

Returns: Stored value or fallback.

```
GetIncludedVariable("utility_gd", "last_total", 0)
```

This store is separate from a script’s member variables and ordinary save-game variables.

SetIncludedVariable()

`SetIncludedVariable(script_id: String, variable_name: String, value: Variant) -> bool`

Write router-owned per-script shared data.

Returns: Boolean; requires an existing script ID and nonempty name.

```
SetIncludedVariable("utility_gd", "last_total", 15)
```

Action()

`Action(action_type: String, properties: Dictionary = {}) -> Variant`

Run an exact shared action ID with a properties dictionary.

Returns: That action’s result.

```
await Action("CameraShake", {"duration": 0.3, "power": 0.1})
```

No automatic try/catch behavior is added. The named wrappers use this same route.

RunVisualGraph()

`RunVisualGraph(graph: Dictionary, execution_context: Dictionary = {}) -> Variant`

Execute graph data through the visual runtime.

Returns: Graph return value, where provided.

```
await RunVisualGraph({"nodes": [], "edges": []}, {})
```

This accepts a graph dictionary, not a saved visual-script ID.

EmitEvent()

`EmitEvent(event_name: String, event_context: Dictionary = {}) -> void`

Broadcast an exact event name with merged caller context and additional data.

Returns: Void.

```
await EmitEvent("OnCustomEvent", {"reason": "signal_powered"})
```

For native shared listeners, subscribe to OnCustomEvent and filter a discriminator in its context.

GetProjectProperty()

`GetProjectProperty(property_path: String, default_value: Variant = null) -> Variant`

Read a supported runtime project-data path.

Returns: Value or fallback.

```
GetProjectProperty("meta.title", "Untitled")
```

SetProjectProperty()

`SetProjectProperty(property_path: String, value: Variant) -> bool`

Write a supported runtime project-data path.

Returns: Boolean.

```
SetProjectProperty("meta.title", "Workshop")
```

This is not a request to save the editor document. Live application depends on the target property.

GetSetting()

`GetSetting(setting_path: String, default_value: Variant = null) -> Variant`

Read a setting through the current Settings proxy.

Returns: Value or fallback.

```
GetSetting("musicVolume", 0.7)
```

SetSetting()

`SetSetting(setting_path: String, value: Variant) -> bool`

Write a setting through the current Settings proxy.

Returns: Boolean.

```
SetSetting("musicVolume", 0.5)
```

Use a dedicated action when immediate feature reconfiguration matters.

Log()

`Log(message: Variant) -> void`

Write text to PCA's runtime log, falling back to print when no runtime logger exists.

Returns: Void.

```
Log("Archive unlocked.")
```

Warn()

`Warn(message: Variant) -> void`

Report a Godot warning and a warning-tagged PCA log message.

Returns: Void.

```
Warn("Optional chime is missing.")
```

ScriptError()

`ScriptError(message: Variant) -> void`

Report a Godot error and an error-tagged PCA log message.

Returns: Void.

```
ScriptError("The puzzle setup is incomplete.")
```

This logs; it does not throw a catchable GDScript exception.

PCAAssert()

`PCAAssert(condition: bool, message: String = "PCA3D script assertion failed.") -> bool`

Report a diagnostic when the condition is false.

Returns: The Boolean condition.

```
PCAAssert(5 > 0, "Points must be positive.")
```

It is not an exception boundary. Native assert has separate debug-build behavior.

Breakpoint()

`Breakpoint() -> void`

Reach a native GDScript breakpoint statement.

Returns: Void.

```
Breakpoint()
```

Use in an actual Godot debugging session. It is not PCA Script VM stepping.

Guarded calls and loop helpers

TryCall()

`TryCall(callable: Callable, arguments: Array = [], fallback: Variant = null) -> Dictionary`

Check whether a Callable is valid, then invoke it with an argument array.

Returns: Dictionary with ok, value and error.

```
TryCall(func(value): return value + 1, [4], 0)
```

An invalid Callable returns ok=false and fallback. A runtime error inside a valid Callable is not caught as an exception. The helper does not await a coroutine body.

DoWhile()

`DoWhile(body: Callable, condition: Callable, safety_limit: int = 100000) -> int`

Run a Callable body at least once, then check a Callable condition after each pass.

Returns: Number of iterations.

```
DoWhile(func(i): print(i), func(i): return i < 3)
```

Body receives the zero-based index. Condition receives the incremented count. Safety limit defaults to 100000 and is clamped to at least 1. Calls are synchronous.

ForEach()

`ForEach(collection_value: Variant, callback: Callable) -> int`

Visit array values or dictionary key/value pairs through a Callable.

Returns: Number of entries processed.

```
ForEach(["key", "note"], func(value, index): print(index, value))
```

Array callback gets value,index. Dictionary callback gets key,value,index. Packed arrays are not handled by this helper. It does not await callbacks.

WhileLoop()

`WhileLoop(condition: Callable, body: Callable, safety_limit: int = 100000) -> int`

Check a Callable condition before each Callable body pass.

Returns: Number of iterations.

```
WhileLoop(func(i): return i < 3, func(i): print(i))
```

Both receive the current index. Safety limit prevents unbounded helper iterations; it does not make blocking work asynchronous.

ForLoop()

`ForLoop(count: int, body: Callable) -> int`

Call a body with indices 0 through count-1.

Returns: Number of completed passes.

```
ForLoop(3, func(i): print(i))
```

Negative counts produce no passes. For asynchronous work, prefer an ordinary GDScript for loop with await in its body.

Game queries and interfaces

HasInventory()

`HasInventory(item_id: String, character_id: String = "player", amount: int = 1) -> bool`

Check an inventory item, owner/route and required amount.

Returns: Boolean.

```
HasInventory("key", "player", 1)
```

Shared versus character-owned inventory settings affect the result.

InventoryCount()

`InventoryCount(item_id: String, character_id: String = "player") -> int`

Count copies of an item for a character/inventory route.

Returns: Integer.

```
InventoryCount("key", "player")
```

CurrentScene()

`CurrentScene() -> String`

Read the active room ID.

Returns: String.

```
CurrentScene()
```

This is a PCA room, not get_tree().current_scene.

ActivePlayer()

`ActivePlayer() -> String`

Read the active player character ID.

Returns: String.

```
ActivePlayer()
```

QuestState()

`QuestState(quest_id: String) -> String`

Read a quest’s runtime state.

Returns: State String.

```
QuestState("signal_quest")
```

ObjectiveState()

`ObjectiveState(quest_id: String, objective_id: String) -> String`

Read an objective’s runtime state within its quest.

Returns: State String.

```
ObjectiveState("signal_quest", "find_key")
```

HasVisited()

`HasVisited(scene_id: String, character_id: String = "player") -> bool`

Query recorded character/room visit history.

Returns: Boolean.

```
HasVisited("lab", "ada")
```

GetGUIProperty()

`GetGUIProperty(gui_id: String, property_name: String, default_value: Variant = null) -> Variant`

Read a whole authored GUI property.

Returns: Value or fallback.

```
GetGUIProperty("hud", "visible", false)
```

SetGUIProperty()

`SetGUIProperty(gui_id: String, property_name: String, value: Variant) -> bool`

Write a supported whole-GUI property.

Returns: Boolean.

```
SetGUIProperty("hud", "visible", true)
```

GetGUIElementProperty()

`GetGUIElementProperty(gui_id: String, element_id: String, property_name: String, default_value: Variant = null) -> Variant`

Read a live authored GUI element/control property.

Returns: Value or fallback.

```
GetGUIElementProperty("name_entry", "name_box", "text", "")
```

The GUI must be built/available for a live input value. This is the text-box input route used by the tutorial.

SetGUIElementProperty()

`SetGUIElementProperty(gui_id: String, element_id: String, property_name: String, value: Variant) -> bool`

Write an authored GUI element property through the runtime manager.

Returns: Boolean.

```
SetGUIElementProperty("hud", "topline", "text", "Ready.")
```

A property update may rebuild a control. Preserve in-progress input/selection deliberately.

IfPlayerWasHere()

`IfPlayerWasHere(scene_id: String) -> bool`

Compatibility query for the current player's room visit history.

Returns: Boolean.

```
IfPlayerWasHere("lab")
```

IfPlayerWasNeverHere()

`IfPlayerWasNeverHere(scene_id: String) -> bool`

Complement of the player-visit query.

Returns: Boolean.

```
IfPlayerWasNeverHere("lab")
```

IfVariable()

`IfVariable(variable_id: String, expected_value: Variant = true) -> bool`

Compare a runtime game variable with an expected value.

Returns: Boolean.

```
IfVariable("powered", true)
```

This does not read a same-named local/member variable.

IfInventoryItemWasUsed()

`IfInventoryItemWasUsed(item_id: String) -> bool`

Check the inventory item supplied by the current interaction context.

Returns: Boolean.

```
IfInventoryItemWasUsed("key")
```

A direct Run can have no item context. Possession is a separate query.

IfCharacterMoving()

`IfCharacterMoving(character_id: String = "player") -> bool`

Check whether a character is currently moving.

Returns: Boolean.

```
IfCharacterMoving("player")
```

EndVisualScript()

`EndVisualScript(end_parent: bool = false) -> void`

Request the running visual graph to finish, optionally including its calling graph.

Returns: Void.

```
EndVisualScript(false)
```

This is not a break or return for a native function or unrelated action array.

End()

`End() -> void`

Compatibility no-op: return from this helper.

Returns: Void.

```
End()
```

It does not end the surrounding native function, quit the game or close a GDScript block. Use return in native code.

Time, files and text

NowUnix()

`NowUnix() -> int`

Read whole Unix seconds from the system clock.

Returns: Integer.

```
NowUnix()
```

DateTimeNow()

`DateTimeNow(utc: bool = false) -> Dictionary`

Read current local/UTC calendar components.

Returns: Dictionary.

```
DateTimeNow(true)
```

DateTimeString()

`DateTimeString(utc: bool = false, use_space: bool = true) -> String`

Format the current local/UTC system date/time.

Returns: String.

```
DateTimeString(false, true)
```

UnixToDateTime()

`UnixToDateTime(unix_time: int, _utc: bool = false) -> Dictionary`

Convert Unix seconds to Godot’s UTC calendar dictionary.

Returns: Dictionary.

```
UnixToDateTime(0)
```

The retained UTC parameter is ignored for compatibility. Use native Time APIs for explicit date operations.

RegexSearch()

`RegexSearch(pattern: String, text: String, all_matches: bool = false) -> Variant`

Compile and search a regular expression.

Returns: RegExMatch or null; Array of RegExMatch when all_matches is true.

```
RegexSearch("(KEY)-(\\d+)", "KEY-204")
```

Use get_string/get_start/get_end on the match object. This is different from PCA Script's dictionary result. Invalid patterns log an error and return null/empty.

ReadTextFile()

`ReadTextFile(path: String, default_value: String = "") -> String`

Read the complete text of an accessible file.

Returns: String or fallback.

```
ReadTextFile("user://notes.txt", "")
```

WriteTextFile()

`WriteTextFile(path: String, text: String) -> bool`

Create/overwrite a text file, creating parent directories as needed.

Returns: Boolean success.

```
WriteTextFile("user://notes.txt", "Visited the archive.\n")
```

This helper accepts user:// only and overwrites. Native FileAccess supports additional modes and platform permissions.

ReadJSONFile()

`ReadJSONFile(path: String, default_value: Variant = null) -> Variant`

Read and parse a JSON file.

Returns: Parsed data or fallback.

```
ReadJSONFile("user://settings.json", {"music": 0.7})
```

Empty/invalid input and parsed JSON null select fallback. For distinguishing valid null, use a JSON parser explicitly.

WriteJSONFile()

`WriteJSONFile(path: String, value: Variant, indent: String = "\t") -> bool`

Serialize and overwrite a user-data JSON file.

Returns: Boolean success.

```
WriteJSONFile("user://settings.json", {"music": 0.5}, "  ")
```

FileExists()

`FileExists(path: String) -> bool`

Check whether a file exists.

Returns: Boolean.

```
FileExists("user://settings.json")
```

DirectoryExists()

`DirectoryExists(path: String) -> bool`

Check whether a directory exists after path globalization.

Returns: Boolean.

```
DirectoryExists("user://tutorial")
```

StringLength()

`StringLength(value: Variant) -> int`

Convert the value to text and count its characters.

Returns: Integer.

```
StringLength("Archive")
```

This is not an array-size helper. Use array.size or native len for containers.

StringSplit()

`StringSplit(value: Variant, delimiter: String = " ", allow_empty: bool = true) -> PackedStringArray`

Convert to text and split at a delimiter.

Returns: PackedStringArray.

```
StringSplit("key,note", ",", false)
```

Split has a similar purpose but returns Array[String].

Lower()

`Lower(value: Variant) -> String`

Convert to lowercase text.

Returns: String.

```
Lower("OPEN")
```

Upper()

`Upper(value: Variant) -> String`

Convert to uppercase text.

Returns: String.

```
Upper("archive")
```

Trim()

`Trim(value: Variant) -> String`

Remove whitespace from both ends of text.

Returns: String.

```
Trim("  key  ")
```

Contains()

`Contains(value: Variant, search: Variant, case_sensitive: bool = true) -> bool`

Search text for a substring, optionally ignoring case.

Returns: Boolean.

```
Contains("Archive Door", "door", false)
```

Use Array.has or in for array membership; this helper converts inputs to String.

StartsWith()

`StartsWith(value: Variant, prefix: Variant, case_sensitive: bool = true) -> bool`

Check the start of text.

Returns: Boolean.

```
StartsWith("open archive", "open ")
```

EndsWith()

`EndsWith(value: Variant, suffix: Variant, case_sensitive: bool = true) -> bool`

Check the end of text.

Returns: Boolean.

```
EndsWith("notes.txt", ".txt")
```

Replace()

`Replace(value: Variant, search: Variant, replacement: Variant) -> String`

Replace case-sensitive occurrences of text.

Returns: String.

```
Replace("red door", "red", "blue")
```

Split()

`Split(value: Variant, delimiter: String = " ", allow_empty: bool = true) -> Array[String]`

Split text into ordinary typed String entries.

Returns: Array[String].

```
Split("key,note", ",", false)
```

Join()

`Join(values: Array, delimiter: String = "") -> String`

Convert array entries to text and join them.

Returns: String.

```
Join(["key", "note"], ", ")
```

Math, randomness and data conversion

Clamp()

`Clamp(value: float, minimum: float, maximum: float) -> float`

Limit a float to a minimum/maximum.

Returns: Float.

```
Clamp(120.0, 0.0, 100.0)
```

Lerp()

```
Lerp(from_value: float, to_value: float, weight: float) -> float
```

Interpolate two float values.

Returns: Float.

```
Lerp(10.0, 20.0, 0.25)
```

This wrapper uses lerpf; use native vector/color methods for those types.

RandomInteger()

```
RandomInteger(minimum: int, maximum: int) -> int
```

Choose an integer including both endpoints.

Returns: Integer.

```
RandomInteger(1, 6)
```

Randomize()

```
Randomize(seed_value: int = -1) -> void
```

Reseed global randomness, or use normal randomization when the supplied seed is negative.

Returns: Void.

```
Randomize(1234)
```

A seed affects the shared random sequence. A native RandomNumberGenerator can keep a separate sequence.

RandomFloat()

```
RandomFloat(minimum: float = 0.0, maximum: float = 1.0) -> float
```

Choose a float within the supplied interval.

Returns: Float.

```
RandomFloat(0.25, 0.75)
```

RandomInt()

`RandomInt(minimum: int, maximum: int) -> int`

Alias route for RandomInteger.

Returns: Integer.

```
RandomInt(1, 6)
```

RandomNumber()

`RandomNumber(minimum: float = 0.0, maximum: float = 1.0) -> float`

Alias route for RandomFloat.

Returns: Float.

```
RandomNumber(0.0, 1.0)
```

Choose()

`Choose(values: Array, fallback: Variant = null) -> Variant`

Pick an array entry, or a fallback for an empty array.

Returns: Selected value or fallback.

```
Choose(["Locked.", "Still locked."], "No reply.")
```

Shuffle()

`Shuffle(values: Array) -> Array`

Make a deeply duplicated array and shuffle the copy.

Returns: Array.

```
Shuffle(["red", "green", "blue"])
```

Contained Object references are still objects, not independently cloned scenes.

Distance()

`Distance(first: Variant, second: Variant) -> float`

Measure matching Vector2/Vector3 or integer-vector values, or a numeric absolute difference.

Returns: Float; 0 for unsupported pairs.

```
Distance(Vector3.ZERO, Vector3(3, 0, 4))
```

It does not automatically look up record/node positions from a proxy.

Format()

`Format(template: Variant, values: Variant = {}) -> String`

Replace dictionary {key}, array {0}, or scalar {0}/{value} placeholders.

Returns: String.

```
Format("Score: {value}", 25)
```

No embedded expression evaluation or translation registration is implied.

ToJSON()

`ToJSON(value: Variant, indent: String = "") -> String`

Serialize ordinary data to JSON text.

Returns: String.

```
ToJSON({"open": true, "score": 5}, "  ")
```

FromJSON()

`FromJSON(text_value: Variant, fallback: Variant = null) -> Variant`

Parse JSON text with an explicit parser and fallback on failure.

Returns: Parsed value or fallback.

```
FromJSON('{"open":true}', {})
```

Valid JSON null remains null. Native vectors, resources and nodes need an explicit representation for portable JSON.

IsValid()

`IsValid(value: Variant) -> bool`

Check non-nullness, and use is_instance_valid for Object values.

Returns: Boolean.

```
IsValid(0)
```

A valid record-proxy object can still refer to a missing record. Use proxy.Exists to check the record itself.

TypeOf()

`TypeOf(value: Variant) -> String`

Return PCA-friendly type text for a native value.

Returns: String such as integer, number, array, vector3 or object.

```
TypeOf([1, 2])
```

A PCAScriptRecordProxy is an Object, so it is normally reported as object. This differs from the PCA Script VM’s record value type.

Gameplay actions: all 206 native wrappers

Every normal shared gameplay action is available as a PascalCase PCAScriptBase method with zero or one properties Dictionary. The options match the shared action catalog. The exact action ID also works through Action.

```
await CameraShake({"duration": 0.5, "power": 0.25})
await Action("CameraShake", {"duration": 0.5, "power": 0.25})
```

Each entry lists all standard options, defaults, timing and a practical use. Patterns show a few important fields, not a complete ready-to-run game. Replace placeholder assets/IDs and configure the underlying designer records.

> The returned value is action-specific. A false return or ok:false dictionary does not automatically throw an exception in GDScript or cancel an unrelated action list. Check a meaningful result explicitly.

Gameplay: Variables, rewards and reusable logic

Keep puzzle facts in variables, then use a visual or text script when a decision is needed. A list is an ordered set of instructions; it does not gain a branch just because an action returns false.

SetVariable()

Set Variable

Assign a value to a runtime game variable. This replaces its previous value and can create the runtime entry if it does not exist.

Option key / field	Meaning and input	Default
Variable `name`	Choose a game variable name. Custom values are accepted.	Blank
Value `value`	A value of the type required by the selected variable, operation or property.	true

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SetVariable({"name": "powered", "value": true})
```

In use. Set powered to true when the lever is used. Use a Boolean value, not the text "true". In the room/object inspector, choose Change Variable with Operation set to Set instead.

CreateVariable()

Create Variable

Create a variable only when that name is absent. It is an initializer, so calling it again does not reset an existing value.

Option key / field	Meaning and input	Default
Variable ID `name`	Text.	Blank
Type `variableType`	Choose: string, boolean, number, array.	"string"
Initial Value `value`	A value of the type required by the selected variable, operation or property.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await CreateVariable({"name": "powered", "variableType": "boolean", "value": true})
```

In use. Create visits with a numeric value of 0, then increment it on room entry. Define long-lived puzzle variables in the Variable Designer so their starting values are easy to find. The variableType field is descriptive; do not rely on it to convert bad input.

ChangeVariable()

Change Variable

Set or modify a variable using set, arithmetic, append or toggle. The operation determines how Value is used.

Option key / field	Meaning and input	Default
Variable `name`	Choose a game variable name.	Blank
Operation `operation`	Choose: set, add, subtract, multiply, divide, append, toggle.	"set"
Value `value`	A value of the type required by the selected variable, operation or property.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await ChangeVariable({"name": "powered", "operation": "set", "value": true})
```

In use. Choose score, Add, 5 for a five-point reward. Toggle suits a repeatable switch; Set is safer when an event may run twice. Keep arithmetic numeric and avoid dividing by zero.

AddScore()

Add Score

Increase the runtime variable named score by the given integer amount.

Option key / field	Meaning and input	Default
Amount `amount`	Number of points to add to score.	1

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await AddScore({"amount": 1})
```

In use. Award 5 after collecting a unique key. This action does not remember that the key was already collected. Put it behind the same one-time condition as the reward. In a native inspector, Change Variable / score / Add is the direct equivalent.

UnlockHint()

Unlock Hint Topic / Answer

Unlock a numbered answer within an existing hint topic.

Option key / field	Meaning and input	Default
Topic ID `topicId`	Text.	Blank
Answer Number `answerNumber`	Whole number.	1

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await UnlockHint({"topicId": "", "answerNumber": 1})
```

In use. Unlock answer 2 for the archive topic after the player has examined the door. Match the topic ID and answer numbering used by the hint system; an invented ID does not create a complete hint topic.

RunVisualScript()

Run Visual Script

Run a selected script and wait for that call. Despite its label, the picker can select the supported text-script languages as well as visual graphs.

Option key / field	Meaning and input	Default
Script `scriptId`	Choose a visual or supported text script.	Blank

Timing. Waits for the selected script call.

```
await RunVisualScript({"scriptId": "utility_gd"})
```

In use. Put the door's inventory check in open_door, then call that script from the door's Use response. Keep the conditional logic in that one place instead of copying it into several event lists.

CallScriptFunction()

Call Script Function

Call a function published through Functions as Nodes, pass named parameters, and optionally store its return value.

Option key / field	Meaning and input	Default
Function `functionKey`	Text/reference.	Blank
Parameters `parameters`	A typed value or valid JSON.	{}
Result Variable `resultVariable`	Name of the game variable that receives the returned value.	Blank

Timing. Waits for the published function call.

```
await CallScriptFunction({
    "functionKey": "scripts/reward_gd/award",
    "parameters": {"points": 5, "message": "Well done."},
    "resultVariable": ""
})
```

In use. Publish a reward function with points and message parameters. Choose it here, enter those parameters, and store the result in reward_ok. The function must be defined and published first; this action does not infer a function from an arbitrary name.

Gameplay: Inventory and two-object interactions

An inventory definition describes an item. Giving an item changes a bag. Removing its scene model is a separate instruction. Decide whether the project uses shared or character-specific inventory before writing transfers.

AddInventory()

Add Inventory

Add copies of an existing inventory item through the runtime's active inventory route.

Option key / field	Meaning and input	Default
Item `itemId`	Choose an inventory definition ID.	Blank
Count `count`	Number of copies or repetitions. Use a whole number.	1

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await AddInventory({"itemId": "key", "count": 1})
```

In use. Add one Archive Key after a successful pickup. This is not a command to import a model, and it does not hide the key on the workbench. Use Give Inventory in the room/object inspector when this entry is not listed there.

RemoveInventory()

Remove Inventory

Remove copies of an item through the current inventory route.

Option key / field	Meaning and input	Default
Item `itemId`	Choose an inventory definition ID.	Blank
Count `count`	Number of copies or repetitions. Use a whole number.	1

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await RemoveInventory({"itemId": "key", "count": 1})
```

In use. Remove one fuse after it has successfully been installed. Test the condition first, then remove it. Removing a missing item is not a substitute for a Has Inventory branch, and the next action in an ordinary list will still run.

ChangeInventoryQuantity()

Change Inventory Quantity

Apply a signed quantity change to an item for a character.

Option key / field	Meaning and input	Default
Player / Character `characterId`	Choose a Character Designer ID; player is a runtime alias where supported.	Blank
Inventory Item `itemId`	Choose an inventory definition ID.	Blank
Change By `amount`	Signed quantity change: positive adds, negative removes.	1

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await ChangeInventoryQuantity({"characterId": "player", "itemId": "key", "amount": 1})
```

In use. Use amount -1 to consume a battery, or +3 to grant three coins. This is a change by an amount, not an instruction to set an absolute total. Choose the same owner that the inventory GUI is displaying.

TransferInventory()

Change Inventory from Player to Player

Move a quantity from one character's inventory to another.

Option key / field	Meaning and input	Default
From Player `fromCharacterId`	Choose a Character Designer ID; player is a runtime alias where supported.	Blank
To Player `toCharacterId`	Choose a Character Designer ID; player is a runtime alias where supported.	Blank
Inventory Item `itemId`	Choose an inventory definition ID.	Blank
Quantity `count`	Number of copies or repetitions. Use a whole number.	1

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await TransferInventory({"fromCharacterId": "", "toCharacterId": "", "itemId": "key"})
```

In use. Transfer the key from Ivo to Elias in a game with individual bags. Check the source character and quantity before playing a handover animation. Changing which character is controlled is not the same thing as transferring an item.

ExecutePreposition()

Execute Two-Object Command

Perform a two-target verb interaction using a source, target and verb.

Option key / field	Meaning and input	Default
Verb `verbId`	Choose a Verb Designer ID.	Blank
First Object (entity:id or inventory:id) `sourceId`	Text.	Blank
Second Object (entity:id or inventory:id) `targetId`	Text.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await ExecutePreposition({"verbId": "", "sourceId": "", "targetId": "archive_door"})
```

In use. For Use Key On Door, supply the source item/record reference, the door reference, and the Use verb. Configure the verb's preposition rules and the matching interaction first. A normal scene entity ID and an inventory reference are different kinds of target.

CancelPreposition()

Cancel Two-Object Command

Cancel the currently assembled two-object interaction.

No options in the standard catalog schema.

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await CancelPreposition({})
```

In use. Let a Cancel button clear Use Key On ... when the player changes their mind. It clears the pending combination, rather than deleting the item or disabling the verb permanently.

Gameplay: Speech, messages, voice and dialogue

Use a message for narration, a speech action for a speaking character, and Start Dialogue for a conversation graph. These are different routes to text. Typewriter controls are shared by the message and speech actions and by newly added dialogue lines and choices.

ShowMessage()

Show Message

Display narration or a short message for a duration, optionally blocking player input.

Option key / field	Meaning and input	Default
Text `text`	Text.	Blank
Duration (seconds) `duration`	Time in seconds for this action. Read its timing note before assuming it waits.	1.8
Block Player Input `blockInput`	Block normal player input while the message is presented.	false
Voice Audio `voiceAssetId`	Choose an imported audio asset. May be blank.	Blank
Voice Key (optional) `voiceKey`	Optional localization/voice lookup key.	Blank
Typewriter Effect `typewriter`	Reveal text progressively instead of displaying it all at once.	false
Typing Speed (characters/second) `typewriterSpeed`	Text reveal rate in characters per second. Range: 1.0 to 240.0.	30.0
Typing Sound `typewriterSound`	Choose: default, none, custom.	"default"
Typing Sound Asset `typewriterSoundAssetId`	The imported audio used when the typing sound is custom. May be blank.	Blank

Timing. Waits for the message routine.

```
await ShowMessage({"text": "The mechanism is ready.", "duration": 1, "blockInput": false})
```

In use. Show "The signal is on." for 2.2 seconds. Block Player Input affects movement/interaction while the message is up; the action call itself still waits for the message routine. Keep long conversations in the Dialogue Designer.

Typing sound: default, none or custom. Custom uses typewriterSoundAssetId; speed is characters per second.

Say()

Say

Speak as the chosen character. With Wait for Continue off, speech uses timed advancement; with it on, the player must continue.

Option key / field	Meaning and input	Default
Speaker `speakerId`	Choose a Character Designer ID; player is a runtime alias where supported.	Blank
Text `text`	Text.	Blank
Wait for Continue `waitForContinue`	Wait for the player to continue the speech instead of using timed advancement.	false
Voice Audio `voiceAssetId`	Choose an imported audio asset. May be blank.	Blank
Voice Key (optional) `voiceKey`	Optional localization/voice lookup key.	Blank
Typewriter Effect `typewriter`	Reveal text progressively instead of displaying it all at once.	false
Typing Speed (characters/second) `typewriterSpeed`	Text reveal rate in characters per second. Range: 1.0 to 240.0.	30.0
Typing Sound `typewriterSound`	Choose: default, none, custom.	"default"
Typing Sound Asset `typewriterSoundAssetId`	The imported audio used when the typing sound is custom. May be blank.	Blank

Timing. Waits for the speech routine; manual continuation is optional.

```
await Say({"speakerId": "elias", "text": "The mechanism is ready.", "waitForContinue": false})
```

In use. Let Elias explain the machine, then run the next instruction. A timed line without a custom duration uses the runtime's short speech timing. Voice and typewriter settings affect presentation; the speaker still needs a valid scene or character definition.

Typing sound: default, none or custom. Custom uses typewriterSoundAssetId; speed is characters per second.

SetVoiceSpeaker()

Set Voice Speaker

Choose the default character whose mouth follows separately played voice audio or message voice.

Option key / field	Meaning and input	Default
Character `characterId`	Choose a Character Designer ID; player is a runtime alias where supported. May be blank.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SetVoiceSpeaker({"characterId": "player"})
```

In use. Select Elias before Play Voice. Clear the selection when that default is no longer wanted. Say and Character Say already name their own speaker; configure the character's mouth and portrait mapping in the Character Designer first.

SetLipSyncEnabled()

Enable Lip Sync

Enable or disable lip sync for a character at runtime.

Option key / field	Meaning and input	Default
Character `characterId`	Choose a Character Designer ID; player is a runtime alias where supported.	Blank
Enabled `enabled`	Explicit on/off state for the feature or response.	true

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SetLipSyncEnabled({"characterId": "player", "enabled": true})
```

In use. Disable lip sync for a silent disguise sequence, then enable it afterward. This is a runtime switch, not a replacement for defining visemes, portrait mouth frames or blend shapes. Turning it off restores the resting face.

PlayVoice()

Play Voice

Play voice audio selected by asset or localization key, optionally waiting for it to finish.

Option key / field	Meaning and input	Default
Voice Audio `voiceAssetId`	Choose an imported audio asset. May be blank.	Blank
Voice Key (optional) `voiceKey`	Optional localization/voice lookup key.	Blank
Text `text`	Text.	Blank
Wait for Voice `wait`	Whether this action should wait for its own effect to finish.	false

Timing. Waits when Wait for Voice is on.

```
await PlayVoice({"voiceAssetId": "", "voiceKey": "", "text": "The mechanism is ready."})
```

In use. Play a radio line while the scene remains visible. If the next instruction must follow the recording, enable Wait for Voice. Text helps the built-in text/voice mouth workflow; it does not automatically turn this into a full dialogue conversation.

PauseVoice()

Pause Voice

Pause the current voice and its mouth timing.

No options in the standard catalog schema.

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await PauseVoice({})
```

In use. Pause a recording while a message is being inspected. A conversation awaiting that recording remains in progress; use Stop Voice if the recording is meant to be cancelled instead.

ResumeVoice()

Resume Voice

Resume paused voice playback and the shared scene/portrait mouth timing.

No options in the standard catalog schema.

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await ResumeVoice({})
```

In use. Pair it with Pause Voice. It resumes the current recording rather than choosing a new asset. Calling it without a paused voice does not start a new line.

SeekVoice()

Seek Voice

Move the current voice playback position to a time in seconds.

Option key / field	Meaning and input	Default
Seconds `seconds`	Number; decimals allowed.	0.0

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SeekVoice({"seconds": 1})
```

In use. Seek to 3.0 seconds while checking a line's mouth cues. Test the associated text and timing track at the new position, especially when a conversation is also waiting for audio completion.

StopVoice()

Stop Voice

Stop the current voice and reset its mouth presentation.

No options in the standard catalog schema.

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await StopVoice({})
```

In use. Use it when an interrupted radio message should end immediately. It is voice-specific; sound effects and music use their own stop actions.

StartDialogue()

Start Dialogue

Start a Dialogue Designer conversation and wait for that dialogue session to finish.

Option key / field	Meaning and input	Default
Dialogue `dialogueId`	Choose a Dialogue Designer ID.	Blank

Timing. Waits for the dialogue session.

```
await StartDialogue({"dialogueId": ""})
```

In use. On Elias's Talk verb, choose Ask Elias. The dialogue record decides its start node, choices and flow. Do not start the same dialogue again in OnDialogueStarted, which can create nested or repeated conversations.

EndDialogue()

End Dialogue

End the active dialogue session.

No options in the standard catalog schema.

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await EndDialogue({})
```

In use. Use it when a scripted event cuts a conversation short. It does not erase the authored dialogue, its choices or the saved read history. Test any actions that were supposed to follow a normal ending.

AddDialogueLine()

Add Dialogue Line

Insert or update a line node in a runtime dialogue graph, including speaker, text, next node, voice and typewriter options.

Option key / field	Meaning and input	Default
Dialogue `dialogueId`	Choose a Dialogue Designer ID.	Blank
New Node ID `nodeId`	Text.	Blank

Option key / field	Meaning and input	Default
Speaker `speakerId`	Choose a Character Designer ID; player is a runtime alias where supported.	Blank
Text `text`	Text.	Blank
Next Node `next`	Choose a node in that dialogue. May be blank. Custom values are accepted.	Blank
Return to previous node `returnToPrevious`	Return through dialogue history rather than relying only on a fixed next link.	true
Voice Audio `voiceAssetId`	Choose an imported audio asset. May be blank.	Blank
Voice Key (optional) `voiceKey`	Optional localization/voice lookup key.	Blank
Typewriter Effect `typewriter`	Reveal text progressively instead of displaying it all at once.	false
Typing Speed (characters/second) `typewriterSpeed`	Text reveal rate in characters per second. Range: 1.0 to 240.0.	30.0
Typing Sound `typewriterSound`	Choose: default, none, custom.	"default"
Typing Sound Asset `typewriterSoundAssetId`	The imported audio used when the typing sound is custom. May be blank.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await AddDialogueLine({"dialogueId": "", "nodeId": "", "speakerId": "elias"})
```

In use. Add an answer node named signal_answer, then point a choice at it. Use a unique stable node ID. Return to previous node is enabled by default, so check that behavior before also relying on a fixed Next link.

Typing sound: default, none or custom. Custom uses typewriterSoundAssetId; speed is characters per second.

AddDialogueChoiceNode()

Add Dialogue Choice Node

Create a choice node in an existing dialogue.

Option key / field	Meaning and input	Default
Dialogue `dialogueId`	Choose a Dialogue Designer ID.	Blank
New Node ID `nodeId`	Text.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await AddDialogueChoiceNode({"dialogueId": "", "nodeId": ""})
```

In use. Create extra_topics, then add choices to that node with Add Dialogue Choice. Creating the choice node alone does not make the player visit it: connect an existing node to it or otherwise change the graph flow.

RemoveDialogueNode()

Remove Dialogue Node

Remove a node from the runtime dialogue graph.

Option key / field	Meaning and input	Default
Dialogue `dialogueId`	Choose a Dialogue Designer ID.	Blank
Dialogue Node ID `nodeId`	Choose a node in that dialogue. Custom values are accepted.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await RemoveDialogueNode({"dialogueId": "", "nodeId": ""})
```

In use. Remove a temporary topic after its story purpose is finished. Check every Next link and choice that points to it; removing a destination can leave a conversation with nowhere useful to go.

AddDialogueChoice()

Add Dialogue Choice

Add or update an option inside a choice node.

Option key / field	Meaning and input	Default
Dialogue `dialogueId`	Choose a Dialogue Designer ID.	Blank
Choice Node ID `nodeId`	Choose a choice node in that dialogue. Custom values are accepted.	Blank
New Choice ID `choiceId`	Text.	Blank
Text `text`	Text.	Blank
Next Node `next`	Choose a node in that dialogue. May be blank. Custom values are accepted.	Blank
Enabled `enabled`	Explicit on/off state for the feature or response.	true
Always Allow Repeat `repeatable`	Allow this dialogue choice to be selected repeatedly under its read policy.	false
Voice Audio `voiceAssetId`	Choose an imported audio asset. May be blank.	Blank
Voice Key (optional) `voiceKey`	Optional localization/voice lookup key.	Blank
Typewriter Effect `typewriter`	Reveal text progressively instead of displaying it all at once.	false
Typing Speed (characters/second) `typewriterSpeed`	Text reveal rate in characters per second. Range: 1.0 to 240.0.	30.0
Typing Sound `typewriterSound`	Choose: default, none, custom.	"default"
Typing Sound Asset `typewriterSoundAssetId`	The imported audio used when the typing sound is custom. May be blank.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await AddDialogueChoice({"dialogueId": "", "nodeId": "", "choiceId": ""})
```

In use. Add a "The signal is working" option after powered becomes true. Choose the dialogue, choice node, new choice ID and next node. Repeatable controls repeated availability; it is separate from whether the choice is currently enabled.

Typing sound: default, none or custom. Custom uses typewriterSoundAssetId; speed is characters per second.

RemoveDialogueChoice()

Remove Dialogue Choice

Remove one option identified by its choice ID from a choice node.

Option key / field	Meaning and input	Default
Dialogue `dialogueId`	Choose a Dialogue Designer ID.	Blank
Choice Node ID `nodeId`	Choose a choice node in that dialogue. Custom values are accepted.	Blank
Choice ID `choiceId`	Choose an option ID in that choice node. Custom values are accepted.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await RemoveDialogueChoice({"dialogueId": "", "nodeId": "", "choiceId": ""})
```

In use. Remove an introductory question once it no longer makes sense. For a choice that may return later, Set Dialogue Choice Enabled is usually easier to reverse than deleting and recreating it.

SetDialogueChoiceEnabled()

Enable / Disable Dialogue Choice

Enable or disable an existing choice without removing its record.

Option key / field	Meaning and input	Default
Dialogue `dialogueId`	Choose a Dialogue Designer ID.	Blank
Choice Node ID `nodeId`	Choose a choice node in that dialogue. Custom values are accepted.	Blank
Choice ID `choiceId`	Choose an option ID in that choice node. Custom values are accepted.	Blank
Enabled `enabled`	Explicit on/off state for the feature or response.	true

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SetDialogueChoiceEnabled({"dialogueId": "", "nodeId": "", "choiceId": ""})
```

In use. Enable the archive question after examining the door. Select the exact choice node and choice ID. Its visible/disabled presentation also depends on the dialogue read and choice-display settings.

ResetDialogueHistory()

Reset Dialogue History

Clear remembered read/use history for a dialogue, node or choice scope.

Option key / field	Meaning and input	Default
Dialogue `dialogueId`	Choose a Dialogue Designer ID. May be blank.	Blank
Dialogue Node ID `nodeId`	Choose a node in that dialogue. May be blank. Custom values are accepted.	Blank
Choice ID `choiceId`	Choose an option ID in that choice node. May be blank. Custom values are accepted.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await ResetDialogueHistory({"dialogueId": "", "nodeId": "", "choiceId": ""})
```

In use. Reset one repeatable hint conversation during a new puzzle phase. Blank scopes can broaden the reset; select the narrowest scope you need so unrelated conversations do not suddenly become unread.

Gameplay: Characters, movement and animation

Movement commands act on characters that the runtime can resolve. A destination can be a target record or coordinates. The room picker helps resolve that destination; it is not a promise that the character will walk through unloaded rooms.

CharacterWalkTo()

Character Walk To

Walk a character to the chosen target or position, waiting for the movement call.

Option key / field	Meaning and input	Default
Character `characterId`	Choose a Character Designer ID; player is a runtime alias where supported.	Blank
Object Room `sceneId`	Choose a scene ID.	Blank
Object (optional) `targetId`	Choose an entity in the selected scene. May be blank.	Blank
XYZ `position`	Destination/placement coordinates as [X, Y, Z].	[0.0, 0.0, 0.0]
Walking Animation `animation`	Choose a clip/state available to that entity. May be blank.	Blank
Speed `speed`	Movement speed; 0 uses the normal movement speed where supported.	0

Timing. Waits for the movement routine.

```
await CharacterWalkTo({"characterId": "player", "sceneId": "lab", "targetId": "archive_door"})
```

In use. Walk Elias to an approach marker beside the workbench. Use a reachable marker instead of the middle of a solid door. The current action bridge uses the character's movement logic; the Animation field is not a reliable override for its walk-state setup.

ClearCharacterWaypoints()

Character Waypoints: Clear Queue

Empty a character's queued destinations.

Option key / field	Meaning and input	Default
Character `characterId`	Choose a Character Designer ID; player is a runtime alias where supported.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await ClearCharacterWaypoints({"characterId": "player"})
```

In use. Clear an NPC's old patrol before adding a new route. This edits the queue, rather than providing a visible destination or starting a new walk by itself.

AddCharacterWaypoint()

Character Waypoints: Add Waypoint

Append one destination to a character's waypoint queue.

Option key / field	Meaning and input	Default
Character `characterId`	Choose a Character Designer ID; player is a runtime alias where supported.	Blank
Waypoint Room `sceneId`	Choose a scene ID.	Blank
Waypoint / Object (optional) `waypointId`	Choose an entity in the selected scene. May be blank.	Blank
XYZ when no waypoint `position`	Destination/placement coordinates as [X, Y, Z].	[0.0, 0.0, 0.0]
Ignore Navigation Mesh `ignoreNavigation`	Use the direct route instead of following normal navigation constraints.	false

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await AddCharacterWaypoint({"characterId": "player", "sceneId": "lab", "waypointId": ""})
```

In use. Add three named markers around the room, then call Character Walk Waypoints. Ignore Navigation permits a direct route; keep it off for ordinary floor walking so obstacles and walkable areas remain meaningful.

CharacterWalkWaypoints()

Character Waypoints: Walk Queue

Run the queued waypoint route for a character.

Option key / field	Meaning and input	Default
Character `characterId`	Choose a Character Designer ID; player is a runtime alias where supported.	Blank
Speed `speed`	Movement speed; 0 uses the normal movement speed where supported.	0

Timing. Runs the queued movement route.

```
await CharacterWalkWaypoints({"characterId": "player", "speed": 0})
```

In use. After clearing and adding the patrol markers, choose Elias and start the queue. Building the queue and walking it are separate steps. A blocked marker can prevent the intended route from completing.

CharacterWalkPath()

Character Walk Through Waypoints

Walk through an array of waypoint references, optionally ignoring navigation.

Option key / field	Meaning and input	Default
Character `characterId`	Choose a Character Designer ID; player is a runtime alias where supported.	Blank
Default Waypoint Room `sceneId`	Choose a scene ID.	Blank
Waypoint IDs / Positions `waypointIds`	JSON array of waypoint references in travel order.	[]
Speed `speed`	Movement speed; 0 uses the normal movement speed where supported.	0
Ignore Navigation Mesh `ignoreNavigation`	Use the direct route instead of following normal navigation constraints.	false

Timing. Runs the selected movement route.

```
await CharacterWalkPath({"characterId": "player", "sceneId": "lab", "waypointIds": []})
```

In use. Supply a JSON array of marker IDs in route order, then choose the walking speed. Use local, reachable markers for a first test. A list of display labels is not a list of the stored IDs.

CharacterTeleport()

Character Instant Teleport To

Move a character directly to a target or coordinates without walking there.

Option key / field	Meaning and input	Default
Character `characterId`	Choose a Character Designer ID; player is a runtime alias where supported.	Blank
Object Room `sceneId`	Choose a scene ID.	Blank
Object (optional) `targetId`	Choose an entity in the selected scene. May be blank.	Blank
XYZ `position`	Destination/placement coordinates as [X, Y, Z].	[0.0, 0.0, 0.0]

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await CharacterTeleport({
    "characterId": "player",
    "sceneId": "lab",
    "targetId": "archive_door"
})
```

In use. Place the player at a staged position before a scene reveal. Check the floor height and collider clearance; teleporting into a wall is still possible. Use Change Scene for a player room transition and Move NPC To Room for an NPC's room assignment.

MoveNPCToRoom()

Move NPC to a Different Room

Change an NPC's room assignment and arrival position or spawn.

Option key / field	Meaning and input	Default
Character `characterId`	Choose a Character Designer ID; player is a runtime alias where supported.	Blank
Room `sceneId`	Choose a scene ID.	Blank
Spawn (optional) `spawnId`	Choose an arrival spawn in the destination scene. May be blank.	Blank
XYZ when no spawn `position`	Destination/placement coordinates as [X, Y, Z].	[0.0, 0.0, 0.0]

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await MoveNPCToRoom({"characterId": "player", "sceneId": "lab", "spawnId": ""})
```

In use. Send Elias to the archive after the puzzle. This is a world-state change rather than visible walking between rooms. Confirm that the destination has the intended spawn and that a scene does not already contain a conflicting copy of the NPC.

FollowCharacter()

Follow Character

Start one character following another with a chosen spacing and speed.

Option key / field	Meaning and input	Default
Character `characterId`	Choose a Character Designer ID; player is a runtime alias where supported.	Blank
Character to Follow `targetCharacterId`	Choose a Character Designer ID; player is a runtime alias where supported.	Blank
Distance Behind `distance`	Number; decimals allowed.	0
Speed `speed`	Movement speed; 0 uses the normal movement speed where supported.	0

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await FollowCharacter({"characterId": "player", "targetCharacterId": "", "distance": 0})
```

In use. Have Elias follow the player at about 1.5 units. It starts a continuing behavior; the next list instruction need not wait for a final destination. Stop the follower before a staged conversation if both actors must hold their marks.

StopFollowCharacter()

Stop Follow Character

Stop the chosen character's follow behavior.

Option key / field	Meaning and input	Default
Character `characterId`	Choose a Character Designer ID; player is a runtime alias where supported.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await StopFollowCharacter({"characterId": "player"})
```

In use. Call it before walking an NPC to a specific scene marker. It stops the following relationship; it does not remove the character or disable future movement commands.

FaceCharacter()

Face Character

Turn one character toward another.

Option key / field	Meaning and input	Default
Character `characterId`	Choose a Character Designer ID; player is a runtime alias where supported.	Blank
Face Character `targetCharacterId`	Choose a Character Designer ID; player is a runtime alias where supported.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await FaceCharacter({"characterId": "player", "targetCharacterId": ""})
```

In use. Face Elias toward the player before his greeting. Select the acting character first and the target second. Turning alone does not bring them closer or guarantee a clear line of sight.

FaceObject()

Face Object

Turn a character toward a scene object.

Option key / field	Meaning and input	Default
Character `characterId`	Choose a Character Designer ID; player is a runtime alias where supported.	Blank
Object Room `sceneId`	Choose a scene ID.	Blank
Object `targetId`	Choose an entity in the selected scene.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await FaceObject({"characterId": "player", "sceneId": "lab", "targetId": "archive_door"})
```

In use. Make the player face the signal lamp after using the lever. The target's position determines the facing direction; a large object's origin may not be the point you intended.

FaceCoordinates()

Face Coordinates

Turn a character toward an explicit world position.

Option key / field	Meaning and input	Default
Character `characterId`	Choose a Character Designer ID; player is a runtime alias where supported.	Blank
XYZ `position`	Destination/placement coordinates as [X, Y, Z].	[0.0, 0.0, 0.0]

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await FaceCoordinates({"characterId": "player", "position": [0.0, 0.0, 0.0]})
```

In use. Use a position near the door handle when the door model's origin is unhelpful. In a 3D scene, provide X, Y and Z. This is a facing target, not a rotation triplet.

HidePlayerCharacter()

Hide Player Character

Hide or reveal the active player's visual representation.

Option key / field	Meaning and input	Default
Hidden `hidden`	Whether the quest or player visual should be hidden for this action.	true

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await HidePlayerCharacter({"hidden": true})
```

In use. Hide the player during a first-person inspection and reveal them on exit. Hiding the model is not an input lock and does not, by itself, stop the player moving.

SetPlayerInputEnabled()

Enable / Disable Player Input

Enable or disable player control input.

Option key / field	Meaning and input	Default
Enabled `enabled`	Explicit on/off state for the feature or response.	true

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SetPlayerInputEnabled({"enabled": true})
```

In use. Disable control during a carefully staged action and restore it on every exit path. Other locks, such as a running dialogue or cutscene, can still keep input unavailable after this action enables its own switch.

SetCharacterActionEnabled()

Enable / Disable Character Action

Enable or disable a named interaction response on a character.

Option key / field	Meaning and input	Default
Character `characterId`	Choose a Character Designer ID; player is a runtime alias where supported.	Blank
Mouse / Trigger Action `actionEvent`	Choose the response/event to enable or disable.	Blank
Enabled `enabled`	Explicit on/off state for the feature or response.	true

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SetCharacterActionEnabled({"characterId": "player", "actionEvent": "", "enabled": true})
```

In use. Disable Talk for a character who is busy, then restore it later. The event/verb selection matters: disabling one response is different from hiding the NPC or turning off all interaction.

ChangeCharacterProperty()

Change Character Property

Change a selected character property in runtime data.

Option key / field	Meaning and input	Default
Character `characterId`	Choose a Character Designer ID; player is a runtime alias where supported.	Blank
Property `property`	Choose a supported character property.	Blank
To `value`	A value of the type required by the selected variable, operation or property.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await ChangeCharacterProperty({"characterId": "player", "property": "visible", "value": true})
```

In use. Change a character's display or movement property supported by the picker. Use a value of the right type. A generic property assignment is not guaranteed to rebuild an already loaded model, material or animation setup.

AddCharacterToScene()

Add Character To Scene

Create a scene instance from an existing Character Designer definition.

Option key / field	Meaning and input	Default
Character `characterId`	Choose a Character Designer ID; player is a runtime alias where supported.	Blank
New ID `id`	Text.	Blank
Name `name`	Text.	Blank
XYZ `position`	Destination/placement coordinates as [X, Y, Z].	[0.0, 0.0, 0.0]
Enabled `enabled`	Explicit on/off state for the feature or response.	true
Visible `visible`	Explicit visible/hidden state.	true
Animation (optional) `animation`	Text.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await AddCharacterToScene({"characterId": "player", "id": "", "name": "powered"})
```

In use. Add Elias at [2,0,-1] with a unique scene entity ID. Choose the character definition as the source; the scene instance's ID identifies the copy you create. Repeating this action without an existence check can create unwanted duplicates.

Jump()

Jump

Ask the selected character to jump through the adventure extension.

Option key / field	Meaning and input	Default
Character `characterId`	Choose a Character Designer ID; player is a runtime alias where supported.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await Jump({"characterId": "player"})
```

In use. Use it for an appropriate controllable character during a test. A jump action does not automatically provide a jump animation, a platforming controller or a valid landing surface; those still depend on the character and room setup.

PlayAnimation()

Play Animation

Request an animation clip or state on the target entity. Most imported clips start and return immediately.

Option key / field	Meaning and input	Default
Room `sceneId`	Choose a scene ID.	Blank
Character / Object `entityId`	Choose an entity in the selected scene.	Blank
Animation `clip`	Choose a clip/state available to that entity.	Blank

Timing. Most imported animation requests return after starting playback. They do not wait for the full clip length; some procedural fallback motions do wait.

```
await PlayAnimation({"sceneId": "lab", "entityId": "archive_door", "clip": ""})
```

In use. Use a short, non-looping lever gesture or nod. Confirm the clip/state ID exists on the actual imported model. Looping animations do not have a natural one-shot finish, so do not build story timing on an endless loop.

StopAnimation()

Stop Animation

Stop the target entity's current animation through the runtime animation route.

Option key / field	Meaning and input	Default
Room `sceneId`	Choose a scene ID.	Blank
Character / Object `entityId`	Choose an entity in the selected scene.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await StopAnimation({"sceneId": "lab", "entityId": "archive_door"})
```

In use. Stop a looping inspection motion before changing the scene. For normal idle/walk behavior, allow the character's state system to resume afterward rather than assuming the stopped pose is a permanent character state.

PauseAnimation()

Animation: Pause

Pause the target entity's animation playback.

Option key / field	Meaning and input	Default
Room `sceneId`	Choose a scene ID.	Blank
Character / Object `entityId`	Choose an entity in the selected scene.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await PauseAnimation({"sceneId": "lab", "entityId": "archive_door"})
```

In use. Freeze an inspection model while the player reads a note. Pausing a clip does not pause the whole game, music or the character's navigation.

ResumeAnimation()

Animation: Resume

Resume the target entity's paused animation playback.

Option key / field	Meaning and input	Default
Room `sceneId`	Choose a scene ID.	Blank
Character / Object `entityId`	Choose an entity in the selected scene.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await ResumeAnimation({"sceneId": "lab", "entityId": "archive_door"})
```

In use. Pair it with Pause Animation on the same entity. If another state or clip replaced the paused animation, reselecting the correct state may be necessary.

Gameplay: Objects, descriptions and scene life

Visible, enabled, interactive and physically blocking are separate ideas. Use the smallest change that matches the story. For an item pickup, hiding its world representation and giving its inventory definition are two distinct actions.

SetObjectActionEnabled()

Enable / Disable Object Action

Enable or disable one object's interaction event or verb response.

Option key / field	Meaning and input	Default
Room `sceneId`	Choose a scene ID.	Blank
Object `entityId`	Choose an entity in the selected scene.	Blank
Mouse / Trigger Action `actionEvent`	Choose the response/event to enable or disable.	Blank
Enabled `enabled`	Explicit on/off state for the feature or response.	true

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SetObjectActionEnabled({
    "sceneId": "lab",
    "entityId": "archive_door",
    "actionEvent": ""
})
```

In use. Disable Use after a machine has been repaired while keeping Look available. Choose the intended event. This is more precise than disabling the entire object.

SetObjectEnabled()

Enable / Disable Object

Set an object's enabled state.

Option key / field	Meaning and input	Default
Room `sceneId`	Choose a scene ID.	Blank
Object `entityId`	Choose an entity in the selected scene.	Blank
Enabled `enabled`	Explicit on/off state for the feature or response.	true

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SetObjectEnabled({"sceneId": "lab", "entityId": "archive_door", "enabled": true})
```

In use. Disable a puzzle object while the scene is being prepared. Test its visual, input and collision behavior in the running game; use Set Object Visible when you only mean to change whether it is drawn.

SetObjectVisible()

Hide / Unhide Object

Show or hide an object.

Option key / field	Meaning and input	Default
Room `sceneId`	Choose a scene ID.	Blank
Object `entityId`	Choose an entity in the selected scene.	Blank
Visible `visible`	Explicit visible/hidden state.	true

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SetObjectVisible({"sceneId": "lab", "entityId": "archive_door", "visible": true})
```

In use. Hide the workbench key after giving its inventory item. Hiding is not a collision-size editor or a substitute for disabling a trigger. For reusable pickup logic, target this in a script with an object caller.

SetTriggerEnabled()

Enable / Disable Trigger

Enable or disable a trigger's runtime event behavior.

Option key / field	Meaning and input	Default
Room `sceneId`	Choose a scene ID.	Blank
Trigger `entityId`	Choose a trigger in the selected scene.	Blank
Enabled `enabled`	Explicit on/off state for the feature or response.	true

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SetTriggerEnabled({"sceneId": "lab", "entityId": "archive_door", "enabled": true})
```

In use. Turn off a doorway trigger during an opening cutscene, then turn it on when play begins. Existing overlaps and the player starting inside the volume need to be tested; do not treat the toggle as a guaranteed fresh Enter event.

SpawnObject()

Spawn Scene Object

Spawn or restore an already authored scene object definition.

Option key / field	Meaning and input	Default
Room `sceneId`	Choose a scene ID.	Blank
Scene Object `entityId`	Choose an entity in the selected scene.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SpawnObject({"sceneId": "lab", "entityId": "archive_door"})
```

In use. Restore a temporary prop that was despawned. The chosen object must have an authored definition the runtime can find. Use Create Primitive or Add Asset To Scene to create a new record instead.

DespawnObject()

Despawn Scene Object

Remove a live object instance through the runtime scene-life route.

Option key / field	Meaning and input	Default
Room `sceneId`	Choose a scene ID.	Blank
Scene Object `entityId`	Choose an entity in the selected scene.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await DespawnObject({"sceneId": "lab", "entityId": "archive_door"})
```

In use. Remove a temporary projection when its sequence ends. Keep an ID or authored definition if it will be restored. Despawning a visible item does not automatically remove a matching inventory item.

ChangeObjectProperty()

Change Object Property

Assign a property on an object's runtime record.

Option key / field	Meaning and input	Default
Scene `sceneId`	Choose a scene ID.	Blank
Object `entityId`	Choose an entity in the selected scene.	Blank
Property `property`	Choose a supported object property.	Blank
To `value`	A value of the type required by the selected variable, operation or property.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await ChangeObjectProperty({
    "sceneId": "lab",
    "entityId": "archive_door",
    "property": "visible"
})
```

In use. Change a supported property such as a display field or simple state value. Dedicated visibility, animation and effect actions are preferable when they exist: a generic record edit may not rebuild the corresponding live visual resource.

ChangeSceneDescription()

Change Scene Description

Change the description stored for a scene.

Option key / field	Meaning and input	Default
Scene `sceneId`	Choose a scene ID.	Blank
Description `description`	Text.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await ChangeSceneDescription({"sceneId": "lab", "description": ""})
```

In use. Replace the room description after the machine starts. This edits descriptive text; it is not an automatic speech line. Add Show Message or Player Say if the player should hear or read it immediately.

ChangeObjectDescription()

Change Object Description

Change an object's descriptive text.

Option key / field	Meaning and input	Default
Scene `sceneId`	Choose a scene ID.	Blank
Object `entityId`	Choose an entity in the selected scene.	Blank
Description `description`	Text.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await ChangeObjectDescription({
    "sceneId": "lab",
    "entityId": "archive_door",
    "description": ""
})
```

In use. After powering the lamp, change its description to "The signal is powered." The next Look response must use that description or the built-in fallback; a hard-coded Look message continues to say whatever you wrote in it.

ChangeInventoryDescription()

Change Inventory Description

Change an inventory item's description.

Option key / field	Meaning and input	Default
Inventory Item `itemId`	Choose an inventory definition ID.	Blank
Description `description`	Text.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await ChangeInventoryDescription({"itemId": "key", "description": ""})
```

In use. Update the key's description after discovering which door it opens. Changing description text is separate from changing its icon, name or quantity.

ChangeCharacterDescription()

Change Character Description

Change a character's description.

Option key / field	Meaning and input	Default
Character / Player `characterId`	Choose a Character Designer ID; player is a runtime alias where supported.	Blank
Description `description`	Text.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await ChangeCharacterDescription({"characterId": "player", "description": ""})
```

In use. Describe Elias differently after he becomes a companion. A custom Look script with its own fixed text will not automatically use the changed description.

ChangeRegionDescription()

Change Region Description

Change the description of a named region.

Option key / field	Meaning and input	Default
Scene `sceneId`	Choose a scene ID.	Blank
Region `regionId`	Choose a region in the selected scene.	Blank
Description `description`	Text.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await ChangeRegionDescription({"sceneId": "lab", "regionId": "", "description": ""})
```

In use. Change a frozen pool's description after it melts. Choose the region in the correct scene. This does not turn a non-interactive region into a clickable hotspot.

AddAssetToScene()

Add Asset to Scene

Create a runtime scene entity using an imported asset.

Option key / field	Meaning and input	Default
Asset `assetId`	Choose an imported asset.	Blank
New ID `id`	Text.	Blank
Name `name`	Text.	Blank
XYZ `position`	Destination/placement coordinates as [X, Y, Z].	[0.0, 0.0, 0.0]
Enabled `enabled`	Explicit on/off state for the feature or response.	true
Visible `visible`	Explicit visible/hidden state.	true

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await AddAssetToScene({"assetId": "your_asset", "id": "", "name": "powered"})
```

In use. Spawn a lantern model at [1,0,-2] with a unique ID. Import and verify the asset first. Repeating the spawn is not the same as showing one existing lantern.

CreatePrimitive()

Create Primitive

Create a runtime primitive such as a box, sphere, cylinder, capsule or plane.

Option key / field	Meaning and input	Default
Primitive `primitive`	Choose: box, sphere, cylinder, capsule, plane.	"box"
ID `id`	Text.	Blank
XYZ `position`	Destination/placement coordinates as [X, Y, Z].	[0.0, 0.0, 0.0]

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await CreatePrimitive({"primitive": "box", "id": "", "position": [0.0, 0.0, 0.0]})
```

In use. Spawn a temporary marker for a puzzle demonstration. The normal action form has a small set of fields; detailed dimensions/material settings may need structured data or later property changes supported by the runtime.

CreateProcedural()

Create Procedural

Create a runtime procedural entity from the requested generator type.

Option key / field	Meaning and input	Default
Procedural Type `procedural`	Text.	Blank
ID `id`	Text.	Blank
XYZ `position`	Destination/placement coordinates as [X, Y, Z].	[0.0, 0.0, 0.0]

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await CreateProcedural({"procedural": "", "id": "", "position": [0.0, 0.0, 0.0]})
```

In use. Use a generator name supported by the engine and check the result in Test Scene. This is not a text-to-model system: an arbitrary description is not a procedural generator ID.

Gameplay: Cameras, transitions and screen effects

An authored camera is a reusable room view. A blend moves the view between cameras. An orbit or shake adds continuing or timed motion. Screen fades and transitions work on the presented image and need not change rooms.

ActivateCamera()

Activate / Blend to Camera

Activate an authored camera and optionally blend to it, waiting when Wait is enabled.

Option key / field	Meaning and input	Default
Camera `cameraId`	Choose an authored camera ID.	Blank
Blend Seconds (-1 = camera default) `duration`	Time in seconds for this action. Read its timing note before assuming it waits. Range: -1.0 to 60.0.	-1.0
Wait for Blend `wait`	Whether this action should wait for its own effect to finish.	true

Timing. Waits when Wait is on.

```
await ActivateCamera({"cameraId": "wide", "duration": 1, "wait": true})
```

In use. Blend to the lamp close-up over 1 second. A negative duration uses the camera's configured blend timing; 0 makes a cut. If a following action must begin after arrival, keep Wait enabled.

SetCameraBehavior()

Set Advanced Camera Behavior

Change the selected camera's advanced behavior, targets, smoothing, framing, path and bounds settings.

Option key / field	Meaning and input	Default
Camera (empty = active) `cameraId`	Choose an authored camera ID. May be blank.	Blank
Enable Advanced Camera `enabled`	Explicit on/off state for the feature or response.	false
Camera Behavior `mode`	Choose: fixed, follow, group, framed, path.	"fixed"

Option key / field	Meaning and input	Default
Follow Target `targetId`	Choose a scene entity ID. May be blank.	"player"
Additional Target IDs (comma separated) `targetIds`	Text.	Blank
Look At Target (optional) `lookAtId`	Choose a scene entity ID. May be blank.	Blank
Target Offset XYZ `targetOffset`	Offset from the camera target as [X, Y, Z].	[0.0, 1.3, 0.0]
Follow Damping (seconds) `damping`	Camera follow smoothing value; tune while watching the target move. Range: 0.0 to 5.0.	0.2
Default Camera Blend (seconds) `blendDuration`	Default camera blend time in seconds. Range: 0.0 to 60.0.	1.0
Blend Curve `ease`	Choose: linear, sine, cubic, quint.	"sine"
Camera Priority `priority`	Camera selection priority number. Range: -100000 to 100000.	0
Automatically Frame Group `autoFrame`	Allow the camera to calculate framing for its targets.	true
Minimum Group Distance / View Height `minDistance`	Nearest allowed camera framing distance. Range: 0.1 to 10000.0.	3.0
Maximum Group Distance / View Height `maxDistance`	Farthest allowed camera framing distance. Range: 0.1 to 10000.0.	18.0
Group Framing Padding `framingPadding`	Extra space around the framed target group. Range: 1.0 to 4.0.	1.3
Framing Zone Width (0-1) `deadZoneWidth`	Width of the camera framing/dead zone. Range: 0.0 to 1.0.	0.4
Framing Zone Height (0-1) `deadZoneHeight`	Height of the camera framing/dead zone. Range: 0.0 to 1.0.	0.3
Camera Path Points (XYZ array) `pathPoints`	Camera path point data as a JSON array; use the path editor when offered.	[]

Option key / field	Meaning and input	Default
Respect Room Camera Bounds `respectBounds`	Apply the room camera bounds to this camera behavior.	true

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SetCameraBehavior({"cameraId": "wide", "enabled": false, "mode": "fixed"})
```

In use. Enable advanced behavior and choose Follow for a tracking shot. Changing the settings is different from activating that camera. Path mode needs usable path points; a group view needs valid targets. Keep Minimum Distance no greater than Maximum Distance.

SetCameraTargets()

Set Camera Targets

Replace the main, group and look-at targets used by an advanced camera.

Option key / field	Meaning and input	Default
Camera (empty = active) `cameraId`	Choose an authored camera ID. May be blank.	Blank
Follow Target `targetId`	Choose a scene entity ID. May be blank.	"player"
Additional Target IDs (comma separated) `targetIds`	Text.	Blank
Look At Target (optional) `lookAtId`	Choose a scene entity ID. May be blank.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SetCameraTargets({"cameraId": "wide", "targetId": "archive_door", "targetIds": ""})
```

In use. Use the player as the follow target and Elias as the look-at target for a conversation. Supply IDs in the format the selected field expects. This does not create the targets or move either actor.

SetCameraPriority()

Set Camera Priority

Change the priority used by the advanced-camera system.

Option key / field	Meaning and input	Default
Camera `cameraId`	Choose an authored camera ID.	Blank
Camera Priority `priority`	Camera selection priority number.	10

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SetCameraPriority({"cameraId": "wide", "priority": 10})
```

In use. Raise a conversation camera's priority while a scene is staged, then restore the previous priority. Priority is camera-selection policy; use Activate Camera when the sequence must explicitly take a particular view.

ChangeCamera()

Change Camera

Switch to a selected authored camera through the direct camera-change route.

Option key / field	Meaning and input	Default
Room `sceneId`	Choose a scene ID.	Blank
Camera View `cameraId`	Choose a camera in the selected scene.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await ChangeCamera({"sceneId": "lab", "cameraId": "wide"})
```

In use. Choose Room View after an inspection ends. The scene field supplies picker context; it does not transport the player to another room. Use Activate Camera when you want explicit blend duration and wait control.

CameraOrbit()

Camera Orbit

Start an orbit around an entity with distance, height, direction, speed and repetitions.

Option key / field	Meaning and input	Default
Target Object `entityId`	Choose a scene entity ID.	Blank
Degrees Per Second `speed`	Orbit angular speed.	45
Distance `distance`	Number; decimals allowed.	5
Orbit Mode `mode`	Choose: 360, 180, 180_back.	"360"
Repeat Count `repetitions`	Whole number.	1
Clockwise `clockwise`	On/off.	false
Height `height`	Number; decimals allowed.	2

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await CameraOrbit({"entityId": "archive_door", "speed": 45, "distance": 5})
```

In use. Orbit the clock for a reveal, then stop it before returning to normal play. The orbit continues as a behavior after it starts; do not assume that placing it before another action makes the list wait for every revolution.

StopCameraOrbit()

Stop Camera Orbit

Stop the current extension-driven orbit.

No options in the standard catalog schema.

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await StopCameraOrbit({})
```

In use. Use it when an inspection closes or a conversation ends. Decide which normal camera should take over and activate that camera explicitly when needed.

CameraShake()

Camera Shake

Shake the 3D camera for the chosen duration and direction, waiting for the shake routine.

Option key / field	Meaning and input	Default
Direction `direction`	Choose: both, horizontal, vertical.	"both"
Duration (seconds) `duration`	Time in seconds for this action. Read its timing note before assuming it waits. Range: 0.01 to 60.0.	0.5
Power `power`	Number; decimals allowed. Range: 0.0 to 10.0.	0.25

Timing. Waits for the shake routine.

```
await CameraShake({"direction": "both", "duration": 1, "power": 0.25})
```

In use. Use a small power, such as 0.1 to 0.25, for a machinery rumble. Its scale differs from the pixel-oriented Shake Screen action. Keep shakes short and provide a reduced-motion option for players who need it.

ShakeScreen()

Shake Screen

Start the extension's screen-shake effect with horizontal, vertical or combined movement.

Option key / field	Meaning and input	Default
Direction `direction`	Choose: both, horizontal, vertical.	"both"
Duration (seconds) `duration`	Time in seconds for this action. Read its timing note before assuming it waits. Range: 0 to 60.	0.5
Power (pixels) `power`	Number; decimals allowed. Range: 0 to 1000.	8

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await ShakeScreen({"direction": "both", "duration": 1, "power": 8})
```

In use. Use a modest screen shake when a heavy mechanism locks into place. This is a different effect route and strength scale from Camera Shake. Use Stop Screen Shake to cancel an ongoing screen effect.

StopScreenShake()

Stop Screen Shake

Stop the current extension-driven screen shake.

No options in the standard catalog schema.

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await StopScreenShake({})
```

In use. Call it if a shake is interrupted by a menu or a scene change. It does not select a camera or undo a separately running camera orbit.

FadeScreen()

Screen Fade

Fade a color over the screen, or fade that cover away, over a duration.

Option key / field	Meaning and input	Default
Direction `direction`	Choose: out, in.	"out"
Color `color`	Color picker or color string.	"#000000"
Duration (seconds) `duration`	Time in seconds for this action. Read its timing note before assuming it waits. Range: 0.0 to 60.0.	0.5

Timing. Waits for the fade routine.

```
await FadeScreen({"direction": "out", "color": "#000000", "duration": 1})
```

In use. Fade Out to black, make the intended change, then Fade In. An Out fade leaves the cover in place; remember the In step. Use Change Scene's transition controls for a normal room change when they fit the job.

ScreenTransition()

Screen Transition

Play a screen transition effect such as dissolve, wipe, iris, pixelation or glitch.

Option key / field	Meaning and input	Default
Transition `transitionType`	Choose: dissolve, fade, wipe, match-cut, whip-pan, iris-wipe, pixelation, glitch, comic-panel.	"dissolve"
Duration (seconds) `duration`	Time in seconds for this action. Read its timing note before assuming it waits. Range: 0.05 to 8.0.	0.8
Color `color`	Color picker or color string.	"#000000"
Direction `direction`	Choose: left, right, up, down.	"left"
Intensity `intensity`	Number; decimals allowed. Range: 0.0 to 10.0.	1.0

Timing. Waits for the transition routine.

```
await ScreenTransition({"transitionType": "dissolve", "duration": 1, "color": "#000000"})
```

In use. Use a short iris wipe for a stylized reveal. This action plays the effect; Change Scene is still the command that changes rooms. Direction and intensity have different visual importance for different transition types.

SetRetroStyle()

Set Retro Rendering Style

Choose the retro screen treatment: none, PSX, N64 or horror fog, with strength and pixel size.

Option key / field	Meaning and input	Default
Retro Rendering Style `style`	Choose: none, psx, n64, horror-fog.	"psx"
Strength `strength`	Number; decimals allowed. Range: 0 to 1.	1.0
Retro Pixel Size `pixelSize`	Pixel-block size for the retro screen treatment. Range: 1 to 16.	3.0

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SetRetroStyle({"style": "psx", "strength": 1.0, "pixelSize": 3.0})
```

In use. Apply a restrained PSX look to a dream, then restore none afterward. These are screen treatments; they are not a replacement for modeling geometry in a particular console style or for physical scene fog.

ZoomBlur()

Zoom Blur

Play a radial zoom blur around a normalized screen center, optionally including GUI and waiting for completion.

Option key / field	Meaning and input	Default
Strength `strength`	Number; decimals allowed. Range: 0 to 1.	0.7
Duration (seconds) `duration`	Time in seconds for this action. Read its timing note before assuming it waits. Range: 0 to 60.	0.6
Center X (0-1) `centerX`	Normalized horizontal effect center: 0 left, 0.5 middle, 1 right. Range: 0 to 1.	0.5
Center Y (0-1) `centerY`	Normalized vertical effect center: 0 top, 0.5 middle, 1 bottom. Range: 0 to 1.	0.5
Clear Center Radius `clearRadius`	Radius around the zoom-blur center kept comparatively clear. Range: 0 to 0.8.	0.08
Direction `direction`	Choose: inward, outward.	"inward"
Include GUI in Effect `includeGUI`	Apply the screen effect to interface presentation as well.	false
Wait Until Finished `wait`	Whether this action should wait for its own effect to finish.	false

Timing. Waits when Wait is on.

```
await ZoomBlur({"strength": 0.7, "duration": 1, "centerX": 0.5})
```

In use. Use a brief outward pulse when the signal comes online. Center X/Y run from 0 to 1, so [0.5,0.5] is the middle. Keep Include GUI off when text must remain crisp.

StopZoomBlur()

Stop Zoom Blur

Cancel the active zoom-blur treatment.

No options in the standard catalog schema.

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await StopZoomBlur({})
```

In use. Use it when a timed effect is interrupted. If a persistent shader stack also contains a blur, stopping this one-shot effect does not necessarily remove that separately authored shader.

Gameplay: Room changes, 2D controls and environmental state

Keep room transitions at the end of a local sequence where possible. After changing rooms, an old-room object ID may no longer resolve. Layer and depth commands apply to the 2D/hybrid scene systems, not to arbitrary 3D model transforms.

ChangeScene()

Change Scene

Move the active player into another scene using a spawn and optional destination position/camera/transition settings.

Option key / field	Meaning and input	Default
Destination Scene `sceneId`	Choose a scene ID.	-
Spawn (optional) `spawnId`	Choose an arrival spawn in the destination scene. May be blank.	Blank
XYZ when no spawn `position`	Destination/placement coordinates as [X, Y, Z].	[0.0, 0.0, 0.0]

Timing. Awaits the room-change routine.

```
await ChangeScene({"sceneId": "lab", "spawnId": "", "position": [0.0, 0.0, 0.0]})
```

In use. Choose The Archive and the from_signal spawn. Native room/object event inspectors expose Destination Scene and transition controls. Some shared action forms omit the scene selector; call a visual or text script from those surfaces instead. A spawn is usually clearer than loose coordinates.

sceneId is accepted by the runtime even though some shared forms omit that field. Native GDScript can supply it directly as shown. Use this room-change route instead of replacing the top-level Godot SceneTree scene.

SwitchCharacter()

Switch Character

Change the currently controlled playable character.

Option key / field	Meaning and input	Default
Active Player `characterId`	Choose a Character Designer ID; player is a runtime alias where supported.	Blank

Timing. Awaits the character-switch routine.

```
await SwitchCharacter({"characterId": "player"})
```

In use. Switch from Ivo to another character after confirming that character is configured as playable. The new character can have its own room, inventory, HUD and journal settings. It is not a command to swap the visible model on the same actor.

IsCurrentRoom()

Is Current Room

Test the current room through the adventure extension and return the result.

Option key / field	Meaning and input	Default
Scene `sceneId`	Choose a scene ID.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await IsCurrentRoom({"sceneId": "lab"})
```

In use. Use it from a script that reads the returned Boolean. An ordinary action list does not branch on the result. In visual scripting, choose If Current Room when you want visible True and False paths.

SetRoomDimension()

Set Room Dimension

Change a scene's runtime dimension setting between 3D, 2D and 2.5D, with an option to retain the 3D layer.

Option key / field	Meaning and input	Default
Room (empty = current) `sceneId`	Choose a scene ID. May be blank.	Blank
Room Dimension `dimension`	Choose: 3d, 2d, 2.5d.	"3d"
Keep 3D Layer `keep3DLayer`	Retain the 3D layer when using the hybrid room setup.	true

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SetRoomDimension({"sceneId": "lab", "dimension": "3d", "keep3DLayer": true})
```

In use. Use it only after the target scene has the assets and navigation appropriate to that mode. Switching a setting does not automatically build a painted backdrop, reposition every entity or convert navigation geometry.

Set2DCamera()

Set 2D Camera

Set the 2D camera's position, zoom and rotation for a scene.

Option key / field	Meaning and input	Default
Room (empty = current) `sceneId`	Choose a scene ID. May be blank.	Blank
Camera Position XY `position`	2D position as [X, Y].	[0.0, 0.0]
Camera Zoom XY `zoom`	[X, Y] numeric array.	[1.0, 1.0]
Camera Rotation (degrees) `rotation`	2D camera rotation angle; follow the field label for its units.	0.0

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await Set2DCamera({"sceneId": "lab", "position": [0.0, 0.0], "zoom": [1.0, 1.0]})
```

In use. Zoom a painted-room view to [1.25,1.25]. This acts on the 2D camera system; it does not change a 3D perspective camera's field of view. Keep both zoom values above zero.

Set2DLayerVisible()

Show / Hide 2D Layer

Show or hide a named 2D layer.

Option key / field	Meaning and input	Default
Room (empty = current) `sceneId`	Choose a scene ID. May be blank.	Blank
2D Layer ID `layerId`	Text.	Blank
Visible `visible`	Explicit visible/hidden state.	true

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await Set2DLayerVisible({"sceneId": "lab", "layerId": "", "visible": true})
```

In use. Hide a foreground curtain layer after it opens. Select the layer ID used by the room, not the ID of an image asset imported into the project.

Set2DEntityDepth()

Set 2D Entity Depth

Change an entity's sorting depth in the 2D/hybrid scene.

Option key / field	Meaning and input	Default
Room (empty = current) `sceneId`	Choose a scene ID. May be blank.	Blank
2D Character / Object `entityId`	Choose an entity in the selected scene.	Blank
Drawing Depth `depth`	Whole number.	100

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await Set2DEntityDepth({"sceneId": "lab", "entityId": "archive_door", "depth": 100})
```

In use. Adjust a foreground prop's depth so it appears in front of the intended sprites. Depth is drawing order; it is not a height coordinate or a collision box dimension.

Set2DParallax()

Set 2D Layer Parallax

Change a named 2D layer's parallax ratio.

Option key / field	Meaning and input	Default
Room (empty = current) `sceneId`	Choose a scene ID. May be blank.	Blank
2D Layer ID `layerId`	Text.	Blank
Parallax Ratio XY `ratio`	[X, Y] numeric array.	[1.0, 1.0]

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await Set2DParallax({"sceneId": "lab", "layerId": "", "ratio": [1.0, 1.0]})
```

In use. Use a smaller camera-motion ratio for a distant sky and a larger one for nearby scenery. The layer must already be set up with suitable artwork and coverage; a ratio does not extend the bitmap.

SetRoomCameraBounds()

Change Room Camera Bounds

Set whether camera bounds are enabled and edit the room's X/Z limits.

Option key / field	Meaning and input	Default
Room `sceneId`	Choose a scene ID.	Blank
Bounds Enabled `enabled`	Explicit on/off state for the feature or response.	true
Left / Minimum X `minX`	Number; decimals allowed.	0
Right / Maximum X `maxX`	Number; decimals allowed.	0
Top / Minimum Z `minZ`	Number; decimals allowed.	0
Bottom / Maximum Z `maxZ`	Number; decimals allowed.	0

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SetRoomCameraBounds({"sceneId": "lab", "enabled": true, "minX": 0})
```

In use. Constrain a 3D tracking camera to the playable room. These are camera limits, not the player's navigation bounds. Expanding them alone does not make an oversized backdrop walkable.

SetInteractionApproachEnabled()

Enable / Disable Automatic Interaction Approach

Set the global runtime approach-to-target behavior, including distance and whether to face the target.

Option key / field	Meaning and input	Default
Automatic Approach Enabled `enabled`	Explicit on/off state for the feature or response.	true
Default Stop Distance `distance`	Number; decimals allowed. Range: 0.0 to 20.0.	0.35
Face Target After Approaching `faceTarget`	On/off.	true

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SetInteractionApproachEnabled({"enabled": true, "distance": 0.35, "faceTarget": true})
```

In use. Enable walking up to interactive props, then choose a modest stop distance. If the player stops too far away, inspect both approach settings and the physical collider. The event volume is a separate object.

SetObjectInteractionApproach()

Set Object Interaction Approach

Override approach behavior for one object, or inherit the global behavior.

Option key / field	Meaning and input	Default
Object Room `sceneId`	Choose a scene ID. May be blank.	Blank
Object / Character `entityId`	Choose an entity in the selected scene.	Blank
Approach Override `mode`	Choose: inherit, enabled, disabled.	"inherit"
Stop Distance (0 = global) `distance`	Number; decimals allowed. Range: 0.0 to 20.0.	0.0
Face Target When Override Is Enabled `faceTarget`	On/off.	true

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SetObjectInteractionApproach({
    "sceneId": "lab",
    "entityId": "archive_door",
    "mode": "inherit"
})
```

In use. Give the door its own enabled approach mode and a close stop distance. A small distance cannot move the character through an oversized blocking collider. Set collider dimensions independently in the object's collision settings.

SetSceneBrightness()

Set Room Brightness

Change a scene's brightness multiplier.

Option key / field	Meaning and input	Default
Scene `sceneId`	Choose a scene ID.	Blank
Brightness (0 = Dark, 1 = Normal) `brightness`	Number; decimals allowed.	1.0

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SetSceneBrightness({"sceneId": "lab", "brightness": 1.0})
```

In use. Darken the room slightly when the power fails. This is a broad rendering adjustment, not a command to move a light or create a reflective surface. Compare the result with exposure and renderer settings.

SetSceneTint()

Set Room Tint

Apply a color tint to a scene with a chosen strength.

Option key / field	Meaning and input	Default
Scene `sceneId`	Choose a scene ID.	Blank
Tint Color `tintColor`	Color picker or color string.	"#ffffff"
Tint Strength (0 = Off, 1 = Full) `strength`	Number; decimals allowed. Range: 0.0 to 1.0.	1.0

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SetSceneTint({"sceneId": "lab", "tintColor": "#ffffff", "strength": 1.0})
```

In use. Use a pale blue tint for a cold room, then restore white or zero strength as appropriate. The tint affects presentation; it does not recolor imported source textures on disk.

SetFlashlightMode()

Enable / Disable Flashlight Mode

Enable or disable the flashlight-style visibility mask and adjust its strength, radius and shape.

Option key / field	Meaning and input	Default
Flashlight Enabled `enabled`	Explicit on/off state for the feature or response.	true
Light Strength `strength`	Number; decimals allowed. Range: 0.0 to 8.0.	1.0
Light Radius `radius`	Number; decimals allowed. Range: 8.0 to 4096.0.	180.0
Light Shape `shape`	Choose: circle, spherical, square, rectangle, triangle.	"circle"

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SetFlashlightMode({"enabled": true, "strength": 1.0, "radius": 180.0})
```

In use. Use a circular mask for a dark inspection scene. Radius is a screen-oriented value in this feature; do not confuse it with an OmniLight3D range. This is not a replacement for a physical light attached to a character's hand.

SetSnowAccumulation()

Set Snow Accumulation

Set or animate accumulated snow on the scene, with duration/pace and optional waiting.

Option key / field	Meaning and input	Default
Scene (empty = current) `sceneId`	Choose a scene ID. May be blank.	Blank
Accumulation Amount (0 to 1) `amount`	Target accumulated snow amount from 0 to 1. Range: 0.0 to 1.0.	1.0
Transition Duration (seconds, 0 = authored pace) `duration`	Time in seconds for this action. Read its timing note before assuming it waits. Range: 0.0 to 86400.0.	0.0
Amount Per Second (used only when duration = 0) `pace`	Snow accumulation/melt pace; use the duration route for a simple timed test. Range: 0.0 to 100.0.	0.0
Wait Until Accumulation Finishes `waitForCompletion`	Wait for the snow animation to complete before continuing.	true

Timing. Waits when Wait for Completion is on.

```
await SetSnowAccumulation({"sceneId": "lab", "amount": 1.0, "duration": 1})
```

In use. Build snow over a short time while a reveal runs. Use Wait for Completion only when subsequent instructions should be delayed. The target room must have snow-capable surfaces and appropriate visual settings.

MeltSnow()

Melt Accumulated Snow

Reduce the scene's accumulated snow over time.

Option key / field	Meaning and input	Default
Scene (empty = current) `sceneId`	Choose a scene ID. May be blank.	Blank
Melt Duration (seconds) `duration`	Time in seconds for this action. Read its timing note before assuming it waits. Range: 0.0 to 86400.0.	2.0
Amount Per Second (used only when duration = 0) `pace`	Snow accumulation/melt pace; use the duration route for a simple timed test. Range: 0.0 to 100.0.	0.0
Wait Until Melting Finishes `waitForCompletion`	Wait for the snow animation to complete before continuing.	true

Timing. Waits when Wait for Completion is on.

```
await MeltSnow({"sceneId": "lab", "duration": 1, "pace": 0.0})
```

In use. Melt the snow after activating a heater. Match duration/pace to the intended effect and test which surfaces receive snow. It changes accumulated snow, not the count of particles emitted by a separate falling-snow effect.

Gameplay: Continuous object effects and attachments

Configure an effect or attach point first, then use these actions to control it. An attach point is a named location on an object, model bone or sprite. Contact watches report proximity between points; they are not a full physics collision system.

ToggleMotion()

Toggle Motion

Invert an object's authored motion-enabled state, optionally mirroring that state into a variable.

Option key / field	Meaning and input	Default
Room `sceneId`	Choose a scene ID.	Blank
Object `entityId`	Choose an entity in the selected scene.	Blank
State Variable (optional) `variable`	Choose a game variable name. May be blank.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await ToggleMotion({"sceneId": "lab", "entityId": "archive_door", "variable": ""})
```

In use. Toggle a rotating sign when a switch is used. Configure the object's motion in its inspector first. Repeated events invert the state again; use Set Motion when a specific final state matters.

SetMotion()

Set Motion

Explicitly enable or disable authored object motion.

Option key / field	Meaning and input	Default
Room `sceneId`	Choose a scene ID.	Blank
Object `entityId`	Choose an entity in the selected scene.	Blank
Enabled `enabled`	Explicit on/off state for the feature or response.	true
State Variable (optional) `variable`	Choose a game variable name. May be blank.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SetMotion({"sceneId": "lab", "entityId": "archive_door", "enabled": true})
```

In use. Start the clock's rotating mechanism after power is restored. Choose Enabled on for a repeat-safe command. This does not create the rotation path, speed or scaling motion; it controls the setup already present on the object.

ToggleGlow()

Toggle Glow

Invert an object's authored glow state and optionally record it in a variable.

Option key / field	Meaning and input	Default
Entity `entityId`	Choose a scene entity ID.	Blank
Variable `variable`	Text.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await ToggleGlow({"entityId": "archive_door", "variable": ""})
```

In use. Toggle a test lamp during inspection. Visible bloom also depends on renderer/environment support. A bright material without bloom is still emissive; glow is not a mirror reflection of the player.

SetGlow()

Set Glow

Set an object's authored glow state explicitly.

Option key / field	Meaning and input	Default
Entity `entityId`	Choose a scene entity ID.	Blank
Enabled `enabled`	Explicit on/off state for the feature or response.	true
Variable `variable`	Text.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SetGlow({"entityId": "archive_door", "enabled": true, "variable": ""})
```

In use. Enable the signal lamp once the circuit is powered. Use the object's glow setup and the room's bloom/glow settings together. Repeating Set Glow Enabled is safer than toggling when a trigger can fire repeatedly.

ToggleRGBString()

Toggle Rgb String

Toggle an authored RGB light string, optionally writing its state to a variable.

Option key / field	Meaning and input	Default
Entity `entityId`	Choose a scene entity ID.	Blank
Variable `variable`	Text.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await ToggleRGBString({"entityId": "archive_door", "variable": ""})
```

In use. Turn decorative bulbs on after a celebration begins. The bulbs, spacing and light settings belong to the RGB string setup; this action does not create a new string from an ordinary mesh.

SetAttachPoint()

Set Attach Point

Create or update a named attach point with position, rotation, coordinate space, bone and contact radius.

Option key / field	Meaning and input	Default
Object / Character `entityId`	Choose a scene entity ID.	Blank
Attach Point `point`	Name of the source attach point.	Blank
Position `position`	Destination/placement coordinates as [X, Y, Z].	[0, 0, 0]
Rotation (degrees) `rotation`	Rotation values as a numeric array; follow the field's stated units.	[0, 0, 0]
Bone Name (optional) `bone`	Exact imported skeleton bone name, if this point follows a bone.	Blank
Coordinate Space `space`	Choose: object, model, sprite_uv.	"object"
Contact Radius `contactRadius`	Radius used to test contact around an attach point.	0.1
Enabled `enabled`	Explicit on/off state for the feature or response.	true

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SetAttachPoint({"entityId": "archive_door", "point": "", "position": [0, 0, 0]})
```

In use. Define right_hand_grip on a character's hand bone. Object/model coordinates and sprite UV coordinates mean different things; choose the space deliberately. The bone name must match the imported skeleton exactly.

RemoveAttachPoint()

Remove Attach Point

Remove a named attach point from an object or character.

Option key / field	Meaning and input	Default
Object / Character `entityId`	Choose a scene entity ID.	Blank
Attach Point `point`	Name of the source attach point.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await RemoveAttachPoint({"entityId": "archive_door", "point": ""})
```

In use. Remove a temporary alignment point after staging a sequence. Any attachment or watch referring to that name must be cleaned up as well; it cannot follow a point that no longer exists.

GetAttachPoint()

Get Attach Point

Read an attach point's information and optionally store the result in a game variable.

Option key / field	Meaning and input	Default
Object / Character `entityId`	Choose a scene entity ID.	Blank
Attach Point `point`	Name of the source attach point.	Blank
Store Result in Variable `resultVariable`	Name of the game variable that receives the returned value.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await GetAttachPoint({"entityId": "archive_door", "point": "", "resultVariable": ""})
```

In use. Store a hand point in hand_info for a script to inspect. This returns structured information rather than a ready-made visual branch. Inspect the result before assuming a missing point has a usable world position.

AttachPoints()

Attach Points

Attach one entity's point to another entity's point, optionally aligning rotation and suspending movement.

Option key / field	Meaning and input	Default
Object / Character `entityId`	Choose a scene entity ID.	Blank
Attach Point `point`	Name of the source attach point.	Blank
Target Object `targetId`	Choose a scene entity ID.	Blank
Target Attach Point `targetPoint`	Name of the attach point on the target entity.	Blank
Align Rotation `alignRotation`	Align point orientation as well as position.	true
Suspend Movement `suspendMovement`	Suspend the attached character/entity movement where supported.	true

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await AttachPoints({"entityId": "archive_door", "point": "", "targetId": "archive_door"})
```

In use. Attach a prop grip point to the player's hand grip point. Choose which entity is the child and which is the target. Suspending a character's movement is useful for staging but must fit the rest of the sequence.

AlignAttachPoints()

Align Attach Points

Align two attach points once without establishing the same continuing attachment relationship.

Option key / field	Meaning and input	Default
Object / Character `entityId`	Choose a scene entity ID.	Blank
Attach Point `point`	Name of the source attach point.	Blank
Target Object `targetId`	Choose a scene entity ID.	Blank
Target Attach Point `targetPoint`	Name of the attach point on the target entity.	Blank
Align Rotation `alignRotation`	Align point orientation as well as position.	true

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await AlignAttachPoints({"entityId": "archive_door", "point": "", "targetId": "archive_door"})
```

In use. Align two hands before a handshake pose. If either character then moves or the animation moves the hands apart, a one-time alignment is not guaranteed to keep contact throughout the animation.

AttachPointsTouching()

Attach Points Touching

Check whether two points are touching within their radii and tolerance, and optionally store the result.

Option key / field	Meaning and input	Default
Object / Character `entityId`	Choose a scene entity ID.	Blank
Attach Point `point`	Name of the source attach point.	Blank
Target Object `targetId`	Choose a scene entity ID.	Blank
Target Attach Point `targetPoint`	Name of the attach point on the target entity.	Blank
Contact Tolerance `tolerance`	Extra allowed contact distance; use a nonnegative number.	0.0
Store Result in Variable `resultVariable`	Name of the game variable that receives the returned value.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await AttachPointsTouching({
    "entityId": "archive_door",
    "point": "",
    "targetId": "archive_door"
})
```

In use. Store hand_contact, then branch on it in a graph or script. Putting this test in a plain list does not make later instructions conditional. Use If Attach Points Touching for visible graph branches.

GetAttachPointContacts()

Get Attach Point Contacts

Get contact information for the requested point/target filters.

Option key / field	Meaning and input	Default
Object / Character `entityId`	Choose a scene entity ID.	Blank
Attach Point `point`	Name of the source attach point.	Blank
Target Object `targetId`	Choose a scene entity ID.	Blank
Target Attach Point `targetPoint`	Name of the attach point on the target entity.	Blank
Contact Tolerance `tolerance`	Extra allowed contact distance; use a nonnegative number.	0.0
Store Result in Variable `resultVariable`	Name of the game variable that receives the returned value.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await GetAttachPointContacts({
    "entityId": "archive_door",
    "point": "",
    "targetId": "archive_door"
})
```

In use. Read contact details for a carried tool and let a script decide which target was touched. A returned collection or record is not automatically converted into a puzzle success flag.

WatchAttachPointContacts()

Watch Attach Point Contacts

Start a named contact watch, optionally using swept detection between frames.

Option key / field	Meaning and input	Default
Watch ID `watchId`	Stable name for this continuing contact watch.	Blank
Object / Character `entityId`	Choose a scene entity ID.	Blank
Attach Point `point`	Name of the source attach point.	Blank
Target Object `targetId`	Choose a scene entity ID.	Blank
Target Attach Point `targetPoint`	Name of the attach point on the target entity.	Blank
Contact Tolerance `tolerance`	Extra allowed contact distance; use a nonnegative number.	0.0
Detect Between Frames `swept`	Check movement between samples to catch brief contacts.	true

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await WatchAttachPointContacts({"watchId": "", "entityId": "archive_door", "point": ""})
```

In use. Watch tool_touch during a fast tool animation. Listen for OnAttachPointContact, filter the watch ID and the relevant point IDs, and stop the watch when the animation or puzzle phase ends.

StopAttachPointWatch()

Stop Attach Point Watch

Stop a previously started contact watch by its watch ID.

Option key / field	Meaning and input	Default
Watch ID `watchId`	Stable name for this continuing contact watch.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await StopAttachPointWatch({"watchId": ""})
```

In use. Stop tool_touch after the tool has been put away. Use the same watch ID that started it. Removing the visible prop alone is not a clear lifecycle plan for a continuing watch.

AttachObject()

Attach Object

Attach an object to a target object's attach point.

Option key / field	Meaning and input	Default
Object `entityId`	Choose a scene entity ID.	Blank
Attach Point `point`	Name of the source attach point.	Blank
Target Object `targetId`	Choose a scene entity ID.	Blank
Target Attach Point `targetPoint`	Name of the attach point on the target entity.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await AttachObject({"entityId": "archive_door", "point": "", "targetId": "archive_door"})
```

In use. Attach a lantern to the player's hand. Use the prop's source point and the target's grip point when available. Giving an inventory lantern and attaching a visible lantern are separate operations.

DetachObject()

Detach Object

Detach an object that was attached through the attachment system.

Option key / field	Meaning and input	Default
Object `entityId`	Choose a scene entity ID.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await DetachObject({"entityId": "archive_door"})
```

In use. Detach the lantern before placing it on the table, then position it as required. Detaching does not automatically give an inventory item back or select a new animation.

AttachParticleEffect()

Attach Particle Effect

Attach an existing particle-effect entity to a target point with an optional offset.

Option key / field	Meaning and input	Default
Particle Effect `entityId`	Choose a scene entity ID.	Blank
Target Object `targetId`	Choose a scene entity ID.	Blank
Target Attach Point `targetPoint`	Name of the attach point on the target entity.	Blank
Attachment Offset `offset`	Attachment offset as [X, Y, Z].	[0, 0, 0]

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await AttachParticleEffect({
    "entityId": "archive_door",
    "targetId": "archive_door",
    "targetPoint": ""
})
```

In use. Attach smoke to a pipe's smoke_outlet point. Select the scene's particle entity, not just an imported texture used by that emitter. Keep the offset in the intended attachment space.

DetachParticleEffect()

Detach Particle Effect

Detach a particle-effect entity from its target.

Option key / field	Meaning and input	Default
Particle Effect `entityId`	Choose a scene entity ID.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await DetachParticleEffect({"entityId": "archive_door"})
```

In use. Detach a trailing effect when the carried object is put down. If the emitter should stop too, change its emission/enabled state through the appropriate effect setup or supported property action.

Gameplay: GUIs, element state and data-driven widgets

GUI targets have two levels: the interface ID and an element ID inside it. Use the interface-level commands for the whole window and element-level commands for one button, label or widget. Visual Designer preview is not the place to test event execution.

ShowGUI()

Show Gui

Show an authored interface.

Option key / field	Meaning and input	Default
GUI `guiId`	Choose an authored GUI ID.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await ShowGUI({"guiId": "hud"})
```

In use. Show a note-reading panel after the player examines a document. Check its runtime activation, layer and blocking settings. Showing it is different from enabling a disabled interface.

HideGUI()

Hide Gui

Hide an authored interface.

Option key / field	Meaning and input	Default
GUI `guiId`	Choose an authored GUI ID.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await HideGUI({"guiId": "hud"})
```

In use. Hide the note panel on its Close button. It does not erase its definition or necessarily clear the story variables behind it. If another activation rule immediately shows it again, fix that rule rather than adding repeated Hide calls.

EnableGUI()

Enable Gui

Enable an authored interface through the runtime GUI manager.

Option key / field	Meaning and input	Default
GUI `guiId`	Choose an authored GUI ID.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await EnableGUI({"guiId": "hud"})
```

In use. Enable a menu after a sequence that disabled it. Whether it is currently visible is a separate state; use Show GUI as well if it needs to appear.

DisableGUI()

Disable Gui

Disable an authored interface through the runtime GUI manager.

Option key / field	Meaning and input	Default
GUI `guiId`	Choose an authored GUI ID.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await DisableGUI({"guiId": "hud"})
```

In use. Prevent interaction with an interface while a transition is running. Use Hide GUI if the intention is to remove it from view. Test the final disabled appearance in the running game.

ShowGUIElement()

Show Gui Element

Show one element inside an authored GUI.

Option key / field	Meaning and input	Default
GUI `guiId`	Choose an authored GUI ID.	Blank
Element / Group `elementId`	Choose an element inside the selected GUI.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await ShowGUIElement({"guiId": "hud", "elementId": "topline"})
```

In use. Reveal an archive tab button after the player finds the key. Select both the owning GUI and that element. The parent GUI and any parent panel must also be visible.

HideGUIElement()

Hide Gui Element

Hide one element inside an authored GUI.

Option key / field	Meaning and input	Default
GUI `guiId`	Choose an authored GUI ID.	Blank
Element / Group `elementId`	Choose an element inside the selected GUI.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await HideGUIElement({"guiId": "hud", "elementId": "topline"})
```

In use. Hide a temporary objective label after completion. A hidden child cannot be made visible merely by showing its parent if its own visible flag remains off.

EnableGUIElement()

Enable Gui Element

Enable one element for interaction.

Option key / field	Meaning and input	Default
GUI `guiId`	Choose an authored GUI ID.	Blank
Element / Group `elementId`	Choose an element inside the selected GUI.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await EnableGUIElement({"guiId": "hud", "elementId": "topline"})
```

In use. Enable the Continue button once the necessary option has been selected. A parent interface or modal input lock can still prevent access even when this child's enabled flag is on.

DisableGUIElement()

Disable Gui Element

Disable one element for interaction.

Option key / field	Meaning and input	Default
GUI `guiId`	Choose an authored GUI ID.	Blank
Element / Group `elementId`	Choose an element inside the selected GUI.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await DisableGUIElement({"guiId": "hud", "elementId": "topline"})
```

In use. Disable Save until a save slot has been selected. Decide what visual disabled treatment the element should use so the player understands why it cannot be used.

ChangeGUIProperty()

Change Gui Property

Change a supported property on an authored GUI.

Option key / field	Meaning and input	Default
GUI `guiId`	Choose an authored GUI ID.	Blank
Property `property`	Choose a GUI property.	Blank
To `value`	A value of the type required by the selected variable, operation or property.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await ChangeGUIProperty({"guiId": "hud", "property": "visible", "value": true})
```

In use. Change a window's runtime appearance or layout using a property offered by the picker. Match the value type to the property. This is a runtime edit, not a request to rewrite the editor's theme.

ChangeGUIElementProperty()

Change Gui Element Property

Change a supported property on a GUI element.

Option key / field	Meaning and input	Default
GUI `guiId`	Choose an authored GUI ID.	Blank
Element / Group `elementId`	Choose an element inside the selected GUI.	Blank
Property `property`	Choose an element property.	Blank
To `value`	A value of the type required by the selected variable, operation or property.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await ChangeGUIElementProperty({"guiId": "hud", "elementId": "topline", "property": "text"})
```

In use. Change the status label's text after the signal is powered. Use the correct property key and value type. In a GUI event's script context, elementId this can refer to the calling element; a global event usually has no such caller.

ListBoxAddItem()

List Box: Add Item

Add a manual list-box item with an ID, text, optional icon, disabled state and insertion index.

Option key / field	Meaning and input	Default
GUI `guiId`	Choose an authored GUI ID.	Blank
List Box `elementId`	Choose an element inside the selected GUI.	Blank
New Item ID `itemId`	Text.	Blank
Item Text `text`	Text.	Blank
Optional Icon `iconAssetId`	Choose an imported image. May be blank.	Blank
Disabled `disabled`	On/off.	true
Insert Index (-1 = End) `insertIndex`	Zero-based insertion position; -1 appends to the list.	-1

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await ListBoxAddItem({"guiId": "hud", "elementId": "topline", "itemId": "key"})
```

In use. Add a discovered location to a travel list with a stable ID. An insertion index of -1 appends. Turn Disabled off when the player should be able to select the row. For list boxes bound to an array or inventory, change the data source rather than expecting manual entries to override it permanently.

ListBoxDeleteItem()

List Box: Delete Item

Delete a list-box item by its item ID.

Option key / field	Meaning and input	Default
GUI `guiId`	Choose an authored GUI ID.	Blank
List Box `elementId`	Choose an element inside the selected GUI.	Blank
Item ID `itemId`	Text.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await ListBoxDeleteItem({"guiId": "hud", "elementId": "topline", "itemId": "key"})
```

In use. Remove a destination that is no longer available. The item's ID is not its visible label or a row number. Consider disabling it instead when the player should still see that it exists.

ListBoxModifyItem()

List Box: Modify Item

Change an existing list item's text, ID, icon, disabled or selected property.

Option key / field	Meaning and input	Default
GUI `guiId`	Choose an authored GUI ID.	Blank
List Box `elementId`	Choose an element inside the selected GUI.	Blank
Item ID `itemId`	Text.	Blank
Property `property`	Choose: text, id, iconAssetId, disabled, selected.	"text"
New Value `value`	A value of the type required by the selected variable, operation or property.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await ListBoxModifyItem({"guiId": "hud", "elementId": "topline", "itemId": "key"})
```

In use. Disable a travel destination during a storm. Changing an item ID can break later lookups, so treat IDs as stable references whenever possible.

ListBoxClearItems()

List Box: Clear Items

Clear the manual items in a list box.

Option key / field	Meaning and input	Default
GUI `guiId`	Choose an authored GUI ID.	Blank
List Box `elementId`	Choose an element inside the selected GUI.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await ListBoxClearItems({"guiId": "hud", "elementId": "topline"})
```

In use. Empty a results list before rebuilding it. An array- or inventory-backed list may repopulate from its source; clear or change that source when appropriate.

CarouselAddImage()

Carousel: Add Image

Add an imported image asset to an image carousel.

Option key / field	Meaning and input	Default
GUI `guiId`	Choose an authored GUI ID.	Blank
Image Carousel `elementId`	Choose an element inside the selected GUI.	Blank
Image Asset `assetId`	Choose an imported image.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await CarouselAddImage({"guiId": "hud", "elementId": "topline", "assetId": "your_asset"})
```

In use. Add a newly discovered photograph to a gallery. Choose the GUI and carousel element, then the image asset. This is not a command to create a new image file.

CarouselRemoveImage()

Carousel: Remove Image

Remove a selected image asset from a carousel's authored/runtime list.

Option key / field	Meaning and input	Default
GUI `guiId`	Choose an authored GUI ID.	Blank
Image Carousel `elementId`	Choose an element inside the selected GUI.	Blank
Image Asset `assetId`	Choose an imported image.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await CarouselRemoveImage({"guiId": "hud", "elementId": "topline", "assetId": "your_asset"})
```

In use. Remove a temporary clue image when the inspection ends. Check what happens if it was the current slide, and test the first/last-image behavior.

CarouselClearImages()

Carousel: Clear Images

Clear a carousel's image list.

Option key / field	Meaning and input	Default
GUI `guiId`	Choose an authored GUI ID.	Blank
Image Carousel `elementId`	Choose an element inside the selected GUI.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await CarouselClearImages({"guiId": "hud", "elementId": "topline"})
```

In use. Reset a manual inspection gallery before filling it with another object's images. Inventory-driven carousels can obtain their contents from inventory instead; update the appropriate data source.

CarouselPrevious()

Carousel: Previous Image

Move a carousel to its previous image.

Option key / field	Meaning and input	Default
GUI `guiId`	Choose an authored GUI ID.	Blank
Carousel `elementId`	Choose an element inside the selected GUI.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await CarouselPrevious({"guiId": "hud", "elementId": "topline"})
```

In use. Wire a left-arrow button to the chosen carousel. Wrapping, animation and layout remain properties of that carousel.

CarouselNext()

Carousel: Next Image

Move a carousel to its next image.

Option key / field	Meaning and input	Default
GUI `guiId`	Choose an authored GUI ID.	Blank
Carousel `elementId`	Choose an element inside the selected GUI.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await CarouselNext({"guiId": "hud", "elementId": "topline"})
```

In use. Wire a right-arrow button to the chosen carousel. This navigates the widget, rather than advancing dialogue or changing scenes.

SaveGameWidgetCommand()

Save Game Widget: Command

Control a save widget's navigation, selection or selected/explicit-slot operations.

Option key / field	Meaning and input	Default
GUI `guiId`	Choose an authored GUI ID.	Blank
Save Game Widget `elementId`	Choose an element inside the selected GUI.	Blank
Command `command`	Choose: previous, next, up, down, refresh, select-first, select-previous, select-next, load-selected, save-selected, rename-selected, delete-selected, select-slot, load-slot, save-slot, rename-slot, delete-slot.	"previous"
Explicit Slot (-1 = Selected) `slot`	Explicit save slot number. Selected-slot widget commands use widget selection instead.	-1

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SaveGameWidgetCommand({"guiId": "hud", "elementId": "topline", "command": "previous"})
```

In use. Use select-slot with a slot number, or save-selected after the player picks a row. Commands ending in -selected use widget selection; commands ending in -slot use Slot. Test overwrite, rename and delete behavior on disposable saves.

CenterGUIModel()

Center GUI Model

Fit and center the model displayed inside a GUI 3D model viewer.

Option key / field	Meaning and input	Default
GUI `guiId`	Choose an authored GUI ID.	Blank
3D Model `elementId`	Choose an element inside the selected GUI.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await CenterGUIModel({"guiId": "hud", "elementId": "topline"})
```

In use. Offer a Reset View button after the player rotates a model. It targets the GUI viewer, not the scene's active gameplay camera or the model's source file.

PlayGUIModelAnimation()

Play GUI Model Animation

Play a named animation in a GUI model viewer, with loop and speed controls.

Option key / field	Meaning and input	Default
GUI `guiId`	Choose an authored GUI ID.	Blank
3D Model `elementId`	Choose an element inside the selected GUI.	Blank
Animation `animation`	Text.	Blank
Loop `loop`	On/off.	true
Speed `speed`	Playback speed multiplier; 1 is normal speed.	1.0

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await PlayGUIModelAnimation({"guiId": "hud", "elementId": "topline", "animation": ""})
```

In use. Play an idle loop in a character inspection window. Match the imported animation name. A model with no animation clips cannot gain one just by entering a name here.

Gameplay: Control profiles, verbs and pointer behavior

These commands affect how the player operates the game. Movement profile, interaction mode, input device and visibility of the pointer are separate settings. Provide a way back when an experiment changes the controls.

SetRuntimeControlsEnabled()

Set Runtime Controls Enabled

Enable or disable the runtime control-profile system.

Option key / field	Meaning and input	Default
Enabled `enabled`	Explicit on/off state for the feature or response.	true

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SetRuntimeControlsEnabled({"enabled": true})
```

In use. Enable the newer adaptive controls for a sequence that uses device switching. Disabling the profile system is not identical to Set Player Input Enabled false; it changes which control machinery is in use.

SetMovementProfile()

Set Movement Profile

Choose legacy, point-and-click, direct, first-person or drag-to-move movement.

Option key / field	Meaning and input	Default
Movement Profile `profile`	Choose: legacy, point-and-click, direct, first-person, drag-to-move.	"point-and-click"

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SetMovementProfile({"profile": "point-and-click"})
```

In use. Switch to direct movement for a short action scene and restore point-and-click afterward. Navigation, camera and input mappings still need to suit the chosen profile.

SetInteractionMode()

Set Interaction Mode

Choose the runtime hotspot interaction style, including context, verb, direct, cycling cursor and verb-coin variants.

Option key / field	Meaning and input	Default
Interaction Mode `mode`	Choose: context, verb-single-click, verb-double-click, verb-direct, verb-direct-double-click, direct, cycling-cursor, verb-coin, verb-coin-first.	"context"

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SetInteractionMode({"mode": "context"})
```

In use. Use verb-coin for a compact interaction menu. Changing the mode does not create missing verb responses on objects. Test empty-ground clicks, inventory combinations and the device the player is actually using.

SetAutoInputSwitching()

Automatic Input Switching

Enable or disable automatic input-device switching.

Option key / field	Meaning and input	Default
Enabled `enabled`	Explicit on/off state for the feature or response.	true

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SetAutoInputSwitching({"enabled": true})
```

In use. Turn it on for a game that should adapt between mouse and gamepad. Turn it off during a fixed-device control demonstration. Device detection and movement-profile switching are related settings, not the same switch.

SetInputDevice()

Set Input Device

Select auto, mouse-keyboard, gamepad or touch input mode.

Option key / field	Meaning and input	Default
Input Device `device`	Choose: auto, mouse-keyboard, gamepad, touch.	"auto"

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SetInputDevice({"device": "auto"})
```

In use. Choose Auto for normal play or force Gamepad while testing controller prompts. Forcing a device is not a substitute for having usable bindings and an actual controller available.

SetProximityInteractionEnabled()

Proximity Interaction

Enable or disable nearest/proximity-based interaction.

Option key / field	Meaning and input	Default
Enabled `enabled`	Explicit on/off state for the feature or response.	true

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SetProximityInteractionEnabled({"enabled": true})
```

In use. Enable it for controller play in a small room. Check proximity radius, hotspot eligibility and line-of-play expectations so the nearest usable target is the one the player expects.

SetControlSetting()

Set Control Setting

Change one supported runtime control setting with a value of the matching type.

Option key / field	Meaning and input	Default
Setting `setting`	Choose: enabled, cursorAdaptation, cyclingEnabled, movementProfile, autoSwitchInput, inputDevice, autoSwitchMovement, mouseProfile, gamepadProfile, touchProfile, proximityInteraction, proximityRadius, showHints, controllerIcons, coinEnabled, dragThreshold, gamepadLookSpeed.	"coinEnabled"
Value `value`	A value of the type required by the selected variable, operation or property.	true

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SetControlSetting({"setting": "coinEnabled", "value": true})
```

In use. Set proximityRadius to a number or coinEnabled to a Boolean. The same Value box accepts different types depending on Setting; the text "false" is not the same as Boolean false.

GetControlState()

Get Control State

Read the current control-profile state and store it in a game variable.

Option key / field	Meaning and input	Default
Result Variable `resultVariable`	Name of the game variable that receives the returned value. Custom values are accepted.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await GetControlState({"resultVariable": ""})
```

In use. Store controls_before before entering an inspection mode and inspect the returned state in a script. The result is structured state, not a single enabled flag or an automatic restore command.

OpenInteractionCoin()

Open Interaction Coin

Open the interaction coin for a target entity when that feature is configured.

Option key / field	Meaning and input	Default
Object / Character `entityId`	Choose a scene entity ID. May be blank.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await OpenInteractionCoin({"entityId": "archive_door"})
```

In use. Open a coin from a dedicated Inspect button or a hotspot interaction. The target must have useful verbs. A coin opened with no meaningful target cannot invent responses.

CloseInteractionCoin()

Close Interaction Coin

Close the currently open interaction coin.

No options in the standard catalog schema.

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await CloseInteractionCoin({})
```

In use. Close the coin when a menu takes over. It hides the coin, rather than disabling the feature for the rest of the game.

CycleInteractionVerb()

Cycle Verb

Move forward or backward through the runtime verb cycle.

Option key / field	Meaning and input	Default
Direction `direction`	Whole number. Range: -1 to 1.	1

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await CycleInteractionVerb({"direction": 1})
```

In use. Use direction 1 for the next verb and -1 for the previous one. A value of 0 does not mean "first verb." The cycle depends on the enabled verbs and current interaction settings.

InteractNearest()

Interact with Nearest Hotspot

Perform a verb interaction with the nearest eligible hotspot under the proximity system.

Option key / field	Meaning and input	Default
Verb `verbId`	Choose a Verb Designer ID. May be blank.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await InteractNearest({"verbId": ""})
```

In use. Use it for a controller interaction button. Select a verb or leave the optional verb selection to the current behavior. Test target selection in a crowded room.

SetMouseClicksEnabled()

Enable Mouse Clicks

Enable or disable the extension's mouse-click interaction switch.

Option key / field	Meaning and input	Default
Enabled `enabled`	Explicit on/off state for the feature or response.	true

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SetMouseClicksEnabled({"enabled": true})
```

In use. Disable scene clicks while a special control sequence owns the pointer. It is a different switch from hiding the cursor. Restore it on every completion and cancellation path.

SetMousePointingEnabled()

Enable Mouse Pointing

Enable or disable mouse pointing/hover behavior through the extension.

Option key / field	Meaning and input	Default
Enabled `enabled`	Explicit on/off state for the feature or response.	true

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SetMousePointingEnabled({"enabled": true})
```

In use. Disable hover targeting during a locked cinematic view. This does not necessarily remove every GUI interaction or keyboard route; use the appropriate broader input lock when needed.

SetMousePointerVisible()

Show Mouse Pointer

Show or hide the mouse pointer.

Option key / field	Meaning and input	Default
Visible `visible`	Explicit visible/hidden state.	true

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SetMousePointerVisible({"visible": true})
```

In use. Hide it for a cinematic, then show it when play returns. A hidden cursor can still leave other controls active, so do not use this as an input lock.

ChangeVerb()

Change Current Verb Command

Select a verb from the project's Verb Designer.

Option key / field	Meaning and input	Default
New Verb `verbId`	Choose a Verb Designer ID.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await ChangeVerb({"verbId": ""})
```

In use. Select Look when an inspection interface opens. The verb ID is stored even when the display name is translated or renamed. Choosing the verb alone does not execute it on an object.

SetVerbUsageEnabled()

Enable / Disable Verb Command Usage

Enable or disable verb usage at runtime.

Option key / field	Meaning and input	Default
Enabled `enabled`	Explicit on/off state for the feature or response.	true

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SetVerbUsageEnabled({"enabled": true})
```

In use. Disable normal verb selection while a special puzzle interface is active, then restore it. It does not remove authored verb definitions or their response lists.

ChangeGameCursor()

Change Game Cursor

Change the runtime cursor to an authored cursor definition.

Option key / field	Meaning and input	Default
Cursor `cursorId`	Choose a Cursor Designer ID.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await ChangeGameCursor({"cursorId": ""})
```

In use. Use a magnifier cursor for inspection. Hover feedback and interaction-mode logic may select other cursor states, so test the change while pointing at both ground and hotspots.

Gameplay: Audio, video, FMV and recorded gameplay

Sound effects, music, voice and video use separate playback routes. A sound channel gives you a handle for stopping, pausing or fading a group of effects. Starting playback usually lets the next instruction run immediately.

PlaySound()

Play Sound

Start an imported audio asset as a sound effect, optionally on a named channel.

Option key / field	Meaning and input	Default
Sound Asset `assetId`	Choose an imported audio asset.	Blank
Channel / ID `channel`	Sound-effect channel name. Reuse it to pause, stop or fade that sound group.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await PlaySound({"assetId": "chime", "channel": "puzzle"})
```

In use. Play chime on puzzle, then continue the list. Use that same channel to stop or fade the effect later. A blank channel has broader/default behavior; name important continuing sounds.

StopSound()

Stop Sound

Stop effects selected by channel.

Option key / field	Meaning and input	Default
Channel / ID `channel`	Sound-effect channel name. Reuse it to pause, stop or fade that sound group.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await StopSound({"channel": "puzzle"})
```

In use. Stop the machine channel when the power is cut. Use a specific channel so unrelated effects can continue. This does not stop music or voice.

PauseSound()

Sound: Pause

Pause effects selected by channel.

Option key / field	Meaning and input	Default
Channel / ID (empty = all) `channel`	Sound-effect channel name. Reuse it to pause, stop or fade that sound group. May be blank.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await PauseSound({"channel": "puzzle"})
```

In use. Pause a repeating machine sound during an inspection that freezes its motion. Pausing is reversible; stopping ends playback instead.

ResumeSound()

Sound: Resume

Resume paused effects selected by channel.

Option key / field	Meaning and input	Default
Channel / ID (empty = all) `channel`	Sound-effect channel name. Reuse it to pause, stop or fade that sound group. May be blank.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await ResumeSound({"channel": "puzzle"})
```

In use. Resume the machine channel together with the machine's motion. It resumes existing players; it does not choose or reload an audio asset.

FadeSound()

Sound: Fade

Fade a sound channel to a normalized volume over time, waiting for the fade routine.

Option key / field	Meaning and input	Default
Channel / ID (empty = all) `channel`	Sound-effect channel name. Reuse it to pause, stop or fade that sound group. May be blank.	Blank
Target Volume `volume`	Normalized target volume: 0 is silent, 1 is full volume. Range: 0.0 to 1.0.	0.0
Duration (seconds) `duration`	Time in seconds for this action. Read its timing note before assuming it waits. Range: 0.0 to 60.0.	1.0

Timing. Waits for the fade routine.

```
await FadeSound({"channel": "puzzle", "volume": 0.0, "duration": 1})
```

In use. Fade ambience to 0 over 1.5 seconds before stopping it. Volume uses 0 to 1, not decibels. A final volume near zero stops the selected sound players.

PlayMusic()

Play Music

Start the chosen audio asset as music.

Option key / field	Meaning and input	Default
Music Asset `assetId`	Choose an imported audio asset.	Blank
Skippable `skippable`	On/off.	true

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await PlayMusic({"assetId": "chime", "skippable": true})
```

In use. Start a room theme and let the following actions continue. The catalog includes Skippable, but the current action bridge does not use that field to implement music skipping; do not build a skip button around it.

StopMusic()

Stop Music

Stop the current music track.

No options in the standard catalog schema.

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await StopMusic({})
```

In use. Stop the room theme before a silent reveal. It is music-specific and leaves separate effects and voice alone.

PauseMusic()

Music: Pause

Pause the current music track.

No options in the standard catalog schema.

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await PauseMusic({})
```

In use. Pause music while an in-game device temporarily takes over the soundtrack. The track position is kept for Resume Music.

ResumeMusic()

Music: Resume

Resume the paused music track.

No options in the standard catalog schema.

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await ResumeMusic({})
```

In use. Use it after the device view closes. If another action replaced the track, the old track is not automatically restored.

FadeMusic()

Music: Fade

Fade the current music to a normalized volume, waiting for the fade routine.

Option key / field	Meaning and input	Default
Target Volume `volume`	Normalized target volume: 0 is silent, 1 is full volume. Range: 0.0 to 1.0.	0.0
Duration (seconds) `duration`	Time in seconds for this action. Read its timing note before assuming it waits. Range: 0.0 to 60.0.	1.0

Timing. Waits for the fade routine.

```
await FadeMusic({"volume": 0.0, "duration": 1})
```

In use. Fade to 0 over 2 seconds for a room ending, or to a quieter nonzero level under narration. A zero target stops playback; volume is 0 to 1.

SeekMusic()

Skip Ahead in Soundtrack

Move playing music to minutes plus seconds.

Option key / field	Meaning and input	Default
Minute `minutes`	Whole minutes; added to Seconds to form a playback position.	0
Second `seconds`	Number; decimals allowed.	0

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SeekMusic({"minutes": 0, "seconds": 1})
```

In use. Seek to 1 minute 10 seconds to check a cue. This is media playback positioning, not a timeline scheduler or a music crossfade.

StopStartupSplashAudio()

Stop Startup Splash Audio

Stop audio still playing from startup splash presentation.

No options in the standard catalog schema.

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await StopStartupSplashAudio({})
```

In use. Use it when the first playable scene should take over the soundtrack. It does not disable splash screens or erase the splash audio asset.

PlayVideo()

Play Video

Start an imported video with full-screen, skippable and muted options.

Option key / field	Meaning and input	Default
Video Asset `assetId`	Choose an imported video.	Blank
Full Screen `fullScreen`	Use the video's full-screen presentation route.	true
Skippable `skippable`	On/off.	true
Muted `muted`	Play the video without its audio.	true

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await PlayVideo({"assetId": "your_asset", "fullScreen": true, "skippable": true})
```

In use. Show a short reel, then use OnVideoEnd if later logic depends on natural completion. Starting the video is not a promise that the next action waits for the whole clip.

StopVideo()

Stop Video

Stop the current video playback.

No options in the standard catalog schema.

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await StopVideo({})
```

In use. Use it to cancel a video inspection. Test the cancel path separately from natural completion; an explicit stop should not be assumed to raise the same completion event.

SeekVideo()

Seek Video

Seek the current video to minutes plus seconds.

Option key / field	Meaning and input	Default
Minute `minutes`	Whole minutes; added to Seconds to form a playback position.	0
Second `seconds`	Number; decimals allowed.	0

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SeekVideo({"minutes": 0, "seconds": 1})
```

In use. Seek to 0 minutes 12 seconds while testing a clip. Playback and seeking depend on the imported format and decoder; test the build you intend to distribute.

StartFMV()

Start FMV

Start the extension's FMV playback route with an end, point or interrupt continuation mode and optional script.

Option key / field	Meaning and input	Default
Movie Asset ID `assetId`	Choose an imported video.	Blank
Playback Limit `mode`	Choose: end, point, interrupt.	"end"
Play Until (seconds) `endTime`	FMV point-mode endpoint; this differs from the seek destination. Range: 0 to 86400.	0
Completion Script `scriptId`	Choose a visual or supported text script. May be blank.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await StartFMV({"assetId": "your_asset", "mode": "end", "endTime": 0})
```

In use. Play until a chosen point, then call the script that presents the next interactive decision. Configure the video and continuation intentionally. FMV is a media-driven interaction route, not the Cutscene Timeline itself.

StopFMV()

Stop FMV

Stop the current FMV playback route.

No options in the standard catalog schema.

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await StopFMV({})
```

In use. Use it when leaving an FMV sequence entirely. Decide what room, interface or script takes over after cancellation.

PauseFMV()

Pause FMV

Pause the current FMV route.

No options in the standard catalog schema.

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await PauseFMV({})
```

In use. Pause the clip while a decision is visible. This does not pause every other game system automatically.

ResumeFMV()

Resume FMV

Resume FMV with the chosen continuation mode, endpoint and script.

Option key / field	Meaning and input	Default
Playback Limit `mode`	Choose: end, point, interrupt.	"end"
Play Until (seconds) `endTime`	FMV point-mode endpoint; this differs from the seek destination. Range: 0 to 86400.	0
Completion Script `scriptId`	Choose a visual or supported text script. May be blank.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await ResumeFMV({"mode": "end", "endTime": 0, "scriptId": "utility_gd"})
```

In use. Continue after the player answers a timed question. Recheck endTime and the chosen script so resuming does not immediately reach the same stop point again.

SeekFMV()

Seek FMV

Move FMV to a time and set its continuation behavior.

Option key / field	Meaning and input	Default
Time `time`	FMV seek destination in seconds. Range: 0 to 86400.	0
Playback Limit `mode`	Choose: end, point, interrupt.	"end"
Play Until (seconds) `endTime`	FMV point-mode endpoint; this differs from the seek destination. Range: 0 to 86400.	0
Completion Script `scriptId`	Choose a visual or supported text script. May be blank.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SeekFMV({"time": 0, "mode": "end", "endTime": 0})
```

In use. Jump to a branch of an interactive film. Use Time for the seek destination and End Time for point-mode continuation; they describe different moments.

StartGameplayRecording()

Start Gameplay Recording

Begin the extension's gameplay-recording route using a writable JSON path.

Option key / field	Meaning and input	Default
Recording Path `path`	Writable recording-data path, such as user://walkthrough.json.	"user://walkthrough.json"

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await StartGameplayRecording({"path": "user://walkthrough.json"})
```

In use. Record a repeatable walk through a puzzle for testing. This creates gameplay data, not a finished MP4 screen recording with sound. Keep the path writable in exported builds.

StopGameplayRecording()

Stop Gameplay Recording

Stop the active gameplay recording.

No options in the standard catalog schema.

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await StopGameplayRecording({})
```

In use. End the test run cleanly before starting another recording. It does not stop a separate operating-system video capture.

PlayGameplayRecording()

Play Gameplay Recording

Play a saved gameplay recording at the chosen speed.

Option key / field	Meaning and input	Default
Recording Path `path`	Writable recording-data path, such as user://walkthrough.json.	"user://walkthrough.json"
Playback Speed `speed`	Playback speed multiplier; 1 is normal speed.	1

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await PlayGameplayRecording({"path": "user://walkthrough.json", "speed": 1})
```

In use. Replay user://walkthrough.json while checking a room. The recording needs a compatible game state and scene setup; it is not a self-contained replacement for the project and its assets.

StopGameplayPlayback()

Stop Gameplay Playback

Stop playback of recorded gameplay data.

No options in the standard catalog schema.

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await StopGameplayPlayback({})
```

In use. Use it before handing control back to the player or starting another test run. Check the active camera and input state after stopping a scripted replay.

Gameplay: Quests and objectives

Quest actions target definitions from the Quest Designer. Quest, stage and objective IDs form a hierarchy. Use the pickers in that order so a valid objective is selected within the intended stage.

StartQuest()

Start Quest

Start a quest and optionally select its initial stage.

Option key / field	Meaning and input	Default
Quest `questId`	Choose a Quest Designer ID.	Blank
Initial Stage (optional) `stageId`	Choose a stage in that quest. May be blank.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await StartQuest({"questId": "", "stageId": ""})
```

In use. Start Restore the Signal when the player accepts the task. A quest definition must already exist. This action does not create a journal widget or force a hidden journal interface onto the screen.

SetQuestStage()

Quest: Set Stage

Move a quest to a chosen stage.

Option key / field	Meaning and input	Default
Quest `questId`	Choose a Quest Designer ID.	Blank
Stage `stageId`	Choose a stage in that quest.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SetQuestStage({"questId": "", "stageId": ""})
```

In use. Advance from Restore Power to Enter the Archive after the lever succeeds. Check whether earlier objectives should be completed before moving on, rather than assuming the stage change supplies all the story bookkeeping.

CompleteObjective()

Quest: Complete Objective

Complete a named objective, optionally selecting a stage first.

Option key / field	Meaning and input	Default
Quest `questId`	Choose a Quest Designer ID.	Blank
Stage `stageId`	Choose a stage in that quest.	Blank
Objective `objectiveId`	Choose an objective in that stage.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await CompleteObjective({"questId": "", "stageId": "", "objectiveId": ""})
```

In use. Complete power inside the restore stage after the signal turns on. A supplied stage ID is applied before the objective operation, so choose it deliberately.

FailObjective()

Quest: Fail Objective

Mark an objective as failed, optionally selecting its stage first.

Option key / field	Meaning and input	Default
Quest `questId`	Choose a Quest Designer ID.	Blank
Stage `stageId`	Choose a stage in that quest.	Blank
Objective `objectiveId`	Choose an objective in that stage.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await FailObjective({"questId": "", "stageId": "", "objectiveId": ""})
```

In use. Fail a timed optional objective when its timer expires. Keep failure consequences in a separate clear branch so a failed side objective does not accidentally end the whole quest.

ResetObjective()

Quest: Reset Objective

Reset one objective to its unfinished state, optionally selecting its stage first.

Option key / field	Meaning and input	Default
Quest `questId`	Choose a Quest Designer ID.	Blank
Stage `stageId`	Choose a stage in that quest.	Blank
Objective `objectiveId`	Choose an objective in that stage.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await ResetObjective({"questId": "", "stageId": "", "objectiveId": ""})
```

In use. Reset a repeatable practice task. It does not automatically restore consumed items, move characters or undo the rest of the room state.

CompleteQuest()

Quest: Complete Quest

Mark the quest completed.

Option key / field	Meaning and input	Default
Quest `questId`	Choose a Quest Designer ID.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await CompleteQuest({"questId": ""})
```

In use. Complete Restore the Signal after the archive has been reached. Award score or items as separate guarded actions. Completing a quest is not automatic proof that a reward has not already been given.

FailQuest()

Quest: Fail Quest

Mark the quest failed.

Option key / field	Meaning and input	Default
Quest `questId`	Choose a Quest Designer ID.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await FailQuest({"questId": ""})
```

In use. Fail a side quest after its story window closes. Provide a readable journal entry or message explaining the result; the state change alone may not tell the player what happened.

ResetQuest()

Quest: Reset Quest

Reset the quest state through the quest manager.

Option key / field	Meaning and input	Default
Quest `questId`	Choose a Quest Designer ID.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await ResetQuest({"questId": ""})
```

In use. Restart a repeatable challenge. Reconcile its room state, inventory and timers as well; resetting the journal state does not rebuild the whole puzzle.

SetQuestHidden()

Quest: Hide / Unhide

Set a quest's hidden state.

Option key / field	Meaning and input	Default
Quest `questId`	Choose a Quest Designer ID.	Blank
Hidden `hidden`	Whether the quest or player visual should be hidden for this action.	true

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SetQuestHidden({"questId": "", "hidden": true})
```

In use. Reveal a secret quest once its first clue is found. Hidden controls journal visibility policy, rather than completing or deleting the quest.

SetQuestTracked()

Quest: Track / Untrack

Set whether a quest is tracked.

Option key / field	Meaning and input	Default
Quest `questId`	Choose a Quest Designer ID.	Blank
Tracked `tracked`	On/off.	true

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SetQuestTracked({"questId": "", "tracked": true})
```

In use. Track the active investigation after the player accepts it. The GUI must contain a suitable tracked-quest display if that state is meant to be visible.

ChangeQuestProperty()

Quest: Change Property

Change a supported quest property such as title, description, category or flags.

Option key / field	Meaning and input	Default
Quest `questId`	Choose a Quest Designer ID.	Blank
Quest Property `property`	Choose a quest property.	Blank
New Value `value`	A value of the type required by the selected variable, operation or property.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await ChangeQuestProperty({
    "questId": "",
    "property": "description",
    "value": "The signal is working."
})
```

In use. Replace the quest description after a revelation. Choose a Boolean for Boolean properties and text for text properties; use dedicated completion actions for progression.

ChangeQuestStageProperty()

Quest: Change Stage Property

Change a supported property's value on one quest stage.

Option key / field	Meaning and input	Default
Quest `questId`	Choose a Quest Designer ID.	Blank
Stage `stageId`	Choose a stage in that quest.	Blank
Stage Property `property`	Choose a stage property.	Blank
New Value `value`	A value of the type required by the selected variable, operation or property.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await ChangeQuestStageProperty({"questId": "", "stageId": "", "property": "description"})
```

In use. Update a stage's descriptive text as new evidence is found. This edits presentation/data; Set Quest Stage is the command that makes a different stage current.

ChangeQuestObjectiveProperty()

Quest: Change Objective Property

Change a supported property such as text, optional or hidden on one objective.

Option key / field	Meaning and input	Default
Quest `questId`	Choose a Quest Designer ID.	Blank
Stage `stageId`	Choose a stage in that quest.	Blank
Objective `objectiveId`	Choose an objective in that stage.	Blank
Objective Property `property`	Choose an objective property.	Blank
New Value `value`	A value of the type required by the selected variable, operation or property.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await ChangeQuestObjectiveProperty({
    "questId": "",
    "stageId": "",
    "objectiveId": "",
    "property": "text"
})
```

In use. Reveal an optional objective after the player reads a note. Choose the correct quest and stage before the objective. Completing an objective remains a separate operation.

Gameplay: Saving, language and project settings

Runtime settings and save files are different kinds of persistence. Test saves with a fresh game, and use the same slot numbering consistently in scripts and custom save widgets.

SaveGame()

Save Game

Save through the basic runtime route. With no structured slot supplied, this route uses slot 0.

No options in the standard catalog schema.

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SaveGame({})
```

In use. Use it only when a deliberate quick-save slot is wanted. The normal catalog form has no fields. Choose Save Game Slot when the player or script must select a slot and a name.

LoadGame()

Load Game

Load through the basic runtime route, defaulting to slot 0.

No options in the standard catalog schema.

Timing. Awaits loading.

```
await LoadGame({})
```

In use. Use Load Game Slot for an explicit slot or a custom load screen. Loading replaces important live state; keep later actions in the old sequence to a minimum and test missing-file behavior.

SaveGameSlot()

Save Game Slot

Save the current game into a chosen slot with an optional name.

Option key / field	Meaning and input	Default
Slot `slot`	Explicit save slot number. Selected-slot widget commands use widget selection instead.	0
Save Name (optional) `saveName`	Optional player-facing name for the saved game. May be blank.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SaveGameSlot({"slot": 0, "saveName": ""})
```

In use. Save slot 1 as "Before the archive." Save a coherent completed puzzle state, rather than halfway through a multi-step reward. Do not silently overwrite a player-chosen slot without a sensible interface policy.

LoadGameSlot()

Load Game Slot

Load the chosen save slot.

Option key / field	Meaning and input	Default
Slot `slot`	Explicit save slot number. Selected-slot widget commands use widget selection instead.	0

Timing. Awaits loading.

```
await LoadGameSlot({"slot": 0})
```

In use. Load the row the player selected in a custom save screen. Test an empty slot and a save from another room. OnSaveLoaded is the hook for follow-up work that belongs to successful loading.

SetSaveSlotImage()

Set Save Slot Image

Set the image override used for save-slot presentation through the extension.

Option key / field	Meaning and input	Default
Save Slot Image `assetId`	Choose an imported image.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SetSaveSlotImage({"assetId": "your_asset"})
```

In use. Choose an authored chapter illustration instead of a live screenshot when appropriate. The asset must be an imported image. It does not create a new save or choose which slot gets saved.

SetGameLanguage()

Set Game Language

Set the game text language, optionally persisting the selection.

Option key / field	Meaning and input	Default
Language Code `language`	Language code used by the project, such as en.	Blank
Save Player Preference `persist`	Remember this language preference beyond the immediate runtime change.	true

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await SetGameLanguage({"language": "", "persist": true})
```

In use. Choose a language code present in the project's translations. Keep Persist on for a player preference that should survive later sessions. This does not translate text that was never entered into the translation workflow.

ChangeVoiceLanguage()

Change Voice Language

Set the runtime voice language.

Option key / field	Meaning and input	Default
Language `language`	Language code used by the project, such as en.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await ChangeVoiceLanguage({"language": ""})
```

In use. Choose the voice-language code that has the corresponding audio resources. It is separate from subtitle language, allowing one voice language with another subtitle language.

ChangeSubtitleLanguage()

Change Subtitle Language

Set the runtime subtitle language.

Option key / field	Meaning and input	Default
Language `language`	Language code used by the project, such as en.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await ChangeSubtitleLanguage({"language": ""})
```

In use. Change subtitles while leaving voice in the original language. Test custom GUI text and dynamically created lines too; fixed strings outside the translation system may not change automatically.

ChangeProjectProperty()

Change Project Property

Assign a supported project property through a property path.

Option key / field	Meaning and input	Default
Property `property`	Choose a project property path.	Blank
To `value`	A value of the type required by the selected variable, operation or property.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await ChangeProjectProperty({"property": "meta.title", "value": "Workshop"})
```

In use. Change a runtime project option offered by the picker. Match the value type and test whether the relevant manager reads it immediately. This is not an instruction to rewrite the saved editor project file.

ChangeGameSetting()

Change Game Setting

Assign a supported runtime game setting.

Option key / field	Meaning and input	Default
Setting `setting`	Choose a runtime game setting.	Blank
To `value`	A value of the type required by the selected variable, operation or property.	Blank

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await ChangeGameSetting({"setting": "", "value": true})
```

In use. Change a control or presentation setting used by a custom options screen. Some systems apply settings when a scene or manager is rebuilt, so confirm the live result instead of assuming every property is reactive.

Gameplay: Cutscene control and timeline-specific actions

A timeline event has a start time and an action. Its event duration and the action's own timing are not always interchangeable. These five legacy pose/movement entries are offered by the Cutscene Timeline; they are not regular shared-list picker entries.

StartCutscene()

Cutscene: Start

Start a Cutscene Timeline record and wait for its session.

Option key / field	Meaning and input	Default
Cutscene `cutsceneId`	Choose a Cutscene Timeline ID.	Blank

Timing. Waits for the cutscene session.

```
await StartCutscene({"cutsceneId": ""})
```

In use. Choose Reveal the Signal from the lever's successful branch. Let the cutscene's input, HUD, skip and camera-restoration settings own the cinematic behavior. Avoid starting the same cutscene from its own first event.

EndCutscene()

Cutscene: End

End the active cutscene, optionally restoring the camera.

Option key / field	Meaning and input	Default
Restore Previous Camera `restoreCamera`	Restore the cutscene's previous camera when ending.	true

Timing. Waits for the runtime routine to return; no further completion wait is promised by this schema.

```
await EndCutscene({"restoreCamera": true})
```

In use. Use it when a branch should cut the sequence short. Keep critical puzzle state outside optional visual beats so skipping and ending early cannot leave the game half-repaired.

Registries, proxies and native nodes

These are the inherited PCA registries supplied after runtime binding. A registry finds records; a record proxy reads/changes one record; a script-call proxy routes a call to another script. None of these is automatically the rendered Node3D.

Registry	Contents
Character	Reusable character definitions; active player has the player shortcut.
Objects	Placed entities, character instances and trigger entities in rooms.
Scene	PCA room definitions, not the top-level Godot scene.
Inventory	Inventory item definitions, separate from a bag’s quantities.
GUI	Authored interfaces; use element helpers for individual controls.
Dialogue	Dialogue graph definitions.
Quest	Quest definitions and runtime progress routes.
Asset	Imported media/model/data records.
Cursor	Pointer definitions.
Verb	Interaction verb definitions.
Fonts	Font definitions.
Cutscene	Authored timeline definitions.
Variable	Saved runtime game variables; use value.
Scripts	Native GDScript records.
PCAScripts	PCA Script records.
LuaScripts	Lua records.
BasicScripts	BASIC records.
VisualScript	Visual graph records.
Project	Whole project data, usually Get("current").
Settings	Current settings data, usually Get("current").

PCA.Get(collection) creates a registry for a supported runtime collection name, including Camera, Spawn and Objective. PCA.Has only tests whether its collection-name string is nonempty; it does not validate that the collection exists. Use the registry’s Has or the record’s Exists for an actual record test.

Registry and script-call methods

Method	Contract
registry.Get(id) / get_record(id)	Obtain a record proxy. Check Has/Exists when absence is possible.
registry.Has(id)	Check that the runtime record exists.
registry.IDs()	Array[String] of IDs in that registry.
Global.Get(script_id) / Global.script_id	Obtain a script-call proxy.
Global.IDs()	Known script IDs.
call_proxy.Call(function, arguments, context)	Await a named routed function.
call_proxy.Run(context)	Await the record’s default run route.
call_proxy.Get(name, fallback)	Read included-variable data for that script ID.
call_proxy.Set(name, value)	Write included-variable data; not an ordinary script member.
call_proxy.Emit(event_name, context)	Invoke that script’s event route directly with inherited/overridden context.

```
var library: Variant = IncludeScript("utility_gd")
var result: Variant = await library.Call("add_points", [10, 5])
library.Set("last_total", result)
```

A convenient member access such as Global.utility_gd is a call proxy, not the kept-alive Node itself. Avoid reaching into another script’s private native instance as an undocumented state-sharing mechanism.

Record proxy methods: the complete authored surface

These methods are implemented on PCAScriptRecordProxy. Use a receiver of the appropriate kind: inventory/movement/speech methods need a character or suitable actor; Get/Set/Exists apply more generally. setup and underscore-prefixed binding helpers are engine plumbing, not normal authored calls.

Get()

```
Get(property_path: String, default_value: Variant = null)
```

Read a property path with a fallback. A Dictionary value is returned as data, unlike nested dot access that creates a child proxy.

Set()

```
Set(property_path: String, value: Variant)
```

Write a property path; returns whether the runtime accepted the route. Live application depends on the field.

Exists()

```
Exists()
```

Check whether the referenced runtime record exists.

ToDictionary()

```
ToDictionary()
```

Deep-copy the whole underlying record. On a child proxy this still describes the record, not only that child path.

AddInventory()

```
AddInventory(inventory_id: String, amount: int = 1)
```

Add an item amount through this character’s inventory route.

RemoveInventory()

```
RemoveInventory(inventory_id: String, amount: int = 1)
```

Remove an item amount through this character’s inventory route.

AddDestinationPath()

```
AddDestinationPath(x: Variant, y: Variant = null, z: Variant = null,
ignore_navigation_mesh_rules: bool = false)
```

Queue a Vector3/XYZ destination. A Vector3 plus Boolean second argument can specify navigation override.

ClearDestinationPath()

```
ClearDestinationPath()
```

Clear this character’s queued destinations.

WalkPath()

```
WalkPath(speed: float = 0.0)
```

Await walking the queued path.

walkpath()

```
walkpath(speed: float = 0.0)
```

Lowercase compatibility alias for WalkPath.

WalkTo()

```
WalkTo(position_value: Variant, speed: float = 0.0, ignore_navigation: bool = false)
```

Await walking to coordinates. Unlike the high-level target-object action, this method converts a position value.

Teleport()

```
Teleport(position_value: Variant)
```

Move to coordinates without navigating there.

ChangeRoom()

```
ChangeRoom(room_id: String, position_value: Variant = null, spawn_point_or_edge: String = "",
facing: Variant = null)
```

Move this character to a room via the character-room route; active-player and NPC handling can differ.

FaceCharacter()

```
FaceCharacter(target_id: String)
```

Face the selected target character.

FaceCoordinates()

```
FaceCoordinates(position_value: Variant)
```

Face a position.

Say()

```
Say(text: String, portrait_id: String = "", voice_asset_id: String = "", voice_key: String = "")
```

Await this character’s speech with optional portrait and voice data.

PlayAnimation()

```
PlayAnimation(animation_name: String)
```

Request this actor’s clip/state. Most imported clip requests return after starting playback.

StopAnimation()

```
StopAnimation()
```

Stop this actor’s animation.

SetVisible()

```
SetVisible(visible_value: bool)
```

Set runtime visibility.

SetEnabled()

```
SetEnabled(enabled_value: bool)
```

Set runtime enabled state.

Nested properties and resource ownership

```
var door: Variant = Objects.Get("archive_door")
var effect: Dictionary = door.Get("shader", {}).duplicate(true)
effect["distortionAmount"] = 0.02
door.Set("shader", effect)
```

For an immediate supported nested effect update, use ChangeObjectProperty with property="shader.distortionAmount". Native Get/Set string paths avoid PCA Script’s separate dotted-path normalizer. A whole-container write can still require a rebuild depending on the target.

Custom is a custom-only child proxy. Get("custom.some_key") and Custom.Get("some_key") are different paths through the bridge, so use the custom-property setup appropriate to the record. Arrays should be copied and written back or edited through dedicated actions.

A native node is reached separately, for example through the guarded runtime entity_nodes lookup shown in the tutorial. A material/resource obtained from a rendered mesh may be shared by other meshes; a proxy’s ToDictionary does not duplicate loaded render resources.

Designer data: reading the property reference

The following tables cover the current property catalog for the 19 record kinds available to PCA’s property system. In native code, use the named registry/proxy shown and pass the full stored path to Get or Set. This avoids confusing a property with a Godot Node member.

R means read-only through this catalog; W means writable. Live marks catalogued immediate application. Definition marks stored data/configuration without that guarantee. A property may apply only to the matching entity kind.

> Runtime writes do not automatically save the editor’s source project. For durable story facts, use game variables and restore visual state on room entry. A schema entry is not a promise that changing imported-model metadata rebuilds a loaded resource.

Designer: Character definitions

Reusable character records. player is a special record ID, not a namespace.

Native record route: `Character.Get("elias")`.

Append `.Get("property.path", fallback)` for a read or `.Set("property.path", value)` for a write. Replace the example record ID with one from your own project.

Stored path	Type / access	Purpose and limits
`id`	text R / Definition	Stable character ID used by scripts and saved games. Rename with the editor so references can be updated.
`character.displayName`	text W / Definition	Name used for dialogue and character labels. This authored Character setting is not guaranteed to update an already-created runtime label.
`character.isPlayer`	boolean W / Definition	Whether this is the selected primary player definition.
`description`	text W / Definition	Character description.
`assetId`	record W / Definition	Model asset used by this character. Record kind: assets. Saved immediately; changing a loaded model is not guaranteed to rebuild the current actor.
`enabled`	boolean W / Definition	Whether this character definition is enabled.
`visible`	boolean W / Definition	Whether the character is visible.
`opacity`	number W / Definition	Character opacity from 0 to 1.
`synonyms`	array W / Definition	Alternative parser names for this character.
`tags`	array W / Definition	Project-defined character tags.
`character`	dictionary W / Definition / Container	Movement, dialogue, player, model, and follow settings.
`transform`	dictionary W / Definition / Container	Authored position, rotation, and scale when this record is a room instance.
`verbPrepositions`	dictionary W / Definition	Objects choose whether to use this preposition in their Verb Actions tab.
`prepositionInteractions`	array W / Definition	Match a verb and second object. If no rule matches, the first object's ordinary verb action receives both objects in its script context.

Stored path	Type / access	Purpose and limits
`hotspot`	dictionary W / Definition / Container	Interaction label, enabled state, and authored verb actions.
`trigger`	dictionary W / Definition / Container	Character enter/exit/click event volume.
`pointerFeedback`	dictionary W / Definition / Container	Hover outline, bloom, tint, and cursor feedback.
`shadow`	dictionary W / Definition / Container	Character shadow settings.
`custom`	dictionary W / Definition / Container	Project-defined custom properties.
`inventory`	array R / Live	Current live inventory records for this character. Read-only live view; use give/remove inventory actions to change it.
`room`	record R / Live	Current live room ID for this character. Record kind: scenes. Read-only live view; use change room actions to move characters.
`position`	vector3 R / Live	Current live character position. Read-only live view; use move/walk statements or transform.position on a room object.
`character.role`	text W / Definition	Character role such as npc or player.
`character.gender`	text W / Definition	Normalized character gender value.
`character.genderCustom`	text W / Definition	Custom gender label when gender is custom.
`character.enabled`	boolean W / Definition	Whether character behavior is enabled.
`character.playable`	boolean W / Definition	Whether players may control this character.
`character.switchEnabled`	boolean W / Definition	Whether this playable character appears in switching controls.
`character.switchLabel`	text W / Definition	Label shown by character switching UI.
`character.switchOrder`	integer W / Definition	Ordering in character switching UI.
`character.speechColor`	color W / Definition	Dialogue text color for this character.

Stored path	Type / access	Purpose and limits
`character.defaultState`	text W / Definition	Default character animation state ID.
`character.useCustomWalkSpeed`	boolean W / Definition	Use this character's authored walk speed.
`character.walkSpeed`	number W / Definition	Character walking speed in world units per second.
`character.runSpeed`	number W / Definition	Character running speed in world units per second.
`character.turnSpeed`	number W / Definition	Character turning speed in degrees per second.
`character.follow`	dictionary W / Definition / Container	Persistent follow defaults.
`character.modelHeight`	number W / Definition	Expected character model height.
`character.modelScale`	number W / Definition	Manual model scale.
`character.modelOffset`	vector3 W / Definition	Model offset relative to the actor.
`character.modelFacingOffset`	number W / Definition	Model yaw correction in radians.
`character.modelTint`	color W / Definition	Character model tint.
`character.autoFitModel`	boolean W / Definition	Automatically fit unskinned models to modelHeight.
`character.lockRootMotion`	boolean W / Definition	Prevent imported animation root motion from moving the actor.
`character.collisionRadius`	number W / Definition	Character navigation/collision radius.
`character.perspectiveScale`	boolean W / Definition	Use near/far perspective scaling.
`character.scaleNear`	number W / Definition	Near perspective scale percentage.
`character.scaleFar`	number W / Definition	Far perspective scale percentage.
`character.personalHudGuiId`	record W / Definition	HUD GUI used by this playable character. Record kind: guis.
`character.personalInventoryGuiId`	record W / Definition	Inventory GUI used by this playable character. Record kind: guis.
`character.personalJournalGuiId`	record W / Definition	Journal GUI used by this playable character. Record kind: guis.

Stored path	Type / access	Purpose and limits
`character.personalDialogueGuiId`	record W / Definition	Dialogue GUI used by this playable character. Record kind: guis.
`character.states`	array W / Definition	Authored animation-state records.
`character.transitions`	array W / Definition	Authored animation transitions.
`character.portraits`	array W / Definition	Authored portrait definitions.
`character.defaultPortraitId`	text W / Definition	Default portrait ID.
`transform.position`	vector3 W / Live	Local position.
`transform.rotation`	vector3 W / Live	Euler rotation in editor-authored degrees.
`transform.scale`	vector3 W / Live	Local scale.
`hotspot.defaultVerb`	text W / Definition	Context mode runs this verb for this object or character, regardless of the selected verb.
`hotspot.enabled`	boolean W / Definition	Whether the hotspot accepts interactions.
`hotspot.label`	text W / Definition	Fallback interaction label after character.displayName and object.name. This is not the object's primary display name.
`hotspot.collisionOffset`	vector3 W / Definition	Interaction collision offset.
`hotspot.collisionSize`	vector3 W / Definition	Interaction collision size.
`hotspot.verbs`	dictionary W / Definition / Container	Actions keyed by authored interaction verb.
`trigger.shape`	text W / Definition	Trigger volume shape such as box, sphere, or capsule.
`trigger.size`	vector3 W / Definition	Trigger volume dimensions.
`trigger.offset`	vector3 W / Definition	Trigger volume offset.
`trigger.onEnter`	array W / Definition	Actions run when a character enters.
`trigger.onExit`	array W / Definition	Actions run when a character exits.
`trigger.onClick`	array W / Definition	Actions run on left click.

Stored path	Type / access	Purpose and limits
`trigger.onRightClick`	array W / Definition	Actions run on right click.
`trigger.onDoubleClick`	array W / Definition	Actions run on double click.
`trigger.onDoubleRightClick`	array W / Definition	Actions run on double right click.
`trigger.onHover`	array W / Definition	Actions run when hover begins.
`trigger.onHoverExit`	array W / Definition	Actions run when hover ends.
`pointerFeedback.displayNameMode`	text W / Definition	Inherit, show, or hide hover-name behavior.
`pointerFeedback.highlightOnHover`	boolean W / Definition	Enable pointer attention feedback.
`pointerFeedback.attentionMode`	text W / Definition	Outline, bloom, tint, or combined attention style.
`pointerFeedback.outlineColor`	color W / Definition	Pointer outline effect color.
`pointerFeedback.outlineThickness`	number W / Definition	Pointer outline thickness.
`pointerFeedback.bloomColor`	color W / Definition	Pointer bloom effect color.
`pointerFeedback.bloomEnergy`	number W / Definition	Pointer bloom energy.
`pointerFeedback.bloomStrength`	number W / Definition	Pointer bloom strength.
`pointerFeedback.tintColor`	color W / Definition	Pointer tint effect color.
`pointerFeedback.tintStrength`	number W / Definition	Pointer tint blend strength.
`pointerFeedback.hoverCursorId`	record W / Definition	Cursor used while hovering. Record kind: cursors.
`pointerFeedback.leftClickCursorId`	record W / Definition	Cursor used for left click. Record kind: cursors.
`pointerFeedback.rightClickCursorId`	record W / Definition	Cursor used for right click. Record kind: cursors.
`shadow.cast`	boolean W / Definition	Whether this object casts shadows.
`shadow.receive`	boolean W / Definition	Whether this object receives shadows.

Stored path	Type / access	Purpose and limits
`shadow.transparent`	boolean W / Definition	Treat material as transparent for shadow setup.
`shadow.doubleSided`	boolean W / Definition	Render both sides for shadows.
`shadow.optimization`	text W / Definition	Shadow optimization mode.
`shadow.contactEnabled`	boolean W / Definition	Enable contact shadow.
`shadow.contactSize`	number W / Definition	Contact shadow size.
`shadow.contactOpacity`	number W / Definition	Contact shadow opacity.
`character.follow.enabled`	boolean W / Definition	Whether persistent following is enabled.
`character.follow.targetId`	record W / Definition	Character to follow. Record kind: characters. This is a persistent default. Use follow statements for immediate runtime behavior.
`character.follow.distance`	number W / Definition	Desired distance behind the target.
`character.follow.speed`	number W / Definition	Maximum following speed.

Designer: Objects and character instances

Objects and character instances placed in rooms.

Native record route: `Objects.Get("archive_door")`.

Append `.Get("property.path", fallback)` for a read or `.Set("property.path", value)` for a write. Replace the example record ID with one from your own project.

Stored path	Type / access	Purpose and limits
`id`	text R / Definition	Stable room-entity ID. Rename with the editor so references can be updated.
`type`	text R / Definition	Entity kind used by PCA3D.
`name`	text W / Definition	Primary object display name. Character objects use character.displayName first. Hover precedence is character.displayName, then name, then hotspot.label, then the stable ID. Objects do not have generic displayName or displayColor properties.
`description`	text W / Definition	Object description.
`assetId`	record W / Definition	Model, sprite, or other source asset. Record kind: assets. Saved to the record; a loaded object may need to be recreated for asset changes.
`folder`	text W / Definition	Editor organization folder.
`enabled`	boolean W / Live	Whether this object can update and interact.
`visible`	boolean W / Live	Whether this object is visible.
`opacity`	number W / Definition	Object opacity from 0 to 1.
`locked`	boolean W / Definition	Whether editor manipulation is locked.
`editorOnly`	boolean W / Definition	Editor visualization metadata. The runtime does not currently use this field to suppress object spawning. Use enabled/visible or scene authoring controls for runtime behavior.
`synonyms`	array W / Definition	Alternative parser names for this object.
`tags`	array W / Definition	Project-defined object tags.

Stored path	Type / access	Purpose and limits
`transform`	dictionary W / Definition / Container	Position, rotation, and scale.
`attachmentTargetId`	text W / Definition	Character or object followed by this particle effect.
`attachmentPoint`	text W / Definition	Optional named attach point on the target. Empty follows its origin. Offset is relative to the point; animation and bone movement are followed.
`attachmentOffset`	vector3 W / Definition	Offset relative to the target point.
`verbPrepositions`	dictionary W / Definition	Objects choose whether to use this preposition in their Verb Actions tab.
`prepositionInteractions`	array W / Definition	Match a verb and second object. If no rule matches, the first object's ordinary verb action receives both objects in its script context.
`hotspot`	dictionary W / Definition / Container	Interaction label, enabled state, and verb actions.
`pointerFeedback`	dictionary W / Definition / Container	Hover outline, bloom, tint, and cursor feedback. Hover text color is global at settings.objectDisplayName.textColor, not an object property.
`trigger`	dictionary W / Definition / Container	Shape, size, and event actions.
`shadow`	dictionary W / Definition / Container	Shadow and contact-shadow settings.
`glow`	dictionary W / Definition / Container	Object glow settings.
`shader`	dictionary W / Definition / Container	Visual effect stack, including heat-haze.
`material`	dictionary W / Definition / Container	Material color, roughness, metallic, and reflection settings.
`motion`	dictionary W / Definition / Container	Automatic or imported animation motion settings.
`sprite`	dictionary W / Definition / Container	Sprite dimensions and generation settings.

Stored path	Type / access	Purpose and limits
`primitive`	dictionary W / Definition / Container	Primitive geometry settings.
`generator`	dictionary W / Definition / Container	Procedural generator settings.
`light`	dictionary W / Definition / Container	Light-specific settings.
`particle`	dictionary W / Definition / Container	Particle effect settings.
`rgbString`	dictionary W / Definition / Container	RGB light-string settings.
`custom`	dictionary W / Definition / Container	Project-defined custom properties.
`character`	dictionary W / Definition / Container	Character-only data present on this Object instance.
`transform.position`	vector3 W / Live	Local position.
`transform.rotation`	vector3 W / Live	Euler rotation in editor-authored degrees.
`transform.scale`	vector3 W / Live	Local scale.
`hotspot.defaultVerb`	text W / Definition	Context mode runs this verb for this object or character, regardless of the selected verb.
`hotspot.enabled`	boolean W / Definition	Whether the hotspot accepts interactions.
`hotspot.label`	text W / Definition	Fallback interaction label after character.displayName and object.name. This is not the object's primary display name.
`hotspot.collisionOffset`	vector3 W / Definition	Interaction collision offset.
`hotspot.collisionSize`	vector3 W / Definition	Interaction collision size.
`hotspot.verbs`	dictionary W / Definition / Container	Actions keyed by authored interaction verb.
`pointerFeedback.displayNameMode`	text W / Definition	Inherit, show, or hide hover-name behavior.

Stored path	Type / access	Purpose and limits
`pointerFeedback.highlightOnHover`	boolean W / Definition	Enable pointer attention feedback.
`pointerFeedback.attentionMode`	text W / Definition	Outline, bloom, tint, or combined attention style.
`pointerFeedback.outlineColor`	color W / Definition	Pointer outline effect color.
`pointerFeedback.outlineThickness`	number W / Definition	Pointer outline thickness.
`pointerFeedback.bloomColor`	color W / Definition	Pointer bloom effect color.
`pointerFeedback.bloomEnergy`	number W / Definition	Pointer bloom energy.
`pointerFeedback.bloomStrength`	number W / Definition	Pointer bloom strength.
`pointerFeedback.tintColor`	color W / Definition	Pointer tint effect color.
`pointerFeedback.tintStrength`	number W / Definition	Pointer tint blend strength.
`pointerFeedback.hoverCursorId`	record W / Definition	Cursor used while hovering. Record kind: cursors.
`pointerFeedback.leftClickCursorId`	record W / Definition	Cursor used for left click. Record kind: cursors.
`pointerFeedback.rightClickCursorId`	record W / Definition	Cursor used for right click. Record kind: cursors.
`trigger.shape`	text W / Definition	Trigger volume shape such as box, sphere, or capsule.
`trigger.size`	vector3 W / Definition	Trigger volume dimensions.
`trigger.offset`	vector3 W / Definition	Trigger volume offset.
`trigger.onEnter`	array W / Definition	Actions run when a character enters.
`trigger.onExit`	array W / Definition	Actions run when a character exits.
`trigger.onClick`	array W / Definition	Actions run on left click.
`trigger.onRightClick`	array W / Definition	Actions run on right click.
`trigger.onDoubleClick`	array W / Definition	Actions run on double click.

Stored path	Type / access	Purpose and limits
`trigger.onDoubleRightClick`	array W / Definition	Actions run on double right click.
`trigger.onHover`	array W / Definition	Actions run when hover begins.
`trigger.onHoverExit`	array W / Definition	Actions run when hover ends.
`shadow.cast`	boolean W / Definition	Whether this object casts shadows.
`shadow.receive`	boolean W / Definition	Whether this object receives shadows.
`shadow.transparent`	boolean W / Definition	Treat material as transparent for shadow setup.
`shadow.doubleSided`	boolean W / Definition	Render both sides for shadows.
`shadow.optimization`	text W / Definition	Shadow optimization mode.
`shadow.contactEnabled`	boolean W / Definition	Enable contact shadow.
`shadow.contactSize`	number W / Definition	Contact shadow size.
`shadow.contactOpacity`	number W / Definition	Contact shadow opacity.
`glow.enabled`	boolean W / Definition	Enable object glow.
`glow.color`	color W / Definition	Glow color.
`glow.energy`	number W / Definition	Glow energy multiplier.
`glow.useAlbedo`	boolean W / Definition	Use material albedo as glow color.
`glow.textureAssetId`	record W / Definition	Optional glow texture asset. Record kind: assets.
`glow.variable`	text W / Definition	Variable used to control glow.
`glow.startOn`	boolean W / Definition	Initial variable-controlled glow state.
`shader.preset`	text W / Definition	Use heat-haze for hot-air refraction. Explicit layers take priority over this legacy single preset.
`shader.layers`	array W / Definition	Ordered effects. Heat Haze is heat-haze; each layer has preset and intensity. Local heat surfaces use heat alone.

Stored path	Type / access	Purpose and limits
shader.intensity	number W / Definition	Strength of a single-preset effect. Zero removes heat distortion; layer intensity overrides this value.
shader.distortionAmount	number W / Definition	Heat refraction distance in screen UV units. Start at 0.02; zero removes distortion.
shader.distortionSpeed	number W / Definition	Rising heat speed. Start at 1; zero freezes movement.
material.color	color W / Definition	Base material color.
material.roughness	number W / Definition	Material roughness.
material.metallic	number W / Definition	Material metallic amount.
material.reflectionColor	color W / Definition	Reflection tint color.
material.reflectionStrength	number W / Definition	Reflection strength.
motion.enabled	boolean W / Definition	Enable automatic object motion.
motion.axis	vector3 W / Definition	Motion direction or rotation axis.
motion.speed	number W / Definition	Motion speed.
motion.space	text W / Definition	Local or global motion space.
motion.animationName	text W / Definition	Imported animation clip name.
motion.useImportedAnimation	boolean W / Definition	Drive motion with an imported clip.
motion.variable	text W / Definition	Variable used to control motion.
motion.startOn	boolean W / Definition	Initial variable-controlled motion state.
particle.imageAssetId	record W / Live	Optional image asset for particle sprites. Import a transparent PNG in Asset Designer, then select it here. Empty uses the preset artwork. Images keep their aspect ratio and are tinted by particle colors; use white colors to preserve the image. Heat Haze and lightning arcs keep their specialized materials; Lightning Ball uses the image for its halo. Record kind: assets.
particle.enabled	boolean W / Definition	Enable lightning playback. Enabling after disabling restarts the flash.

Stored path	Type / access	Purpose and limits
particle.oneShot	boolean W / Definition	Play one flash instead of repeating.
particle.ribbon	boolean W / Definition	Flat camera-facing lightning strip instead of a volumetric bolt. Both follow attachment points. This is a beam, not a trail left behind a moving object.
particle.color	text W / Definition	Lightning uses Primary Color for the outer bolt, Emission Color for its bright core, and Opacity for both. Glow depends on the renderer and environment. Motion, amount and lifetime fields belong to other particle presets.
particle.emissionColor	text W / Definition	Lightning uses Primary Color for the outer bolt, Emission Color for its bright core, and Opacity for both. Glow depends on the renderer and environment. Motion, amount and lifetime fields belong to other particle presets.
particle.opacity	number W / Definition	Particle opacity from 0 to 1.
particle.direction	vector3 W / Definition	Bolt direction relative to the emitter.
particle.boltWidth	number W / Definition	Lightning thickness in scene units. The bright core is one third of this width.
particle.boltLength	number W / Definition	Distance from the emitter to the bolt tip in scene units, along Direction. Transform scale also affects the result.
particle.flashDuration	number W / Definition	Seconds each flash stays visible. One Shot plays one flash; disable and enable Emit In Game to replay it.
particle.flashInterval	number W / Definition	Dark time in seconds between flashes. Zero produces a continuous bolt. Each new flash gets a new jagged shape.
particle.boltJaggedness	number W / Definition	Sideways displacement in scene units. Zero makes a straight beam. Start and end remain fixed.
particle.boltSegments	number W / Definition	Number of lightning segments, from 2 to 128. Fewer segments reduce rendering work.
character.displayName	text W / Definition	Name used for dialogue and character labels. This authored Character setting is not guaranteed to update an already-created runtime label.
character.role	text W / Definition	Character role such as npc or player.
character.gender	text W / Definition	Normalized character gender value.
character.genderCustom	text W / Definition	Custom gender label when gender is custom.
character.enabled	boolean W / Definition	Whether character behavior is enabled.

Stored path	Type / access	Purpose and limits
`character.playable`	boolean W / Definition	Whether players may control this character.
`character.isPlayer`	boolean W / Definition	Whether this is the selected primary player definition.
`character.switchEnabled`	boolean W / Definition	Whether this playable character appears in switching controls.
`character.switchLabel`	text W / Definition	Label shown by character switching UI.
`character.switchOrder`	integer W / Definition	Ordering in character switching UI.
`character.speechColor`	color W / Definition	Dialogue text color for this character.
`character.defaultState`	text W / Definition	Default character animation state ID.
`character.useCustomWalkSpeed`	boolean W / Definition	Use this character's authored walk speed.
`character.walkSpeed`	number W / Definition	Character walking speed in world units per second.
`character.runSpeed`	number W / Definition	Character running speed in world units per second.
`character.turnSpeed`	number W / Definition	Character turning speed in degrees per second.
`character.follow`	dictionary W / Definition / Container	Persistent follow defaults.
`character.modelHeight`	number W / Definition	Expected character model height.
`character.modelScale`	number W / Definition	Manual model scale.
`character.modelOffset`	vector3 W / Definition	Model offset relative to the actor.
`character.modelFacingOffset`	number W / Definition	Model yaw correction in radians.
`character.modelTint`	color W / Definition	Character model tint.
`character.autoFitModel`	boolean W / Definition	Automatically fit unskinned models to modelHeight.
`character.lockRootMotion`	boolean W / Definition	Prevent imported animation root motion from moving the actor.
`character.collisionRadius`	number W / Definition	Character navigation/collision radius.

Stored path	Type / access	Purpose and limits
`character.perspectiveScale`	boolean W / Definition	Use near/far perspective scaling.
`character.scaleNear`	number W / Definition	Near perspective scale percentage.
`character.scaleFar`	number W / Definition	Far perspective scale percentage.
`character.personalHudGuiId`	record W / Definition	HUD GUI used by this playable character. Record kind: guis.
`character.personalInventoryGuiId`	record W / Definition	Inventory GUI used by this playable character. Record kind: guis.
`character.personalJournalGuiId`	record W / Definition	Journal GUI used by this playable character. Record kind: guis.
`character.personalDialogueGuiId`	record W / Definition	Dialogue GUI used by this playable character. Record kind: guis.
`character.states`	array W / Definition	Authored animation-state records.
`character.transitions`	array W / Definition	Authored animation transitions.
`character.portraits`	array W / Definition	Authored portrait definitions.
`character.defaultPortraitId`	text W / Definition	Default portrait ID.
`character.follow.enabled`	boolean W / Definition	Whether persistent following is enabled.
`character.follow.targetId`	record W / Definition	Character to follow. Record kind: characters. This is a persistent default. Use follow statements for immediate runtime behavior.
`character.follow.distance`	number W / Definition	Desired distance behind the target.
`character.follow.speed`	number W / Definition	Maximum following speed.

Designer: Triggers

Trigger-type room entities.

Native record route: `Objects.Get("welcome_trigger")`.

Append `.Get("property.path", fallback)` for a read or `.Set("property.path", value)` for a write. Replace the example record ID with one from your own project.

Stored path	Type / access	Purpose and limits
`id`	text R / Definition	Stable room-entity ID. Rename with the editor so references can be updated.
`type`	text R / Definition	Trigger entity kind.
`name`	text W / Definition	Primary object display name. Character objects use character.displayName first. Hover precedence is character.displayName, then name, then hotspot.label, then the stable ID. Objects do not have generic displayName or displayColor properties.
`description`	text W / Definition	Object description.
`assetId`	record W / Definition	Model, sprite, or other source asset. Record kind: assets. Saved to the record; a loaded object may need to be recreated for asset changes.
`folder`	text W / Definition	Editor organization folder.
`enabled`	boolean W / Live	Whether this object can update and interact.
`visible`	boolean W / Live	Whether this object is visible.
`opacity`	number W / Definition	Object opacity from 0 to 1.
`locked`	boolean W / Definition	Whether editor manipulation is locked.
`editorOnly`	boolean W / Definition	Editor visualization metadata. The runtime does not currently use this field to suppress object spawning. Use enabled/visible or scene authoring controls for runtime behavior.
`synonyms`	array W / Definition	Alternative parser names for this object.
`tags`	array W / Definition	Project-defined object tags.
`transform`	dictionary W / Definition / Container	Position, rotation, and scale.

Stored path	Type / access	Purpose and limits
`attachmentTargetId`	text W / Definition	Character or object followed by this particle effect.
`attachmentPoint`	text W / Definition	Optional named attach point on the target. Empty follows its origin. Offset is relative to the point; animation and bone movement are followed.
`attachmentOffset`	vector3 W / Definition	Offset relative to the target point.
`verbPrepositions`	dictionary W / Definition	Objects choose whether to use this preposition in their Verb Actions tab.
`prepositionInteractions`	array W / Definition	Match a verb and second object. If no rule matches, the first object's ordinary verb action receives both objects in its script context.
`hotspot`	dictionary W / Definition / Container	Interaction label, enabled state, and verb actions.
`pointerFeedback`	dictionary W / Definition / Container	Hover outline, bloom, tint, and cursor feedback. Hover text color is global at settings.objectDisplayName.textColor, not an object property.
`trigger`	dictionary W / Definition / Container	Shape, size, and event actions.
`shadow`	dictionary W / Definition / Container	Shadow and contact-shadow settings.
`glow`	dictionary W / Definition / Container	Object glow settings.
`shader`	dictionary W / Definition / Container	Visual effect stack, including heat-haze.
`material`	dictionary W / Definition / Container	Material color, roughness, metallic, and reflection settings.
`motion`	dictionary W / Definition / Container	Automatic or imported animation motion settings.
`sprite`	dictionary W / Definition / Container	Sprite dimensions and generation settings.
`primitive`	dictionary W / Definition / Container	Primitive geometry settings.

Stored path	Type / access	Purpose and limits
`generator`	dictionary W / Definition / Container	Procedural generator settings.
`light`	dictionary W / Definition / Container	Light-specific settings.
`particle`	dictionary W / Definition / Container	Particle effect settings.
`rgbString`	dictionary W / Definition / Container	RGB light-string settings.
`custom`	dictionary W / Definition / Container	Project-defined custom properties.
`transform.position`	vector3 W / Live	Local position.
`transform.rotation`	vector3 W / Live	Euler rotation in editor-authored degrees.
`transform.scale`	vector3 W / Live	Local scale.
`hotspot.defaultVerb`	text W / Definition	Context mode runs this verb for this object or character, regardless of the selected verb.
`hotspot.enabled`	boolean W / Definition	Whether the hotspot accepts interactions.
`hotspot.label`	text W / Definition	Fallback interaction label after character.displayName and object.name. This is not the object's primary display name.
`hotspot.collisionOffset`	vector3 W / Definition	Interaction collision offset.
`hotspot.collisionSize`	vector3 W / Definition	Interaction collision size.
`hotspot.verbs`	dictionary W / Definition / Container	Actions keyed by authored interaction verb.
`pointerFeedback.displayNameMode`	text W / Definition	Inherit, show, or hide hover-name behavior.
`pointerFeedback.highlightOnHover`	boolean W / Definition	Enable pointer attention feedback.
`pointerFeedback.attentionMode`	text W / Definition	Outline, bloom, tint, or combined attention style.
`pointerFeedback.outlineColor`	color W / Definition	Pointer outline effect color.

Stored path	Type / access	Purpose and limits
`pointerFeedback.outlineThickness`	number W / Definition	Pointer outline thickness.
`pointerFeedback.bloomColor`	color W / Definition	Pointer bloom effect color.
`pointerFeedback.bloomEnergy`	number W / Definition	Pointer bloom energy.
`pointerFeedback.bloomStrength`	number W / Definition	Pointer bloom strength.
`pointerFeedback.tintColor`	color W / Definition	Pointer tint effect color.
`pointerFeedback.tintStrength`	number W / Definition	Pointer tint blend strength.
`pointerFeedback.hoverCursorId`	record W / Definition	Cursor used while hovering. Record kind: cursors.
`pointerFeedback.leftClickCursorId`	record W / Definition	Cursor used for left click. Record kind: cursors.
`pointerFeedback.rightClickCursorId`	record W / Definition	Cursor used for right click. Record kind: cursors.
`trigger.shape`	text W / Definition	Trigger volume shape such as box, sphere, or capsule.
`trigger.size`	vector3 W / Definition	Trigger volume dimensions.
`trigger.offset`	vector3 W / Definition	Trigger volume offset.
`trigger.onEnter`	array W / Definition	Actions run when a character enters.
`trigger.onExit`	array W / Definition	Actions run when a character exits.
`trigger.onClick`	array W / Definition	Actions run on left click.
`trigger.onRightClick`	array W / Definition	Actions run on right click.
`trigger.onDoubleClick`	array W / Definition	Actions run on double click.
`trigger.onDoubleRightClick`	array W / Definition	Actions run on double right click.
`trigger.onHover`	array W / Definition	Actions run when hover begins.
`trigger.onHoverExit`	array W / Definition	Actions run when hover ends.

Stored path	Type / access	Purpose and limits
shadow.cast	boolean W / Definition	Whether this object casts shadows.
shadow.receive	boolean W / Definition	Whether this object receives shadows.
shadow.transparent	boolean W / Definition	Treat material as transparent for shadow setup.
shadow.doubleSided	boolean W / Definition	Render both sides for shadows.
shadow.optimization	text W / Definition	Shadow optimization mode.
shadow.contactEnabled	boolean W / Definition	Enable contact shadow.
shadow.contactSize	number W / Definition	Contact shadow size.
shadow.contactOpacity	number W / Definition	Contact shadow opacity.
glow.enabled	boolean W / Definition	Enable object glow.
glow.color	color W / Definition	Glow color.
glow.energy	number W / Definition	Glow energy multiplier.
glow.useAlbedo	boolean W / Definition	Use material albedo as glow color.
glow.textureAssetId	record W / Definition	Optional glow texture asset. Record kind: assets.
glow.variable	text W / Definition	Variable used to control glow.
glow.startOn	boolean W / Definition	Initial variable-controlled glow state.
shader.preset	text W / Definition	Use heat-haze for hot-air refraction. Explicit layers take priority over this legacy single preset.
shader.layers	array W / Definition	Ordered effects. Heat Haze is heat-haze; each layer has preset and intensity. Local heat surfaces use heat alone.
shader.intensity	number W / Definition	Strength of a single-preset effect. Zero removes heat distortion; layer intensity overrides this value.
shader.distortionAmount	number W / Definition	Heat refraction distance in screen UV units. Start at 0.02; zero removes distortion.
shader.distortionSpeed	number W / Definition	Rising heat speed. Start at 1; zero freezes movement.

Stored path	Type / access	Purpose and limits
material.color	color W / Definition	Base material color.
material.roughness	number W / Definition	Material roughness.
material.metallic	number W / Definition	Material metallic amount.
material.reflectionColor	color W / Definition	Reflection tint color.
material.reflectionStrength	number W / Definition	Reflection strength.
motion.enabled	boolean W / Definition	Enable automatic object motion.
motion.axis	vector3 W / Definition	Motion direction or rotation axis.
motion.speed	number W / Definition	Motion speed.
motion.space	text W / Definition	Local or global motion space.
motion.animationName	text W / Definition	Imported animation clip name.
motion.useImportedAnimation	boolean W / Definition	Drive motion with an imported clip.
motion.variable	text W / Definition	Variable used to control motion.
motion.startOn	boolean W / Definition	Initial variable-controlled motion state.
particle.imageAssetId	record W / Live	Optional image asset for particle sprites. Import a transparent PNG in Asset Designer, then select it here. Empty uses the preset artwork. Images keep their aspect ratio and are tinted by particle colors; use white colors to preserve the image. Heat Haze and lightning arcs keep their specialized materials; Lightning Ball uses the image for its halo. Record kind: assets.
particle.enabled	boolean W / Definition	Enable lightning playback. Enabling after disabling restarts the flash.
particle.oneShot	boolean W / Definition	Play one flash instead of repeating.
particle.ribbon	boolean W / Definition	Flat camera-facing lightning strip instead of a volumetric bolt. Both follow attachment points. This is a beam, not a trail left behind a moving object.

Stored path	Type / access	Purpose and limits
`particle.color`	text W / Definition	Lightning uses Primary Color for the outer bolt, Emission Color for its bright core, and Opacity for both. Glow depends on the renderer and environment. Motion, amount and lifetime fields belong to other particle presets.
`particle.emissionColor`	text W / Definition	Lightning uses Primary Color for the outer bolt, Emission Color for its bright core, and Opacity for both. Glow depends on the renderer and environment. Motion, amount and lifetime fields belong to other particle presets.
`particle.opacity`	number W / Definition	Particle opacity from 0 to 1.
`particle.direction`	vector3 W / Definition	Bolt direction relative to the emitter.
`particle.boltWidth`	number W / Definition	Lightning thickness in scene units. The bright core is one third of this width.
`particle.boltLength`	number W / Definition	Distance from the emitter to the bolt tip in scene units, along Direction. Transform scale also affects the result.
`particle.flashDuration`	number W / Definition	Seconds each flash stays visible. One Shot plays one flash; disable and enable Emit In Game to replay it.
`particle.flashInterval`	number W / Definition	Dark time in seconds between flashes. Zero produces a continuous bolt. Each new flash gets a new jagged shape.
`particle.boltJaggedness`	number W / Definition	Sideways displacement in scene units. Zero makes a straight beam. Start and end remain fixed.
`particle.boltSegments`	number W / Definition	Number of lightning segments, from 2 to 128. Fewer segments reduce rendering work.

Designer: Rooms

Rooms and scenes.

Native record route: `Scene.Get("lab")`.

Append `.Get("property.path", fallback)` for a read or `.Set("property.path", value)` for a write. Replace the example record ID with one from your own project.

Stored path	Type / access	Purpose and limits
`id`	text R / Definition	Stable room ID. Rename with the editor so references can be updated.
`name`	text W / Definition	Display name for this room.
`description`	text W / Definition	Room description.
`folder`	text W / Definition	Editor organization folder.
`defaultCameraId`	record W / Definition	Camera selected when the room loads. Record kind: cameras.
`hidePlayerCharacter`	boolean W / Definition	Hide the player while this room is active.
`hideAllGuisOnLoad`	boolean W / Definition	Hide authored GUIs when entering this room.
`objectDisplayNameMode`	text W / Definition	Room override for object hover names.
`environment`	dictionary W / Definition / Container	Brightness, fog, glow, skybox, outline, and post-processing.
`transition`	dictionary W / Definition / Container	Transition used when entering this room.
`viewport`	dictionary W / Definition / Container	Room rendering and GUI design size overrides.
`cameraBounds`	dictionary W / Definition / Container	Room camera movement limits.
`screenOverlay`	dictionary W / Definition / Container	Room screen-overlay settings.
`navigation`	dictionary W / Definition / Container	Walkable areas and navigation-agent settings.

Stored path	Type / access	Purpose and limits
dynamicCamera	dictionary W / Definition / Container	Room dynamic-camera settings.
entities	array R / Definition	Objects placed in the room. Edit individual objects through the objects namespace.
cameras	array R / Definition	Cameras nested in this room. Edit individual cameras through the cameras namespace.
spawns	array R / Definition	Spawn points nested in this room. Edit individual spawns through the spawns namespace.
regions	array W / Definition	Navigation and gameplay regions.
custom	dictionary W / Definition / Container	Project-defined room properties.
environment.brightness	number W / Definition	Room environment brightness.
environment.fogEnabled	boolean W / Definition	Enable room fog.
environment.fogMode	text W / Definition	Depth, height, or volumetric fog mode.
environment.fogColor	color W / Definition	Fog color.
environment.fogDensity	number W / Definition	Fog density.
environment.fogDepthBegin	number W / Definition	Depth fog start distance.
environment.fogDepthEnd	number W / Definition	Depth fog end distance.
environment.fogHeight	number W / Definition	Height fog reference height.
environment.fogHeightDensity	number W / Definition	Height fog density.
environment.glowEnabled	boolean W / Definition	Enable environment glow.
environment.glowIntensity	number W / Definition	Environment glow intensity.
environment.glowStrength	number W / Definition	Environment glow strength.
environment.volumetricFogEnabled	boolean W / Definition	Enable volumetric fog in Forward+.
environment.volumetricFogDensity	number W / Definition	Volumetric fog density from 0 to 1.

Stored path	Type / access	Purpose and limits
`environment.volumetricFogLength`	number W / Definition	Volumetric fog drawing distance.
`environment.reflectionProbePosition`	vector3 W / Definition	Room probe center in world coordinates.
`environment.shadowQuality`	text W / Definition	Room shadow-quality preset.
`environment.postProcessOverrideMode`	text W / Definition	Inherit, custom, or disabled post-processing.
`environment.postProcess`	dictionary W / Definition / Container	Room post-processing effect settings.
`environment.sceneOutline`	dictionary W / Definition / Container	Full-scene outline settings.
`environment.sceneOutline.edgesOnly`	boolean W / Definition	Outlines visible depth edges and keeps the original surface shading.
`environment.skybox`	dictionary W / Definition / Container	Room skybox settings.
`transition.type`	text W / Definition	Cut, fade, wipe, dissolve, or another authored transition.
`transition.duration`	number W / Definition	Transition duration in seconds.
`transition.color`	color W / Definition	Transition color.
`transition.direction`	text W / Definition	Transition direction.
`transition.origin`	vector2 W / Definition	Normalized transition origin.
`transition.intensity`	number W / Definition	Transition intensity.
`transition.randomPool`	array W / Definition	Transition types available to random mode.
`transition.avoidImmediateRepeat`	boolean W / Definition	Avoid choosing the same random transition twice.
`viewport.overrideMode`	text W / Definition	Inherit or override viewport sizing.
`viewport.width`	integer W / Definition	Room viewport width.
`viewport.height`	integer W / Definition	Room viewport height.

Stored path	Type / access	Purpose and limits
`viewport.guiSizeMode`	text W / Definition	GUI design-size behavior.
`viewport.overlaySizeMode`	text W / Definition	Screen-overlay size behavior.
`cameraBounds.enabled`	boolean W / Definition	Enable camera movement bounds.
`cameraBounds.minX`	number W / Definition	Minimum camera X position.
`cameraBounds.maxX`	number W / Definition	Maximum camera X position.
`cameraBounds.minZ`	number W / Definition	Minimum camera Z position.
`cameraBounds.maxZ`	number W / Definition	Maximum camera Z position.
`screenOverlay.enabled`	boolean W / Definition	Enable the screen overlay.
`screenOverlay.assetId`	record W / Definition	Image/video asset used by the overlay. Record kind: assets.
`screenOverlay.overrideMode`	text W / Definition	Room/global overlay override mode.
`screenOverlay.shaderMode`	text W / Definition	Overlay shader mode.
`screenOverlay.parallaxEnabled`	boolean W / Definition	Enable cursor/camera parallax.
`screenOverlay.parallaxStrength`	number W / Definition	Overlay parallax strength.
`screenOverlay.parallaxSmoothing`	number W / Definition	Overlay parallax smoothing.
`screenOverlay.transitionAffectsOverlay`	boolean W / Definition	Apply room transitions to the overlay.
`navigation.walkableY`	number W / Definition	Default walkable Y plane.
`navigation.cellSize`	number W / Definition	Navigation map cell size.
`navigation.cellHeight`	number W / Definition	Navigation map cell height.
`navigation.agentRadius`	number W / Definition	Default navigation-agent radius.
`navigation.type`	text W / Definition	Plane or polygon navigation area type.

Stored path	Type / access	Purpose and limits
navigation.bounds	dictionary W / Definition / Container	Primary navigation-area bounds.
navigation.polygon	array W / Definition	Primary navigation polygon points.
navigation.areas	array W / Definition	Walkable navigation-area records.
navigation.blocked	array W / Definition	Navigation blocker records.
environment.postProcess.preset	text W / Definition	Use heat-haze for hot-air refraction. Explicit layers take priority over this legacy single preset.
environment.postProcess.layers	array W / Definition	Ordered effects. Heat Haze is heat-haze; each layer has preset and intensity. Local heat surfaces use heat alone.
environment.postProcess.intensity	number W / Definition	Strength of a single-preset effect. Zero removes heat distortion; layer intensity overrides this value.
environment.postProcess.distortionAmount	number W / Definition	Heat refraction distance in screen UV units. Start at 0.02; zero removes distortion.
environment.postProcess.distortionSpeed	number W / Definition	Rising heat speed. Start at 1; zero freezes movement.

Designer: Inventory definitions

Inventory item definitions.

Native record route: `Inventory.Get("key")`.

Append `.Get("property.path", fallback)` for a read or `.Set("property.path", value)` for a write. Replace the example record ID with one from your own project.

Stored path	Type / access	Purpose and limits
`id`	text R / Definition	Stable inventory item ID. Rename with the editor so references can be updated.
`name`	text W / Definition	Inventory item display name.
`description`	text W / Definition	Inventory item description.
`category`	text W / Definition	Inventory category.
`enabled`	boolean W / Definition	Whether this inventory item is enabled.
`iconAssetId`	record W / Definition	Inventory icon asset. Record kind: assets.
`icon`	text W / Definition	Legacy/direct icon path.
`iconAnimated`	boolean W / Definition	Use animated inventory icon frames.
`iconFps`	number W / Definition	Animated icon frames per second.
`iconLoop`	boolean W / Definition	Loop animated icon frames.
`stackable`	boolean W / Definition	Whether item amounts stack.
`maxStack`	integer W / Definition	Maximum amount in one stack.
`consumeOnUse`	boolean W / Definition	Remove the item after a successful use.
`value`	number W / Definition	Game-defined item value.
`weight`	number W / Definition	Game-defined item weight.
`rarity`	text W / Definition	Game-defined rarity label.
`sortOrder`	integer W / Definition	Inventory sorting order.

Stored path	Type / access	Purpose and limits
`tags`	array W / Definition	Item tags.
`types`	array W / Definition	Item type/category tags.
`canBeCarriedBy`	array W / Definition	Character IDs allowed to carry this item.
`acquireSceneRemoval`	dictionary W / Definition / Container	Optional scene object removed when acquired.
`verbPrepositions`	dictionary W / Definition	Objects choose whether to use this preposition in their Verb Actions tab.
`prepositionInteractions`	array W / Definition	Match a verb and second object. If no rule matches, the first object's ordinary verb action receives both objects in its script context.
`interactions`	dictionary W / Definition / Container	Inventory interaction actions keyed by verb.
`inventoryInteractions`	array W / Definition	Rules for using this item on inventory/world targets.
`custom`	dictionary W / Definition / Container	Project-defined item properties.
`acquireSceneRemoval.enabled`	boolean W / Definition	Remove a scene entity when this item is acquired.
`acquireSceneRemoval.sceneId`	record W / Definition	Room containing the entity to remove. Record kind: scenes.
`acquireSceneRemoval.entityId`	record W / Definition	Object removed from the room. Record kind: entities.

Designer: GUIs

Authored GUI layouts.

Native record route: `GUI.Get("hud")`.

Append `.Get("property.path", fallback)` for a read or `.Set("property.path", value)` for a write. Replace the example record ID with one from your own project.

Stored path	Type / access	Purpose and limits
`id`	text R / Definition	Stable GUI ID.
`name`	text W / Definition	GUI display/editor name.
`visible`	boolean W / Definition	Authored initial GUI visibility. Use show/hide GUI statements for immediate runtime visibility.
`modal`	boolean W / Definition	Whether this GUI blocks gameplay input.
`zIndex`	integer W / Definition	GUI draw ordering.
`activationMode`	text W / Definition	Manual, hover, click, or another activation mode.
`clickActivationAtMouseP osition`	boolean W / Definition	Place click-activated GUI at the mouse.
`clickActivationStopsPla yer`	boolean W / Definition	Stop player movement when click-activated.
`hideWhenMouseLeaves`	boolean W / Definition	Hide hover GUI when the pointer leaves.
`hoverDelaySeconds`	number W / Definition	Delay before hover activation in seconds.
`openTransition`	dictionary W / Definition / Container	Transition used when this GUI opens.
`closeTransition`	dictionary W / Definition / Container	Transition used when this GUI closes.
`windowFrame`	dictionary W / Definition / Container	Window, title bar, and close-button appearance.
`elements`	array R / Definition	GUI element records. Edit individual elements in the GUI editor.
`custom`	dictionary W / Definition / Container	Project-defined GUI properties.
`openTransition.type`	text W / Definition	Cut, fade, wipe, dissolve, or another authored transition.

Stored path	Type / access	Purpose and limits
openTransition.duration	number W / Definition	Transition duration in seconds.
openTransition.color	color W / Definition	Transition color.
openTransition.direction	text W / Definition	Transition direction.
openTransition.origin	vector2 W / Definition	Normalized transition origin.
openTransition.intensity	number W / Definition	Transition intensity.
openTransition.randomPool	array W / Definition	Transition types available to random mode.
openTransition.avoidImmediateRepeat	boolean W / Definition	Avoid choosing the same random transition twice.
closeTransition.type	text W / Definition	Cut, fade, wipe, dissolve, or another authored transition.
closeTransition.duration	number W / Definition	Transition duration in seconds.
closeTransition.color	color W / Definition	Transition color.
closeTransition.direction	text W / Definition	Transition direction.
closeTransition.origin	vector2 W / Definition	Normalized transition origin.
closeTransition.intensity	number W / Definition	Transition intensity.
closeTransition.randomPool	array W / Definition	Transition types available to random mode.
closeTransition.avoidImmediateRepeat	boolean W / Definition	Avoid choosing the same random transition twice.
windowFrame.enabled	boolean W / Definition	Enable authored window framing.
windowFrame.rootWindow	boolean W / Definition	Treat this GUI as a root window.
windowFrame.x	number W / Definition	Window X position.
windowFrame.y	number W / Definition	Window Y position.
windowFrame.width	number W / Definition	Window width.
windowFrame.height	number W / Definition	Window height.
windowFrame.opacity	number W / Definition	Window opacity.

Stored path	Type / access	Purpose and limits
windowFrame.windowBackgroundEnabled	boolean W / Definition	Enable window background.
windowFrame.windowBackground	color W / Definition	Window background color.
windowFrame.windowBorderEnabled	boolean W / Definition	Enable window border.
windowFrame.windowBorderColor	color W / Definition	Window border color.
windowFrame.windowBorderWidth	number W / Definition	Window border width.
windowFrame.windowBorderRadius	number W / Definition	Window border radius.
windowFrame.titleBarEnabled	boolean W / Definition	Enable the title bar.
windowFrame.titleBarDraggable	boolean W / Definition	Allow dragging by the title bar.
windowFrame.titleBarHeight	number W / Definition	Title-bar height.
windowFrame.titleBarBackground	color W / Definition	Title-bar background color.
windowFrame.titleText	text W / Definition	Window title text.
windowFrame.titleTextColor	color W / Definition	Window title text color.
windowFrame.titleFontId	record W / Definition	Window title font. Record kind: fonts.
windowFrame.titleFontSize	integer W / Definition	Window title font size.
windowFrame.closeButtonEnabled	boolean W / Definition	Show a close button.
windowFrame.closeButtonText	text W / Definition	Close-button text.
windowFrame.closeButtonTextColor	color W / Definition	Close-button text color.
windowFrame.closeButtonBackground	color W / Definition	Close-button background color.

Designer: Dialogue graphs

Dialogue graph records.

Native record route: `Dialogue.Get("elias_intro")`.

Append `.Get("property.path", fallback)` for a read or `.Set("property.path", value)` for a write. Replace the example record ID with one from your own project.

Stored path	Type / access	Purpose and limits
`id`	text R / Definition	Stable dialogue ID.
`name`	text W / Definition	Dialogue display/editor name.
`start`	text W / Definition	Dialogue start-node ID.
`nodes`	dictionary R / Definition / Container	Dialogue graph nodes keyed by ID. Edit dialogue graph structure in the Dialogue Editor.
`custom`	dictionary W / Definition / Container	Project-defined dialogue properties.

Designer: Quests

Quest definitions.

Native record route: `Quest.Get("signal_quest")`.

Append `.Get("property.path", fallback)` for a read or `.Set("property.path", value)` for a write. Replace the example record ID with one from your own project.

Stored path	Type / access	Purpose and limits
`id`	text R / Definition	Stable quest ID.
`title`	text W / Definition	Quest title.
`titleKey`	text W / Definition	Translation key for the title.
`description`	text W / Definition	Quest description.
`descriptionKey`	text W / Definition	Translation key for the description.
`category`	text W / Definition	Quest category.
`hidden`	boolean W / Definition	Hide this quest from normal journal views.
`repeatable`	boolean W / Definition	Allow this quest to repeat.
`autoTrack`	boolean W / Definition	Automatically track this quest.
`folder`	text W / Definition	Editor organization folder.
`stages`	array R / Definition	Quest stage definitions. Edit nested objectives through the objective namespace or Quest Editor.
`completionActions`	array W / Definition	Actions run when the quest completes.
`failureActions`	array W / Definition	Actions run when the quest fails.
`custom`	dictionary W / Definition / Container	Project-defined quest properties.

Designer: Objectives

Objectives nested in quest stages.

Native record route: `PCA.Get("Objective").Get("find_key")`.

Append `.Get("property.path", fallback)` for a read or `.Set("property.path", value)` for a write. Replace the example record ID with one from your own project.

Stored path	Type / access	Purpose and limits
`id`	text R / Definition	Stable objective ID within its quest stage.
`text`	text W / Definition	Text displayed for the objective.
`textKey`	text W / Definition	Translation key for objective text.
`optional`	boolean W / Definition	Whether this objective is optional.
`hidden`	boolean W / Definition	Hide this objective from normal journal views.
`custom`	dictionary W / Definition / Container	Project-defined objective properties.

Designer: Assets

Model, image, sprite sheet, image animation, audio, music, video, font, and data assets.

Native record route: `Asset.Get("chime")`.

Append `.Get("property.path", fallback)` for a read or `.Set("property.path", value)` for a write. Replace the example record ID with one from your own project.

Stored path	Type / access	Purpose and limits
`id`	text R / Definition	Stable asset ID.
`name`	text W / Definition	Asset display/editor name.
`kind`	text R / Definition	Model, image, sprite sheet, image animation, audio, video, font, data, or other.
`format`	text W / Definition	Asset file format.
`mainFile`	text W / Definition	Primary asset filename. Changing this does not import or reload a file; use the Asset Designer.
`folder`	text W / Definition	Asset database folder.
`importedAt`	text R / Definition	Import timestamp.
`files`	array R / Definition	Files belonging to this asset. Managed by the Asset Designer.
`metadata`	dictionary W / Definition / Container	PCA3D/Godot import metadata.
`custom`	dictionary W / Definition / Container	Project-defined asset properties.
`metadata.godotPath`	text R / Definition	Resolved res:// asset path. Managed by asset import.
`metadata.sourceFormat`	text R / Definition	Original/imported source format.
`metadata.animations`	array R / Definition	Discovered animation clip metadata. Refreshed by model import and clip discovery.

Designer: Cameras

Cameras nested in rooms.

Native record route: `PCA.Get("Camera").Get("wide")`.

Append `.Get("property.path", fallback)` for a read or `.Set("property.path", value)` for a write. Replace the example record ID with one from your own project.

Stored path	Type / access	Purpose and limits
`id`	text R / Definition	Stable camera ID within its room.
`name`	text W / Definition	Camera display/editor name.
`type`	text W / Definition	Perspective or orthographic camera type.
`position`	vector3 W / Definition	Authored camera position. This is not live-applied to the active runtime camera.
`target`	vector3 W / Definition	Authored camera look-at target. This is not live-applied to the active runtime camera.
`fov`	number W / Definition	Perspective vertical field of view.
`near`	number W / Definition	Near clipping distance.
`far`	number W / Definition	Far clipping distance.
`size`	number W / Definition	Orthographic camera size.
`enabled`	boolean W / Definition	Whether this camera record is enabled.
`visible`	boolean W / Definition	Editor visibility.
`locked`	boolean W / Definition	Whether editor manipulation is locked.
`folder`	text W / Definition	Editor organization folder.
`depthOfField`	dictionary W / Definition / Container	Camera depth-of-field settings.
`dynamicFollow`	dictionary W / Definition / Container	Dynamic-camera follow settings.

Stored path	Type / access	Purpose and limits
`custom`	dictionary W / Definition / Container	Project-defined camera properties.
`depthOfField.enabled`	boolean W / Definition	Enable camera depth of field.
`depthOfField.amount`	number W / Definition	Depth-of-field blur amount.
`depthOfField.nearEnabled`	boolean W / Definition	Enable near blur.
`depthOfField.nearDistance`	number W / Definition	Near blur distance.
`depthOfField.nearTransition`	number W / Definition	Near blur transition distance.
`depthOfField.farEnabled`	boolean W / Definition	Enable far blur.
`depthOfField.farDistance`	number W / Definition	Far blur distance.
`depthOfField.farTransition`	number W / Definition	Far blur transition distance.
`dynamicFollow.enabled`	boolean W / Definition	Enable dynamic camera follow.
`dynamicFollow.overrideMode`	text W / Definition	Inherit or override room dynamic-camera behavior.
`dynamicFollow.mode`	text W / Definition	Dynamic follow mode.
`dynamicFollow.axes`	text W / Definition	Axes followed by the camera.
`dynamicFollow.smoothing`	number W / Definition	Follow smoothing.
`dynamicFollow.maxSpeed`	number W / Definition	Maximum follow speed.
`dynamicFollow.deadZone`	vector2 W / Definition	Follow dead-zone size.
`dynamicFollow.lookAhead`	number W / Definition	Movement look-ahead amount.
`dynamicFollow.clampToRoomBounds`	boolean W / Definition	Clamp camera to room bounds.
`dynamicFollow.boundsMargin`	number W / Definition	Room-bounds margin.

Designer: Spawn points

Spawn points nested in rooms.

Native record route: `PCA.Get("Spawn").Get("start")`.

Append `.Get("property.path", fallback)` for a read or `.Set("property.path", value)` for a write. Replace the example record ID with one from your own project.

Stored path	Type / access	Purpose and limits
`id`	text R / Definition	Stable spawn ID within its room.
`name`	text W / Definition	Spawn display/editor name.
`position`	vector3 W / Definition	Spawn position.
`facing`	number W / Definition	Spawn facing/yaw.
`enabled`	boolean W / Definition	Whether this spawn is enabled.
`visible`	boolean W / Definition	Editor visibility.
`locked`	boolean W / Definition	Whether editor manipulation is locked.
`folder`	text W / Definition	Editor organization folder.
`custom`	dictionary W / Definition / Container	Project-defined spawn properties.

Designer: Cursors

Cursor definitions.

Native record route: `Cursor.Get("walk")`.

Append `.Get("property.path", fallback)` for a read or `.Set("property.path", value)` for a write. Replace the example record ID with one from your own project.

Stored path	Type / access	Purpose and limits
`id`	text R / Definition	Stable cursor ID.
`name`	text W / Definition	Cursor display/editor name.
`imageAssetId`	record W / Definition	Cursor image asset. Record kind: assets.
`symbol`	text W / Definition	Fallback text/system cursor symbol.
`hotspotX`	integer W / Definition	Cursor hotspot X coordinate.
`hotspotY`	integer W / Definition	Cursor hotspot Y coordinate.
`color`	color W / Definition	Cursor tint color.
`verb`	text W / Definition	Legacy verb binding.
`enabled`	boolean W / Definition	Whether this cursor is enabled.
`custom`	dictionary W / Definition / Container	Project-defined cursor properties.

Designer: Verbs

Interaction verbs.

Native record route: `Verb.Get("look")`.

Append `.Get("property.path", fallback)` for a read or `.Set("property.path", value)` for a write. Replace the example record ID with one from your own project.

Stored path	Type / access	Purpose and limits
`id`	text R / Definition	Stable lowercase verb ID.
`name`	text W / Definition	Verb displayed to players.
`cursorId`	record W / Definition	Cursor used by this verb. Record kind: cursors.
`synonyms`	array W / Definition	Parser synonyms for this verb.
`allowPreposition`	boolean W / Definition	Objects choose whether to use this preposition in their Verb Actions tab.
`preposition`	text W / Definition	Preposition
`enabled`	boolean W / Definition	Whether this verb is enabled.
`folder`	text W / Definition	Editor organization folder.
`custom`	dictionary W / Definition / Container	Project-defined verb properties.

Designer: Fonts

Font definitions.

Native record route: `Fonts.Get("body")`.

Append `.Get("property.path", fallback)` for a read or `.Set("property.path", value)` for a write. Replace the example record ID with one from your own project.

Stored path	Type / access	Purpose and limits
`id`	text R / Definition	Stable font ID.
`name`	text W / Definition	Font display/editor name.
`family`	text W / Definition	Font family label.
`assetId`	record W / Definition	Font asset resource. Record kind: assets.
`style`	text W / Definition	Font style.
`weight`	integer W / Definition	Font weight.
`fallback`	text W / Definition	Comma-separated fallback font-family names.
`enabled`	boolean W / Definition	Whether this font is enabled.
`custom`	dictionary W / Definition / Container	Project-defined font properties.

Designer: PCA Script records

PCA Script source records.

Native record route: `PCAScripts.Get("utility_pca")`.

Append `.Get("property.path", fallback)` for a read or `.Set("property.path", value)` for a write. Replace the example record ID with one from your own project.

Stored path	Type / access	Purpose and limits
`id`	text R / Definition	Stable PCA Script ID.
`name`	text W / Definition	PCA Script display/editor name.
`description`	text W / Definition	Script description.
`folder`	text W / Definition	Editor organization folder.
`languageVersion`	text R / Definition	PCA Script language version.
`source`	text W / Live	PCA Script source text. Changing source invalidates the compiled script cache.
`enabled`	boolean W / Definition	Whether this script is enabled.
`breakpoints`	array W / Definition	Authored breakpoint line numbers.
`events`	array W / Definition	Stored event metadata.
`custom`	dictionary W / Definition / Container	Project-defined script properties.

Designer: Lua script records

Lua 5.4 source records.

Native record route: `LuaScripts.Get("math_lua")`.

Append `.Get("property.path", fallback)` for a read or `.Set("property.path", value)` for a write. Replace the example record ID with one from your own project.

Stored path	Type / access	Purpose and limits
`id`	text R / Definition	Stable Lua Script ID.
`name`	text W / Definition	Lua Script display/editor name.
`description`	text W / Definition	Lua Script description.
`folder`	text W / Definition	Editor organization folder.
`languageVersion`	text R / Definition	Lua language version.
`source`	text W / Definition	Lua 5.4 source text. Edit source in the Lua Designer; changes invalidate the Lua runtime cache.
`global`	boolean W / Definition	Whether one persistent instance listens globally.
`events`	array W / Definition	Declared Lua-script event bindings.
`enabled`	boolean W / Definition	Whether this Lua Script is enabled.
`custom`	dictionary W / Definition / Container	Project-defined script properties.

Designer: BASIC script records

PCA3D BASIC 1.0 source records.

Native record route: `BasicScripts.Get("math_basic")`.

Append `.Get("property.path", fallback)` for a read or `.Set("property.path", value)` for a write. Replace the example record ID with one from your own project.

Stored path	Type / access	Purpose and limits
`id`	text R / Definition	Stable BASIC Script ID.
`name`	text W / Definition	BASIC Script display/editor name.
`description`	text W / Definition	BASIC Script description.
`folder`	text W / Definition	Editor organization folder.
`languageVersion`	text R / Definition	PCA3D BASIC language version.
`source`	text W / Definition	PCA3D BASIC 1.0 source text. Edit source in the BASIC Designer; changes invalidate the BASIC runtime cache.
`global`	boolean W / Definition	Whether this BASIC script listens globally.
`events`	array W / Definition	Declared BASIC-script event bindings.
`enabled`	boolean W / Definition	Whether this BASIC Script is enabled.
`custom`	dictionary W / Definition / Container	Project-defined script properties.

Designer: Visual script records

Visual Script graph records.

Native record route: `VisualScript.Get("reward_graph")`.

Append `.Get("property.path", fallback)` for a read or `.Set("property.path", value)` for a write. Replace the example record ID with one from your own project.

Stored path	Type / access	Purpose and limits
`id`	text R / Definition	Stable Visual Script ID.
`name`	text W / Definition	Visual Script display/editor name.
`nodes`	array R / Definition	Visual Script node records. Edit graph structure in the Visual Script Editor.
`edges`	array R / Definition	Visual Script graph connections. Edit graph structure in the Visual Script Editor.
`enabled`	boolean W / Definition	Whether this Visual Script is enabled.
`custom`	dictionary W / Definition / Container	Project-defined script properties.

Variables, native scripts, project data and cutscenes

Record route	Useful data / operations
Variable.Get(id)	value reads/writes runtime game state; definitions also provide name/type/start value in project data. Do not confuse a member variable with this store.
Scripts.Get(id)	Native record fields include id, name, description, folder, source, global, events and exposedFunctions. Use the Script Designer for routine source/definition edits.
Cutscene.Get(id)	Read its actual definition with ToDictionary. Use StartCutscene/EndCutscene for playback; editing timeline arrays does not imply immediate live reconstruction.
Project.Get("current")	Project data, with property paths such as meta.title and settings.musicVolume.
Settings.Get("current")	Settings below the current settings dictionary; use GetSetting/SetSetting for a convenient wrapper.

The raw record can contain project-specific dictionaries, arrays and custom data beyond a fixed catalog. ToDictionary/Get gives you the actual data. A newly invented key has no engine behavior until code uses it.

> Native GDScript supports Godot APIs, but not every editor operation has a runtime helper. Import assets, bake/edit navigation, author complex graphs, manage translations and build a game through their tools unless you deliberately extend the editor/source project.

Character-property action keys

These are exact keys offered by the shared property picker in this build. Use them with the matching native helper/action and the expected value type. A type-specific widget/entity may not implement every generic field.

Key / path	Usage
`visible`	Boolean on/off state. Visibility and interaction enablement are separate.
`position`	Match the target’s vector type. GUI values usually use Vector2; 3D object transforms use Vector3.
`rotation_degrees`	Read the existing value to confirm its type, then use the matching helper/action. Type-specific controls can expose extra behavior.
`scale`	Match the target’s vector type. GUI values usually use Vector2; 3D object transforms use Vector3.
`walk_speed`	Read the existing value to confirm its type, then use the matching helper/action. Type-specific controls can expose extra behavior.
`character.displayName`	Read the existing value to confirm its type, then use the matching helper/action. Type-specific controls can expose extra behavior.
`character.enabled`	Boolean on/off state. Visibility and interaction enablement are separate.
`character.playable`	Read the existing value to confirm its type, then use the matching helper/action. Type-specific controls can expose extra behavior.
`character.switchEnabled`	Boolean on/off state. Visibility and interaction enablement are separate.

Game setting keys

These are exact keys offered by the shared property picker in this build. Use them with the matching native helper/action and the expected value type. A type-specific widget/entity may not implement every generic field.

Key / path	Usage
`renderQuality`	Setting below settings. Read/write with GetSetting/SetSetting, preserving the existing value type. Prefer a dedicated action for immediate reconfiguration.
`globalSceneShaders`	Setting below settings. Read/write with GetSetting/SetSetting, preserving the existing value type. Prefer a dedicated action for immediate reconfiguration.
`globalSceneShaders.layers`	Setting below settings. Read/write with GetSetting/SetSetting, preserving the existing value type. Prefer a dedicated action for immediate reconfiguration.
`globalSceneShaders.preset`	Setting below settings. Read/write with GetSetting/SetSetting, preserving the existing value type. Prefer a dedicated action for immediate reconfiguration.
`globalSceneShaders.intensity`	Setting below settings. Read/write with GetSetting/SetSetting, preserving the existing value type. Prefer a dedicated action for immediate reconfiguration.
`globalSceneShaders.psxPixelSize`	Setting below settings. Read/write with GetSetting/SetSetting, preserving the existing value type. Prefer a dedicated action for immediate reconfiguration.
`globalSceneShaders.psxColorBits`	Color value/string in the form expected by the target property.
`globalSceneShaders.psxDither`	Setting below settings. Read/write with GetSetting/SetSetting, preserving the existing value type. Prefer a dedicated action for immediate reconfiguration.
`globalSceneShaders.psxJitter`	Setting below settings. Read/write with GetSetting/SetSetting, preserving the existing value type. Prefer a dedicated action for immediate reconfiguration.
`globalSceneShaders.n64PixelSize`	Setting below settings. Read/write with GetSetting/SetSetting, preserving the existing value type. Prefer a dedicated action for immediate reconfiguration.
`globalSceneShaders.n64FilterStrength`	Setting below settings. Read/write with GetSetting/SetSetting, preserving the existing value type. Prefer a dedicated action for immediate reconfiguration.
`globalSceneShaders.horrorFogAmount`	Setting below settings. Read/write with GetSetting/SetSetting, preserving the existing value type. Prefer a dedicated action for immediate reconfiguration.
`globalSceneShaders.horrorFogStart`	Setting below settings. Read/write with GetSetting/SetSetting, preserving the existing value type. Prefer a dedicated action for immediate reconfiguration.

Key / path	Usage
`globalSceneShaders.horrorFogEnd`	Setting below settings. Read/write with GetSetting/SetSetting, preserving the existing value type. Prefer a dedicated action for immediate reconfiguration.
`globalSceneShaders.horrorFogColor`	Color value/string in the form expected by the target property.
`globalSceneShaders.horrorFogGrain`	Setting below settings. Read/write with GetSetting/SetSetting, preserving the existing value type. Prefer a dedicated action for immediate reconfiguration.
`globalSceneShaders.horrorFogVignette`	Setting below settings. Read/write with GetSetting/SetSetting, preserving the existing value type. Prefer a dedicated action for immediate reconfiguration.
`enhancedRendering.quality`	Setting below settings. Read/write with GetSetting/SetSetting, preserving the existing value type. Prefer a dedicated action for immediate reconfiguration.
`enhancedRendering.giEnabled`	Boolean on/off state. Visibility and interaction enablement are separate.
`enhancedRendering.ssrEnabled`	Boolean on/off state. Visibility and interaction enablement are separate.
`enhancedRendering.ssaoEnabled`	Boolean on/off state. Visibility and interaction enablement are separate.
`enhancedRendering.reflectionProbeEnabled`	Boolean on/off state. Visibility and interaction enablement are separate.
`enhancedRendering.giEnergy`	Setting below settings. Read/write with GetSetting/SetSetting, preserving the existing value type. Prefer a dedicated action for immediate reconfiguration.
`enhancedRendering.giCellSize`	Setting below settings. Read/write with GetSetting/SetSetting, preserving the existing value type. Prefer a dedicated action for immediate reconfiguration.
`enhancedRendering.giReadSky`	Setting below settings. Read/write with GetSetting/SetSetting, preserving the existing value type. Prefer a dedicated action for immediate reconfiguration.
`enhancedRendering.giOcclusion`	Setting below settings. Read/write with GetSetting/SetSetting, preserving the existing value type. Prefer a dedicated action for immediate reconfiguration.
`enhancedRendering.aoIntensity`	Setting below settings. Read/write with GetSetting/SetSetting, preserving the existing value type. Prefer a dedicated action for immediate reconfiguration.
`enhancedRendering.reflectionProbeSize`	Setting below settings. Read/write with GetSetting/SetSetting, preserving the existing value type. Prefer a dedicated action for immediate reconfiguration.
`cinematic.enabled`	Boolean on/off state. Visibility and interaction enablement are separate.
`cinematic.exposure`	Setting below settings. Read/write with GetSetting/SetSetting, preserving the existing value type. Prefer a dedicated action for immediate reconfiguration.

Key / path	Usage
`cinematic.contrast`	Setting below settings. Read/write with GetSetting/SetSetting, preserving the existing value type. Prefer a dedicated action for immediate reconfiguration.
`cinematic.saturation`	Setting below settings. Read/write with GetSetting/SetSetting, preserving the existing value type. Prefer a dedicated action for immediate reconfiguration.
`cinematic.warmth`	Setting below settings. Read/write with GetSetting/SetSetting, preserving the existing value type. Prefer a dedicated action for immediate reconfiguration.
`cinematic.vignette`	Setting below settings. Read/write with GetSetting/SetSetting, preserving the existing value type. Prefer a dedicated action for immediate reconfiguration.
`cinematic.grain`	Setting below settings. Read/write with GetSetting/SetSetting, preserving the existing value type. Prefer a dedicated action for immediate reconfiguration.
`cinematic.letterbox`	Setting below settings. Read/write with GetSetting/SetSetting, preserving the existing value type. Prefer a dedicated action for immediate reconfiguration.
`subtitles`	Setting below settings. Read/write with GetSetting/SetSetting, preserving the existing value type. Prefer a dedicated action for immediate reconfiguration.
`voiceLanguage`	Setting below settings. Read/write with GetSetting/SetSetting, preserving the existing value type. Prefer a dedicated action for immediate reconfiguration.
`subtitleLanguage`	Setting below settings. Read/write with GetSetting/SetSetting, preserving the existing value type. Prefer a dedicated action for immediate reconfiguration.
`masterVolume`	Normalized float, usually 0 to 1. Use a dedicated playback action for an audio fade.
`musicVolume`	Normalized float, usually 0 to 1. Use a dedicated playback action for an audio fade.
`soundVolume`	Normalized float, usually 0 to 1. Use a dedicated playback action for an audio fade.
`uiScale`	Setting below settings. Read/write with GetSetting/SetSetting, preserving the existing value type. Prefer a dedicated action for immediate reconfiguration.
`fontScale`	Setting below settings. Read/write with GetSetting/SetSetting, preserving the existing value type. Prefer a dedicated action for immediate reconfiguration.
`highContrast`	Setting below settings. Read/write with GetSetting/SetSetting, preserving the existing value type. Prefer a dedicated action for immediate reconfiguration.

Key / path	Usage
`fullscreen`	Setting below settings. Read/write with GetSetting/SetSetting, preserving the existing value type. Prefer a dedicated action for immediate reconfiguration.
`showMainMenuOnStart`	Setting below settings. Read/write with GetSetting/SetSetting, preserving the existing value type. Prefer a dedicated action for immediate reconfiguration.
`customCursor`	Setting below settings. Read/write with GetSetting/SetSetting, preserving the existing value type. Prefer a dedicated action for immediate reconfiguration.
`cursorScale`	Setting below settings. Read/write with GetSetting/SetSetting, preserving the existing value type. Prefer a dedicated action for immediate reconfiguration.

GUI element property keys

These are exact keys offered by the shared property picker in this build. Use them with the matching native helper/action and the expected value type. A type-specific widget/entity may not implement every generic field.

Key / path	Usage
`visible`	Boolean on/off state. Visibility and interaction enablement are separate.
`position`	Match the target's vector type. GUI values usually use Vector2; 3D object transforms use Vector3.
`size`	Match the target's vector type. GUI values usually use Vector2; 3D object transforms use Vector3.
`scale`	Match the target's vector type. GUI values usually use Vector2; 3D object transforms use Vector3.
`rotation_degrees`	Read the existing value to confirm its type, then use the matching helper/action. Type-specific controls can expose extra behavior.
`modulate`	Read the existing value to confirm its type, then use the matching helper/action. Type-specific controls can expose extra behavior.
`mouse_filter`	Read the existing value to confirm its type, then use the matching helper/action. Type-specific controls can expose extra behavior.
`z_index`	Read the existing value to confirm its type, then use the matching helper/action. Type-specific controls can expose extra behavior.
`windowBorderRadius`	Numeric corner/part radius. Use the intended part and its inheritance override; do not write the same value to unrelated widget parts unless intended.
`radiusTopLeft`	Numeric corner/part radius. Use the intended part and its inheritance override; do not write the same value to unrelated widget parts unless intended.
`radiusTopRight`	Numeric corner/part radius. Use the intended part and its inheritance override; do not write the same value to unrelated widget parts unless intended.
`radiusBottomRight`	Numeric corner/part radius. Use the intended part and its inheritance override; do not write the same value to unrelated widget parts unless intended.
`radiusBottomLeft`	Numeric corner/part radius. Use the intended part and its inheritance override; do not write the same value to unrelated widget parts unless intended.
`windowBackgroundEnabled`	Boolean on/off state. Visibility and interaction enablement are separate.
`windowBackground`	Read the existing value to confirm its type, then use the matching helper/action. Type-specific controls can expose extra behavior.
`windowBorderEnabled`	Boolean on/off state. Visibility and interaction enablement are separate.
`windowBorderWidth`	Read the existing value to confirm its type, then use the matching helper/action. Type-specific controls can expose extra behavior.

Key / path	Usage
windowBorderColor	Color value/string in the form expected by the target property.
backgroundImageAssetId	Asset ID String. Import/select the asset; a display label or arbitrary local path is not the same reference.
hoverImageAssetId	Asset ID String. Import/select the asset; a display label or arbitrary local path is not the same reference.
pressedImageAssetId	Asset ID String. Import/select the asset; a display label or arbitrary local path is not the same reference.
backgroundFit	Read the existing value to confirm its type, then use the matching helper/action. Type-specific controls can expose extra behavior.
gridBackgroundAssetId	Asset ID String. Import/select the asset; a display label or arbitrary local path is not the same reference.
gridHoverImageAssetId	Asset ID String. Import/select the asset; a display label or arbitrary local path is not the same reference.
gridPressedImageAssetId	Asset ID String. Import/select the asset; a display label or arbitrary local path is not the same reference.
gridBackgroundFit	Read the existing value to confirm its type, then use the matching helper/action. Type-specific controls can expose extra behavior.
slotBackgroundImageAssetId	Asset ID String. Import/select the asset; a display label or arbitrary local path is not the same reference.
slotHoverImageAssetId	Asset ID String. Import/select the asset; a display label or arbitrary local path is not the same reference.
slotPressedImageAssetId	Asset ID String. Import/select the asset; a display label or arbitrary local path is not the same reference.
slotBackgroundFit	Read the existing value to confirm its type, then use the matching helper/action. Type-specific controls can expose extra behavior.
saveScreenshotBackgroundImageAssetId	Asset ID String. Import/select the asset; a display label or arbitrary local path is not the same reference.
saveScreenshotHoverImageAssetId	Asset ID String. Import/select the asset; a display label or arbitrary local path is not the same reference.
saveScreenshotPressedImageAssetId	Asset ID String. Import/select the asset; a display label or arbitrary local path is not the same reference.
saveScreenshotFit	Read the existing value to confirm its type, then use the matching helper/action. Type-specific controls can expose extra behavior.
gridRadius	Numeric corner/part radius. Use the intended part and its inheritance override; do not write the same value to unrelated widget parts unless intended.
gridRadiusTopLeft	Numeric corner/part radius. Use the intended part and its inheritance override; do not write the same value to unrelated widget parts unless intended.
gridRadiusTopRight	Numeric corner/part radius. Use the intended part and its inheritance override; do not write the same value to unrelated widget parts unless intended.

Key / path	Usage
gridRadiusBottomRight	Numeric corner/part radius. Use the intended part and its inheritance override; do not write the same value to unrelated widget parts unless intended.
gridRadiusBottomLeft	Numeric corner/part radius. Use the intended part and its inheritance override; do not write the same value to unrelated widget parts unless intended.
slotRadius	Numeric corner/part radius. Use the intended part and its inheritance override; do not write the same value to unrelated widget parts unless intended.
slotRadiusTopLeft	Numeric corner/part radius. Use the intended part and its inheritance override; do not write the same value to unrelated widget parts unless intended.
slotRadiusTopRight	Numeric corner/part radius. Use the intended part and its inheritance override; do not write the same value to unrelated widget parts unless intended.
slotRadiusBottomRight	Numeric corner/part radius. Use the intended part and its inheritance override; do not write the same value to unrelated widget parts unless intended.
slotRadiusBottomLeft	Numeric corner/part radius. Use the intended part and its inheritance override; do not write the same value to unrelated widget parts unless intended.
saveScreenshotRadius	Numeric corner/part radius. Use the intended part and its inheritance override; do not write the same value to unrelated widget parts unless intended.
saveScreenshotRadiusTopLeft	Numeric corner/part radius. Use the intended part and its inheritance override; do not write the same value to unrelated widget parts unless intended.
saveScreenshotRadiusTopRight	Numeric corner/part radius. Use the intended part and its inheritance override; do not write the same value to unrelated widget parts unless intended.
saveScreenshotRadiusBottomRight	Numeric corner/part radius. Use the intended part and its inheritance override; do not write the same value to unrelated widget parts unless intended.
saveScreenshotRadiusBottomLeft	Numeric corner/part radius. Use the intended part and its inheritance override; do not write the same value to unrelated widget parts unless intended.
radius	Numeric corner/part radius. Use the intended part and its inheritance override; do not write the same value to unrelated widget parts unless intended.
sliderThumbAssetId	Asset ID String. Import/select the asset; a display label or arbitrary local path is not the same reference.
sliderThumbHoverAssetId	Asset ID String. Import/select the asset; a display label or arbitrary local path is not the same reference.

Key / path	Usage
`sliderThumbSize`	Read the existing value to confirm its type, then use the matching helper/action. Type-specific controls can expose extra behavior.
`modelInitialView`	Read the existing value to confirm its type, then use the matching helper/action. Type-specific controls can expose extra behavior.
`modelStartDistance`	Read the existing value to confirm its type, then use the matching helper/action. Type-specific controls can expose extra behavior.
`modelRotationX`	Read the existing value to confirm its type, then use the matching helper/action. Type-specific controls can expose extra behavior.
`modelRotationY`	Read the existing value to confirm its type, then use the matching helper/action. Type-specific controls can expose extra behavior.
`modelRotationZ`	Read the existing value to confirm its type, then use the matching helper/action. Type-specific controls can expose extra behavior.
`modelPanX`	Read the existing value to confirm its type, then use the matching helper/action. Type-specific controls can expose extra behavior.
`modelPanY`	Read the existing value to confirm its type, then use the matching helper/action. Type-specific controls can expose extra behavior.
`modelFov`	Read the existing value to confirm its type, then use the matching helper/action. Type-specific controls can expose extra behavior.
`modelAnimated`	Read the existing value to confirm its type, then use the matching helper/action. Type-specific controls can expose extra behavior.
`modelAnimation`	Read the existing value to confirm its type, then use the matching helper/action. Type-specific controls can expose extra behavior.
`modelAnimationLoop`	Read the existing value to confirm its type, then use the matching helper/action. Type-specific controls can expose extra behavior.
`modelAnimationSpeed`	Read the existing value to confirm its type, then use the matching helper/action. Type-specific controls can expose extra behavior.
`modelInteractive`	Read the existing value to confirm its type, then use the matching helper/action. Type-specific controls can expose extra behavior.
`modelMouseRotate`	Read the existing value to confirm its type, then use the matching helper/action. Type-specific controls can expose extra behavior.
`modelMousePan`	Read the existing value to confirm its type, then use the matching helper/action. Type-specific controls can expose extra behavior.
`modelMouseResize`	Read the existing value to confirm its type, then use the matching helper/action. Type-specific controls can expose extra behavior.

Whole-GUI property keys

These are exact keys offered by the shared property picker in this build. Use them with the matching native helper/action and the expected value type. A type-specific widget/entity may not implement every generic field.

Key / path	Usage
`visible`	Boolean on/off state. Visibility and interaction enablement are separate.
`position`	Match the target's vector type. GUI values usually use Vector2; 3D object transforms use Vector3.
`size`	Match the target's vector type. GUI values usually use Vector2; 3D object transforms use Vector3.
`scale`	Match the target's vector type. GUI values usually use Vector2; 3D object transforms use Vector3.
`rotation_degrees`	Read the existing value to confirm its type, then use the matching helper/action. Type-specific controls can expose extra behavior.
`modulate`	Read the existing value to confirm its type, then use the matching helper/action. Type-specific controls can expose extra behavior.
`mouse_filter`	Read the existing value to confirm its type, then use the matching helper/action. Type-specific controls can expose extra behavior.
`z_index`	Read the existing value to confirm its type, then use the matching helper/action. Type-specific controls can expose extra behavior.
`windowBorderRadius`	Numeric corner/part radius. Use the intended part and its inheritance override; do not write the same value to unrelated widget parts unless intended.
`radiusTopLeft`	Numeric corner/part radius. Use the intended part and its inheritance override; do not write the same value to unrelated widget parts unless intended.
`radiusTopRight`	Numeric corner/part radius. Use the intended part and its inheritance override; do not write the same value to unrelated widget parts unless intended.
`radiusBottomRight`	Numeric corner/part radius. Use the intended part and its inheritance override; do not write the same value to unrelated widget parts unless intended.
`radiusBottomLeft`	Numeric corner/part radius. Use the intended part and its inheritance override; do not write the same value to unrelated widget parts unless intended.
`windowBackgroundEnabled`	Boolean on/off state. Visibility and interaction enablement are separate.
`windowBackground`	Read the existing value to confirm its type, then use the matching helper/action. Type-specific controls can expose extra behavior.
`windowBorderEnabled`	Boolean on/off state. Visibility and interaction enablement are separate.
`windowBorderWidth`	Read the existing value to confirm its type, then use the matching helper/action. Type-specific controls can expose extra behavior.

Key / path	Usage
`windowBorderColor`	Color value/string in the form expected by the target property.
`backgroundImageAssetId`	Asset ID String. Import/select the asset; a display label or arbitrary local path is not the same reference.
`hoverImageAssetId`	Asset ID String. Import/select the asset; a display label or arbitrary local path is not the same reference.
`pressedImageAssetId`	Asset ID String. Import/select the asset; a display label or arbitrary local path is not the same reference.
`backgroundFit`	Read the existing value to confirm its type, then use the matching helper/action. Type-specific controls can expose extra behavior.

Object-property action keys

These are exact keys offered by the shared property picker in this build. Use them with the matching native helper/action and the expected value type. A type-specific widget/entity may not implement every generic field.

Key / path	Usage
`visible`	Boolean on/off state. Visibility and interaction enablement are separate.
`position`	Match the target’s vector type. GUI values usually use Vector2; 3D object transforms use Vector3.
`rotation_degrees`	Read the existing value to confirm its type, then use the matching helper/action. Type-specific controls can expose extra behavior.
`scale`	Match the target’s vector type. GUI values usually use Vector2; 3D object transforms use Vector3.
`enabled`	Boolean on/off state. Visibility and interaction enablement are separate.
`transform.position`	Read the existing value to confirm its type, then use the matching helper/action. Type-specific controls can expose extra behavior.
`transform.rotation`	Read the existing value to confirm its type, then use the matching helper/action. Type-specific controls can expose extra behavior.
`transform.scale`	Read the existing value to confirm its type, then use the matching helper/action. Type-specific controls can expose extra behavior.

Project property paths

These are exact keys offered by the shared property picker in this build. Use them with the matching native helper/action and the expected value type. A type-specific widget/entity may not implement every generic field.

Key / path	Usage
`meta.title`	Runtime project-data path. GetProjectProperty/SetProjectProperty do not save the editor document.
`meta.version`	Runtime project-data path. GetProjectProperty/SetProjectProperty do not save the editor document.
`meta.startSceneId`	Runtime project-data path. GetProjectProperty/SetProjectProperty do not save the editor document.
`meta.inventoryMode`	Runtime project-data path. GetProjectProperty/SetProjectProperty do not save the editor document.
`meta.interactionMode`	Runtime project-data path. GetProjectProperty/SetProjectProperty do not save the editor document.
`settings.controlProfiles.movementProfile`	Runtime project-data path. GetProjectProperty/SetProjectProperty do not save the editor document.
`settings.controlProfiles.autoSwitchInput`	Runtime project-data path. GetProjectProperty/SetProjectProperty do not save the editor document.
`settings.controlProfiles.inputDevice`	Runtime project-data path. GetProjectProperty/SetProjectProperty do not save the editor document.
`settings.controlProfiles.autoSwitchMovement`	Runtime project-data path. GetProjectProperty/SetProjectProperty do not save the editor document.
`settings.controlProfiles.proximityInteraction`	Runtime project-data path. GetProjectProperty/SetProjectProperty do not save the editor document.
`settings.controlProfiles.coinEnabled`	Boolean on/off state. Visibility and interaction enablement are separate.
`settings.globalSceneShaders`	Runtime project-data path. GetProjectProperty/SetProjectProperty do not save the editor document.
`settings.globalSceneShaders.layers`	Runtime project-data path. GetProjectProperty/SetProjectProperty do not save the editor document.
`settings.globalSceneShaders.preset`	Runtime project-data path. GetProjectProperty/SetProjectProperty do not save the editor document.
`settings.globalSceneShaders.intensity`	Runtime project-data path. GetProjectProperty/SetProjectProperty do not save the editor document.
`settings.globalSceneShaders.psxPixelSize`	Runtime project-data path. GetProjectProperty/SetProjectProperty do not save the editor document.
`settings.globalSceneShaders.psxColorBits`	Color value/string in the form expected by the target property.
`settings.globalSceneShaders.psxDither`	Runtime project-data path. GetProjectProperty/SetProjectProperty do not save the editor document.

Key / path	Usage
settings.globalSceneShaders.psxJitter	Runtime project-data path. GetProjectProperty/SetProjectProperty do not save the editor document.
settings.globalSceneShaders.n64PixelSize	Runtime project-data path. GetProjectProperty/SetProjectProperty do not save the editor document.
settings.globalSceneShaders.n64FilterStrength	Runtime project-data path. GetProjectProperty/SetProjectProperty do not save the editor document.
settings.globalSceneShaders.horrorFogAmount	Runtime project-data path. GetProjectProperty/SetProjectProperty do not save the editor document.
settings.globalSceneShaders.horrorFogStart	Runtime project-data path. GetProjectProperty/SetProjectProperty do not save the editor document.
settings.globalSceneShaders.horrorFogEnd	Runtime project-data path. GetProjectProperty/SetProjectProperty do not save the editor document.
settings.globalSceneShaders.horrorFogColor	Color value/string in the form expected by the target property.
settings.globalSceneShaders.horrorFogGrain	Runtime project-data path. GetProjectProperty/SetProjectProperty do not save the editor document.
settings.globalSceneShaders.horrorFogVignette	Runtime project-data path. GetProjectProperty/SetProjectProperty do not save the editor document.

Quest-objective action keys

These are exact keys offered by the shared property picker in this build. Use them with the matching native helper/action and the expected value type. A type-specific widget/entity may not implement every generic field.

Key / path	Usage
`text`	String. A translation-key field is a key; visible text is content. GUI wildcard substitution belongs to that GUI presentation.
`textKey`	String. A translation-key field is a key; visible text is content. GUI wildcard substitution belongs to that GUI presentation.
`optional`	Boolean on/off state. Visibility and interaction enablement are separate.
`hidden`	Boolean on/off state. Visibility and interaction enablement are separate.

Quest-property action keys

These are exact keys offered by the shared property picker in this build. Use them with the matching native helper/action and the expected value type. A type-specific widget/entity may not implement every generic field.

Key / path	Usage
`title`	String. A translation-key field is a key; visible text is content. GUI wildcard substitution belongs to that GUI presentation.
`titleKey`	String. A translation-key field is a key; visible text is content. GUI wildcard substitution belongs to that GUI presentation.
`description`	String. A translation-key field is a key; visible text is content. GUI wildcard substitution belongs to that GUI presentation.
`descriptionKey`	String. A translation-key field is a key; visible text is content. GUI wildcard substitution belongs to that GUI presentation.
`category`	Read the existing value to confirm its type, then use the matching helper/action. Type-specific controls can expose extra behavior.
`hidden`	Boolean on/off state. Visibility and interaction enablement are separate.
`repeatable`	Boolean on/off state. Visibility and interaction enablement are separate.
`autoTrack`	Boolean on/off state. Visibility and interaction enablement are separate.

Quest-stage action keys

These are exact keys offered by the shared property picker in this build. Use them with the matching native helper/action and the expected value type. A type-specific widget/entity may not implement every generic field.

Key / path	Usage
`title`	String. A translation-key field is a key; visible text is content. GUI wildcard substitution belongs to that GUI presentation.
`titleKey`	String. A translation-key field is a key; visible text is content. GUI wildcard substitution belongs to that GUI presentation.
`description`	String. A translation-key field is a key; visible text is content. GUI wildcard substitution belongs to that GUI presentation.

PCA events: all shared hooks

Native records subscribe through the editor’s Event Functions checkboxes. Each shared hook calls the matching on_* method. A method that is merely named on_scene_loaded does not automatically subscribe itself.

The event Dictionary depends on the emitter. The keys below describe standard runtime emitters; optional/local routes can supply different data. Read optional values with get and an appropriate fallback.

> A global native instance defining on_global_ready receives PCA’s deferred ready call after binding. That special initialization route differs from an ordinary broadcast subscription.

OnGameStart

```
func on_game_start(event: Dictionary = {}) -> void:
    Log(ToJSON(event))
```

The runtime reaches its game-start hook. It is a game-session event, not a once-per-room event.

Common context	Use and limits
sceneId, playerId	Initialize session-level logic here. Do not reset saved puzzle variables on every scene entry. A direct Test Scene can start a test session too; test a normal new game and a loaded game separately.

OnSceneLoaded

```
func on_scene_loaded(event: Dictionary = {}) -> void:
    Log(ToJSON(event))
```

A room has been constructed and its local entry actions have run.

Common context	Use and limits
sceneId, previousSceneId, playerId, firstVisit	Filter sceneId before changing room-specific props. In the normal entry path, firstEnter runs before enter, then this event is emitted. It is not a substitute for the room's first-time event.

OnSceneLeaving

```
func on_scene_leaving(event: Dictionary = {}) -> void:
    Log(ToJSON(event))
```

The scene-loading route is leaving one scene for another.

Common context	Use and limits
sceneId, nextSceneId	sceneId is the departing room. Do short cleanup here. The normal player Change Scene route also runs its local firstLeave/leave action lists; other loading routes need not reproduce every local transition hook.

OnPlayerChanged

```
func on_player_changed(event: Dictionary = {}) -> void:
    Log(ToJSON(event))
```

The controlled playable character changes.

Common context	Use and limits
playerId, displayName	Refresh character-specific presentation. This event concerns player selection, not every movement of a character. The runtime follows it with OnCharacterControl in the switch path.

OnCharacterControl

```
func on_character_control(event: Dictionary = {}) -> void:
    Log(ToJSON(event))
```

A character becomes the controlled character, including initial control at startup.

Common context	Use and limits
playerId, displayName	Use it for HUD/inventory presentation or control-specific setup. It also has a per-character On Control script hook. This event is a special case in which playerId can identify the calling character for this.

OnItemFound

```
func on_item_found(event: Dictionary = {}) -> void:
    Log(ToJSON(event))
```

The active-inventory add route adds an item that was previously absent from that bag.

Common context	Use and limits
characterId, itemId, amount	It is based on previous possession, not a permanent lifetime discovery log. Remove the last copy and acquire it again, and this route can report it again. Not every NPC/character transfer route emits this event; use a story flag for one-time discoveries.

OnVideoEnd

```
func on_video_end(event: Dictionary = {}) -> void:
    Log(ToJSON(event))
```

The runtime receives the video-finished callback.

Common context	Use and limits
videoAssetId	Filter the asset ID. Do not assume an explicit Stop Video or an interrupted/looping clip has the same natural-end behavior. Dedicated On Video End graph nodes are an additional entry route; avoid duplicating the same reward in both routes.

OnMusicEnd

```
func on_music_end(event: Dictionary = {}) -> void:
    Log(ToJSON(event))
```

The music-finished callback reaches the runtime.

Common context	Use and limits
musicAssetId	Use a non-looping track when testing. Stopping or replacing a track is not the same as natural completion. Dedicated On Music End graph nodes can also listen for a selected asset.

OnDialogueStarted

```
func on_dialogue_started(event: Dictionary = {}) -> void:
    Log(ToJSON(event))
```

A valid dialogue session begins, before its graph is processed.

Common context	Use and limits
dialogueId, playerId; inherited caller context may also be present	Filter dialogueId before adjusting a conversation camera or HUD. Starting the same dialogue from this handler can recurse. Keep listener work short when it must complete before the first line appears.

OnDialogueEnded

```
func on_dialogue_ended(event: Dictionary = {}) -> void:
    Log(ToJSON(event))
```

A dialogue session ends, including interrupted endings.

Common context	Use and limits
dialogueId, playerId, interrupted; inherited caller context may also be present	Check interrupted before granting a reward that belongs only to successful completion. Some explicit end paths can supply a blank dialogueId, so do not assume every payload identifies a normal completed graph.

OnInventoryChanged

```
func on_inventory_changed(event: Dictionary = {}) -> void:
    Log(ToJSON(event))
```

An inventory mutation route reports an add, remove or quantity change.

Common context	Use and limits
characterId, itemId, amount, operation	Use operation and the actual bag state together. amount is supplied by that mutation route; it is not guaranteed to be the resulting total. Adding/removing inside this listener can trigger another inventory event, so guard feedback loops.

OnSaveLoaded

```
func on_save_loaded(event: Dictionary = {}) -> void:
    Log(ToJSON(event))
```

A save has successfully restored game state.

Common context	Use and limits
slot, sceneId, playerId	Refresh presentation from restored variables. Do not award a pickup again merely because its saved state was loaded. This is a successful-load hook; a missing or invalid save should not be treated as a completed restore.

OnCustomEvent

```
func on_custom_event(event: Dictionary = {}) -> void:
    Log(ToJSON(event))
```

A script or Emit Event graph node explicitly emits this event name.

Common context	Use and limits
Whatever keys the emitter sends; eventName is added for the listener	Use a reason key, such as signal_powered, to distinguish several messages sharing OnCustomEvent. It is not automatically emitted by every button or variable edit. The graph form uses its Context JSON; the GDScript helper merges its current caller context before emitting.

OnGlobalReady

```
func on_global_ready(event: Dictionary = {}) -> void:
    Log(ToJSON(event))
```

A persistent/global script is initialized through its language runtime's ready route.

Common context	Use and limits
Language-specific startup context; native GDScript includes eventName and scriptId	Use it to initialize that global instance, not to assume a playable room is ready. The GDScript runtime calls a global script's matching method directly. The visual On Event picker does not offer this ready event; do not design it as a universal room-ready graph broadcast.

OnInteraction

```
func on_interaction(event: Dictionary = {}) -> void:
    Log(ToJSON(event))
```

A scene-entity verb interaction reaches its execution point.

Common context	Use and limits
entityId, targetEntityId, targetKind, sceneId, playerId, characterId, verb, inventoryItemId, sourceItemId; regionId for a region	This is the shared notification before the local verb action list. Filter both target and verb. The local list receives a related but smaller context; do not assume every key shown here is passed unchanged to every nested call.

OnTriggerEntered

```
func on_trigger_entered(event: Dictionary = {}) -> void:
    Log(ToJSON(event))
```

The active player enters an eligible trigger/interactive-region overlap.

Common context	Use and limits
triggerId, entityId, sceneId, playerId, event="enter"; regionId for a region	In the checked 3D route, other NPC bodies do not trigger this player-entry hook. The shared event runs before local onEnter actions. Starting inside a volume can create startup contact behavior; test that case separately.

OnTriggerExited

```
func on_trigger_exited(event: Dictionary = {}) -> void:
    Log(ToJSON(event))
```

The active player leaves an eligible trigger/interactive-region overlap.

Common context	Use and limits
triggerId, entityId, sceneId, playerId, event="exit"; regionId for a region	Use it for leaving a local effect area. Teleports and room changes are not guaranteed to behave like ordinary walking out of the same live volume; perform important cleanup in the owning scene/sequence too.

OnAttachPointContact

```
func on_attach_point_contact(event: Dictionary = {}) -> void:
    Log(ToJSON(event))
```

A named attach-point watch reports entering or leaving point contact.

Common context	Use and limits
watchId, phase, entityId, point, targetId, targetPoint, sceneId, touching, swept, position, distance, fraction, contactRadius, dimension	Filter watchId and phase="enter" for a one-shot success. A swept contact may report enter and exit between samples. Stop the watch when its job ends and set a completion flag before awarding the result.

Godot global functions: the checked runtime

The installed Godot 4.7.2 runtime exposes the following 130 global/GDScript functions. This inventory was generated from that executable’s API documentation, so names, parameters and defaults match the checked runtime. It is separate from PCA’s 78 capitalized helpers.

Constructors such as Vector3, Color, Array and Dictionary are built-in types, not extra functions missing from this list. Object methods live in the class reference. The tutorial covers their normal patterns; the official reference covers the larger engine API.

> A signature is a reference, not a script to execute all at once. Some functions are low-level, diagnostic or compatibility facilities. Prefer the clear native method/typed form for new code.

GDScript-specific globals

Color8()

```
Color8(r8: int, g8: int, b8: int, a8: int = 255) -> Color
```

Construct a Color from 8-bit channel values.

assert()

```
assert(condition: bool, message: String = "") -> void
```

Debug assertion; do not use as release-build input validation.

char()

```
char(code: int) -> String
```

Convert between a Unicode code point and a character.

convert()

```
convert(what: Variant, type: int) -> Variant
```

Compatibility type conversion. Prefer current type_convert/constructors where appropriate.

dict_to_inst()

```
dict_to_inst(dictionary: Dictionary) -> Object
```

Compatibility object/dictionary conversion; use an explicit data format for portable game state.

get_stack()

```
get_stack() -> Array
```

Inspect or print native debug stack/location information.

inst_to_dict()

```
inst_to_dict(instance: Object) -> Dictionary
```

Compatibility object/dictionary conversion; use an explicit data format for portable game state.

is_instance_of()

```
is_instance_of(value: Variant, type: Variant) -> bool
```

Test a value against a supplied type/class.

len()

```
len(var: Variant) -> int
```

Length/size of a supported String or container.

load()

```
load(path: String) -> Resource
```

Load a Resource by a runtime path in the running Godot resource environment.

ord()

```
ord(char: String) -> int
```

Convert between a Unicode code point and a character.

preload()

```
preload(path: String) -> Resource
```

Prepare a Resource from a constant path while the script is loaded.

print_debug()

```
print_debug(...) -> void
```

Inspect or print native debug stack/location information.

print_stack()

```
print_stack() -> void
```

Inspect or print native debug stack/location information.

range()

```
range(...) -> Array
```

Integer range for loops; stop is excluded and a negative step counts down.

type_exists()

```
type_exists(type: StringName) -> bool
```

Check whether an exposed named type exists.

Shared global functions

abs()

```
abs(x: Variant) -> Variant
```

Absolute value; suffix f/i selects a float/integer form.

absf()

```
absf(x: float) -> float
```

Absolute value; suffix f/i selects a float/integer form.

absi()

```
absi(x: int) -> int
```

Absolute value; suffix f/i selects a float/integer form.

acos()

```
acos(x: float) -> float
```

Inverse trigonometric/hyperbolic calculation; angle results use radians where applicable.

acosh()

```
acosh(x: float) -> float
```

Inverse trigonometric/hyperbolic calculation; angle results use radians where applicable.

angle_difference()

```
angle_difference(from: float, to: float) -> float
```

Signed shortest difference between angles in radians.

asin()

```
asin(x: float) -> float
```

Inverse trigonometric/hyperbolic calculation; angle results use radians where applicable.

asinh()

```
asinh(x: float) -> float
```

Inverse trigonometric/hyperbolic calculation; angle results use radians where applicable.

atan()

```
atan(x: float) -> float
```

Inverse trigonometric/hyperbolic calculation; angle results use radians where applicable.

atan2()

```
atan2(y: float, x: float) -> float
```

Inverse trigonometric/hyperbolic calculation; angle results use radians where applicable.

atanh()

```
atanh(x: float) -> float
```

Inverse trigonometric/hyperbolic calculation; angle results use radians where applicable.

bezier_derivative()

```
bezier_derivative(start: float, control_1: float, control_2: float, end: float, t: float) -> float
```

Evaluate a cubic Bezier curve or its derivative.

bezier_interpolate()

```
bezier_interpolate(start: float, control_1: float, control_2: float, end: float, t: float) -> float
```

Evaluate a cubic Bezier curve or its derivative.

bytes_to_var()

```
bytes_to_var(bytes: PackedByteArray) -> Variant
```

Convert supported Variant data to/from binary data without object encoding.

bytes_to_var_with_objects()

```
bytes_to_var_with_objects(bytes: PackedByteArray) -> Variant
```

Binary Variant conversion that can include objects; use only with an intentional trusted-data format.

ceil()

```
ceil(x: Variant) -> Variant
```

Round upward; the suffix selects a typed variant.

ceilf()

```
ceilf(x: float) -> float
```

Round upward; the suffix selects a typed variant.

ceili()

```
ceili(x: float) -> int
```

Round upward; the suffix selects a typed variant.

clamp()

```
clamp(value: Variant, min: Variant, max: Variant) -> Variant
```

Limit a value between minimum and maximum.

clampf()

```
clampf(value: float, min: float, max: float) -> float
```

Limit a value between minimum and maximum.

clampi()

```
clampi(value: int, min: int, max: int) -> int
```

Limit a value between minimum and maximum.

cos()

```
cos(angle_rad: float) -> float
```

Trigonometric/hyperbolic calculation; ordinary angle inputs use radians.

cosh()

```
cosh(x: float) -> float
```

Trigonometric/hyperbolic calculation; ordinary angle inputs use radians.

cubic_interpolate()

```
cubic_interpolate(from: float, to: float, pre: float, post: float, weight: float) -> float
```

Cubic interpolation using surrounding values, optionally time samples or angle handling.

cubic_interpolate_angle()

```
cubic_interpolate_angle(from: float, to: float, pre: float, post: float, weight: float) -> float
```

Cubic interpolation using surrounding values, optionally time samples or angle handling.

cubic_interpolate_angle_in_time()

```
cubic_interpolate_angle_in_time(from: float, to: float, pre: float, post: float, weight: float, to_t: float, pre_t: float, post_t: float) -> float
```

Cubic interpolation using surrounding values, optionally time samples or angle handling.

cubic_interpolate_in_time()

```
cubic_interpolate_in_time(from: float, to: float, pre: float, post: float, weight: float, to_t: float, pre_t: float, post_t: float) -> float
```

Cubic interpolation using surrounding values, optionally time samples or angle handling.

db_to_linear()

```
db_to_linear(db: float) -> float
```

Convert audio amplitude between decibels and linear volume.

deg_to_rad()

```
deg_to_rad(deg: float) -> float
```

Convert angles between degrees and radians.

ease()

```
ease(x: float, curve: float) -> float
```

Shape a transition weight/curve; useful for animation and blending.

error_string()

```
error_string(error: int) -> String
```

Human-readable description of an Error code.

exp()

```
exp(x: float) -> float
```

Exponential, natural logarithm, power or square root. Check the mathematical domain.

floor()

```
floor(x: Variant) -> Variant
```

Round downward; the suffix selects a typed variant.

floorf()

```
floorf(x: float) -> float
```

Round downward; the suffix selects a typed variant.

floori()

```
floori(x: float) -> int
```

Round downward; the suffix selects a typed variant.

fmod()

```
fmod(x: float, y: float) -> float
```

Remainder/modulo variants; pay attention to negative inputs and positive wrapping behavior.

fposmod()

```
fposmod(x: float, y: float) -> float
```

Remainder/modulo variants; pay attention to negative inputs and positive wrapping behavior.

hash()

```
hash(variable: Variant) -> int
```

Hash a value for lookup/comparison support; not a serialization or cryptographic guarantee.

instance_from_id()

```
instance_from_id(instance_id: int) -> Object
```

Validate an instance ID or look up the live Object for that ID.

inverse_lerp()

```
inverse_lerp(from: float, to: float, weight: float) -> float
```

Find the interpolation weight represented by a value in an interval.

is_equal_approx()

```
is_equal_approx(a: float, b: float) -> bool
```

Approximate floating-point equality/zero checks.

is_finite()

```
is_finite(x: float) -> bool
```

Check finite, infinite or invalid numeric values.

is_inf()

```
is_inf(x: float) -> bool
```

Check finite, infinite or invalid numeric values.

is_instance_id_valid()

```
is_instance_id_valid(id: int) -> bool
```

Validate an instance ID or look up the live Object for that ID.

is_instance_valid()

```
is_instance_valid(instance: Variant) -> bool
```

Check whether a Godot Object is still alive.

is_nan()

```
is_nan(x: float) -> bool
```

Check finite, infinite or invalid numeric values.

is_same()

```
is_same(a: Variant, b: Variant) -> bool
```

Check strict value/identity sameness; not an approximate numeric comparison.

is_zero_approx()

```
is_zero_approx(x: float) -> bool
```

Approximate floating-point equality/zero checks.

lerp()

```
lerp(from: Variant, to: Variant, weight: Variant) -> Variant
```

Linear interpolation. A weight outside 0-1 can extrapolate.

lerp_angle()

```
lerp_angle(from: float, to: float, weight: float) -> float
```

Interpolate angles along the appropriate circular route.

lerpf()

```
lerpf(from: float, to: float, weight: float) -> float
```

Linear interpolation. A weight outside 0-1 can extrapolate.

linear_to_db()

```
linear_to_db(lin: float) -> float
```

Convert audio amplitude between decibels and linear volume.

log()

```
log(x: float) -> float
```

Exponential, natural logarithm, power or square root. Check the mathematical domain.

max()

```
max(...) -> Variant
```

Largest supplied value; match typed/variadic signature.

maxf()

```
maxf(a: float, b: float) -> float
```

Largest supplied value; match typed/variadic signature.

maxi()

```
maxi(a: int, b: int) -> int
```

Largest supplied value; match typed/variadic signature.

min()

```
min(...) -> Variant
```

Smallest supplied value; match typed/variadic signature.

minf()

```
minf(a: float, b: float) -> float
```

Smallest supplied value; match typed/variadic signature.

mini()

```
mini(a: int, b: int) -> int
```

Smallest supplied value; match typed/variadic signature.

move_toward()

```
move_toward(from: float, to: float, delta: float) -> float
```

Move a value toward a target without overshooting.

nearest_po2()

```
nearest_po2(value: int) -> int
```

Smallest suitable power-of-two boundary for a positive integer.

pingpong()

```
pingpong(value: float, length: float) -> float
```

Reflect a repeating value back and forth across an interval.

posmod()

```
posmod(x: int, y: int) -> int
```

Remainder/modulo variants; pay attention to negative inputs and positive wrapping behavior.

pow()

```
pow(base: float, exp: float) -> float
```

Exponential, natural logarithm, power or square root. Check the mathematical domain.

print()

```
print(...) -> void
```

Write diagnostic output, with spacing/tab/rich/verbose/error variants.

print_rich()

```
print_rich(...) -> void
```

Write diagnostic output, with spacing/tab/rich/verbose/error variants.

print_verbose()

```
print_verbose(...) -> void
```

Write diagnostic output, with spacing/tab/rich/verbose/error variants.

printerr()

```
printerr(...) -> void
```

Write diagnostic output, with spacing/tab/rich/verbose/error variants.

printraw()

```
printraw(...) -> void
```

Write diagnostic output, with spacing/tab/rich/verbose/error variants.

prints()

```
prints(...) -> void
```

Write diagnostic output, with spacing/tab/rich/verbose/error variants.

printt()

```
printt(...) -> void
```

Write diagnostic output, with spacing/tab/rich/verbose/error variants.

push_error()

```
push_error(...) -> void
```

Report a development diagnostic. These are not catchable exceptions.

push_warning()

```
push_warning(...) -> void
```

Report a development diagnostic. These are not catchable exceptions.

rad_to_deg()

```
rad_to_deg(rad: float) -> float
```

Convert angles between degrees and radians.

rand_from_seed()

```
rand_from_seed(seed: int) -> PackedInt64Array
```

Generate a seeded result together with updated seed state.

randf()

```
randf() -> float
```

Global random float, ranged float or normally distributed float.

randf_range()

```
randf_range(from: float, to: float) -> float
```

Global random float, ranged float or normally distributed float.

randfn()

```
randfn(mean: float, deviation: float) -> float
```

Global random float, ranged float or normally distributed float.

randi()

```
randi() -> int
```

Global random integer value; a range includes its endpoints.

randi_range()

```
randi_range(from: int, to: int) -> int
```

Global random integer value; a range includes its endpoints.

randomize()

```
randomize() -> void
```

Set the global random seed or randomize it. Use a separate generator for isolated puzzle randomness.

remap()

```
remap(value: float, istart: float, istop: float, ostart: float, ostop: float) -> float
```

Convert a value from one numeric range to another.

rid_allocate_id()

```
rid_allocate_id() -> int
```

Low-level resource identifier operations; not ordinary PCA record IDs.

rid_from_int64()

```
rid_from_int64(base: int) -> RID
```

Low-level resource identifier operations; not ordinary PCA record IDs.

rotate_toward()

```
rotate_toward(from: float, to: float, delta: float) -> float
```

Move an angle toward a target by a maximum angular step.

round()

```
round(x: Variant) -> Variant
```

Round to nearest whole value; the suffix selects a typed variant.

roundf()

```
roundf(x: float) -> float
```

Round to nearest whole value; the suffix selects a typed variant.

roundi()

```
roundi(x: float) -> int
```

Round to nearest whole value; the suffix selects a typed variant.

seed()

```
seed(base: int) -> void
```

Set the global random seed or randomize it. Use a separate generator for isolated puzzle randomness.

sign()

```
sign(x: Variant) -> Variant
```

Sign of a value as negative, zero or positive unit.

signf()

```
signf(x: float) -> float
```

Sign of a value as negative, zero or positive unit.

signi()

```
signi(x: int) -> int
```

Sign of a value as negative, zero or positive unit.

sin()

```
sin(angle_rad: float) -> float
```

Trigonometric/hyperbolic calculation; ordinary angle inputs use radians.

sinh()

```
sinh(x: float) -> float
```

Trigonometric/hyperbolic calculation; ordinary angle inputs use radians.

smoothstep()

```
smoothstep(from: float, to: float, x: float) -> float
```

Shape a transition weight/curve; useful for animation and blending.

snapped()

```
snapped(x: Variant, step: Variant) -> Variant
```

Snap a value to the supplied increment.

snappedf()

```
snappedf(x: float, step: float) -> float
```

Snap a value to the supplied increment.

snappedi()

```
snappedi(x: float, step: int) -> int
```

Snap a value to the supplied increment.

sqrt()

```
sqrt(x: float) -> float
```

Exponential, natural logarithm, power or square root. Check the mathematical domain.

step_decimals()

```
step_decimals(x: float) -> int
```

Decimal precision needed for an increment; useful for display/editing.

str()

```
str(...) -> String
```

Convert one or more values to String text.

str_to_var()

```
str_to_var(string: String) -> Variant
```

Convert supported Variant data to/from Godot’s text representation.

tan()

```
tan(angle_rad: float) -> float
```

Trigonometric/hyperbolic calculation; ordinary angle inputs use radians.

tanh()

```
tanh(x: float) -> float
```

Trigonometric/hyperbolic calculation; ordinary angle inputs use radians.

type_convert()

```
type_convert(variant: Variant, type: int) -> Variant
```

Query a Variant type ID, name a type ID, or convert to a requested type.

type_string()

```
type_string(type: int) -> String
```

Query a Variant type ID, name a type ID, or convert to a requested type.

typeof()

```
typeof(variable: Variant) -> int
```

Query a Variant type ID, name a type ID, or convert to a requested type.

var_to_bytes()

```
var_to_bytes(variable: Variant) -> PackedByteArray
```

Convert supported Variant data to/from binary data without object encoding.

var_to_bytes_with_objects()

```
var_to_bytes_with_objects(variable: Variant) -> PackedByteArray
```

Binary Variant conversion that can include objects; use only with an intentional trusted-data format.

var_to_str()

```
var_to_str(variable: Variant) -> String
```

Convert supported Variant data to/from Godot’s text representation.

weakref()

```
weakref(obj: Variant) -> Variant
```

Hold a weak reference that does not keep the Object alive.

wrap()

```
wrap(value: Variant, min: Variant, max: Variant) -> Variant
```

Wrap a value into a repeating interval.

wrapf()

```
wrapf(value: float, min: float, max: float) -> float
```

Wrap a value into a repeating interval.

wrapi()

```
wrapi(value: int, min: int, max: int) -> int
```

Wrap a value into a repeating interval.

Godot class map and API lookup

These are useful entry points, not a replacement for Godot’s entire class library. The named members below are starting points. Open the official class reference for complete signatures, inherited members, signals and platform notes.

The class documentation generated from the installed runtime was also used to check the examples. Stable online documentation can describe a newer release later, so validate a copied example against the version shipped with your PCA build.

> A class being available does not automatically configure its content, native extension binaries, platform permissions or export files. A node still needs the correct parent and lifetime.

Object

Common base for Godot objects.

Useful members: `get`, `set`, `call`, `has_method`, `get_method_list`, `get_property_list`, `connect`, `is_class`.

Official Object reference

Node

Tree structure, lifetime, groups and callbacks.

Useful members: `add_child`, `get_node_or_null`, `get_children`, `queue_free`, `add_to_group`, `get_tree`, `create_tween`.

Official Node reference

SceneTree

The live Godot tree and scheduling.

Useful members: `create_timer`, `process_frame`, `physics_frame`, `get_nodes_in_group`, `call_group`.

Official SceneTree reference

Timer

An owned repeatable/one-shot timer node.

Useful members: `start`, `stop`, `timeout`, `wait_time`, `one_shot`, `time_left`.

Official Timer reference

Tween

Property animation and sequencing.

Useful members: `tween_property`, `tween_callback`, `set_parallel`, `parallel`, `chain`, `finished`, `kill`.

Official Tween reference

Input

Native input state.

Useful members: `is_action_pressed`, `is_action_just_pressed`, `get_vector`, `is_key_pressed`.

Official Input reference

InputMap

Define/query named native input actions.

Useful members: `has_action`, `add_action`, `action_add_event`, `action_get_events`.

Official InputMap reference

Control

Base for native interface controls.

Useful members: `position`, `size`, `mouse_filter`, `gui_input`, `grab_focus`, `accept_event`.

Official Control reference

Button

A native clickable control.

Useful members: `pressed`, `button_down`, `button_up`, `text`, `disabled`.

Official Button reference

LineEdit

Single-line native text input.

Useful members: `text`, `placeholder_text`, `text_changed`, `text_submitted`, `caret_column`.

Official LineEdit reference

Node3D

Spatial transforms and world access.

Useful members: `position`, `global_position`, `rotation`, `rotation_degrees`, `scale`, `get_world_3d`.

Official Node3D reference

MeshInstance3D

Render a mesh with materials.

Useful members: `mesh`, `material_override`, `get_active_material`, `set_surface_override_material`.

Official MeshInstance3D reference

StandardMaterial3D

Common physically based material settings.

Useful members: `albedo_color`, `roughness`, `metallic`, `emission_enabled`, `emission`.

Official StandardMaterial3D reference

ShaderMaterial

Material with named shader uniforms.

Useful members: `shader`, `set_shader_parameter`, `get_shader_parameter`.

Official ShaderMaterial reference

AnimationPlayer

Native animation playback.

Useful members: `play`, `stop`, `pause`, `has_animation`, `animation_finished`, `get_animation_list`.

Official AnimationPlayer reference

Resource

Reference-counted engine data.

Useful members: `duplicate`, `resource_path`, `changed`, `emit_changed`.

Official Resource reference

ResourceLoader

Export-aware resource lookup/loading.

Useful members: `exists`, `load`, `load_threaded_request`, `load_threaded_get_status`, `load_threaded_get`.

Official ResourceLoader reference

PackedScene

A saved Godot scene resource.

Useful members: `instantiate`, `pack`, `can_instantiate`.

Official PackedScene reference

FileAccess

Read/write byte and text files.

Useful members: `open`, `get_as_text`, `get_line`, `store_string`, `store_line`, `seek`, `seek_end`, `close`.

Official FileAccess reference

DirAccess

Directory and file-management operations.

Useful members: `open`, `get_files`, `get_directories`, `make_dir_recursive_absolute`, `rename_absolute`, `remove_absolute`.

Official DirAccess reference

JSON

JSON serialization with parse diagnostics.

Useful members: `stringify`, `parse_string`, `parse`, `data`, `get_error_message`, `get_error_line`.

Official JSON reference

ConfigFile

Section/key configuration files.

Useful members: `load`, `save`, `get_value`, `set_value`, `has_section_key`.

Official ConfigFile reference

Time

Calendar time and monotonic ticks.

Useful members: `get_ticks_msec`, `get_ticks_usec`, `get_unix_time_from_system`, `get_datetime_dict_from_system`.

Official Time reference

Engine

Runtime version and engine state.

Useful members: `get_version_info`, `is_editor_hint`, `time_scale`, `max_fps`.

Official Engine reference

ClassDB

Inspect exposed engine classes.

Useful members: `class_exists`, `can_instantiate`, `instantiate`, `class_get_method_list`, `class_get_property_list`.

Official ClassDB reference

RandomNumberGenerator

An isolated random sequence.

Useful members: `seed`, `state`, `randomize`, `randi_range`, `randf_range`, `randfn`.

Official RandomNumberGenerator reference

PhysicsDirectSpaceState3D

Queries against a World3D physics space.

Useful members: `intersect_ray`, `intersect_shape`, `cast_motion`.

Official PhysicsDirectSpaceState3D reference

AudioStreamPlayer

Native non-positional audio node.

Useful members: `stream`, `play`, `stop`, `finished`, `volume_db`, `bus`.

Official AudioStreamPlayer reference

HTTPRequest

An owned HTTP request node.

Useful members: `request`, `cancel_request`, `request_completed`; check `Error` and response status.

Official HTTPRequest reference

Thread

Explicit background work.

Useful members: `start`, `is_started`, `is_alive`, `wait_to_finish`; keep scene-tree edits on the main thread.

Official Thread reference

Godot resources in an exported PCA project

PCA’s authored native records are stored in the game data and prepared at runtime. They are not automatically written to individual .gd files with matching record IDs. CallScript is the record router; ResourceLoader loads real resources.

1. Use Build Game → Export Editable Godot Project and choose a new destination. Open its project.godot in the matching Godot editor.
2. Add a custom folder for new .gd, .tscn, .tres and other resources. Let Godot import them.
3. Reference those real paths from your bridge code. For a custom helper Script use preload/load plus new; for a PackedScene use instantiate.
4. Run the full PCA game under Godot, then export using its presets with the required files and plugins included.
5. Keep those custom additions when exporting a fresh source project later; a fresh export is not an automatic merge.

Native extensions and addons can require platform-specific binaries. File/network/OS behavior can differ on desktop, mobile and web. Test the produced build, including the first launch from a clean folder, before treating editor success as distribution success.

Official Godot references

The guide’s PCA API inventory was checked against the installed application source. The following official references provide the broader Godot language/class documentation. Their online stable branch can move forward independently of the checked application build.

GDScript language reference

Language syntax, inheritance, functions, signals and await.

Node and the scene tree

Node ownership, processing, lifecycle and input callbacks.

SceneTree timers

Timer scheduling and pause/time-scale parameters.

FileAccess

File modes, cursor operations and exported-resource considerations.

Resource

Resource lifetime and sharing.

Godot class index

The complete online class library, including methods not repeated in this book.

GDScript quick reference

Job	Native GDScript form
Comment	`# comment`; not //
Declaration	`var count: int = 0` or `var count := 0`
Constant / enum	`const LIMIT = 10`; `enum State { LOCKED, OPEN }`
Function	`func name(value: int = 1) -> int:` followed by an indented body
Decision	if / elif / else, each with a colon
Conditional value	`a if condition else b`
State selection	match with indented patterns and _ fallback
Loop	for value in collection / range; while condition
Loop control	break / continue
Function exit	return / return value
Array	`[1, 2]`, `Array[String]`, append/size/has
Dictionary	`{"open": false}`, get/has/keys/erase
Wait	`await signal` or `await coroutine_call()`
Signal	signal declaration; connect Callable; emit(arguments)
Inheritance	extends one base; super calls parent
Reuse	preload/load for files; CallScript/IncludeScript for PCA records
Native context	self = Node instance; this = PCA caller proxy; player = character proxy

Case matters. Godot native globals usually use lowercase snake_case. PCA helpers/actions use the capitalized names in their reference. Dictionary keys keep their own exact spelling.

PCA Script to GDScript: important differences

PCA Script	Native GDScript
function / end	func / colon / indentation
let / global / const	var / const; saved game globals through Variable
when room lab loads	on_scene_loaded method plus subscription checkbox
set value to 5	value = 5
say "text"	await Say({"text": "text", ...}) or a character proxy Say method
Automatic action suspension	Use await when the caller needs completion/result
try / catch	No native try/catch; guard and inspect Error/status values
include_script returns ID	IncludeScript returns a script-call proxy
Record reference / VM value	Native RefCounted record proxy object
PCA regex match dictionary	Native RegExMatch object
PCA VM debugger	Native Godot debugger for GDScript
PCA array helpers are limited	Native Array/String/Dictionary methods are available

Do not translate a script by changing function to func and leaving everything else alone. Event wiring, waits, variable lifetime, helper capitalization and return contracts need deliberate conversion.

Troubleshooting native scripts

Symptom	Check
Say/Variable/Action is unknown	Did an explicit extends Node replace the PCA base? Keep the inherited PCAScriptBase route.
Runtime is null in _ready	PCA binds after adding the script Node. Use on_global_ready or a routed call for PCA-dependent startup.
A timer stops after run returns	The record may be temporary. Keep a global instance or await the complete finite job.
Member values reset	Ordinary records get a fresh instance each call. Use a global member for session state or Variable for saved state.
A handler never runs	Check the Event Functions subscription and exact on_* method.
A local name causes warnings/confusion	Avoid hiding inherited context, Runtime, player, name or a helper method.
A proxy looks valid but the record is missing	Use proxy.Exists or registry.Has; IsValid only validates the proxy object.
A child path cannot be found	The script Node is not the calling object’s rendered Node. Inspect the correct hierarchy.
Resource load fails only in a build	The actual resource file/import/dependencies may not be included. A record ID is not a res:// .gd file.
A visual change is overwritten	Another controller/animation owns the property, or the write changed only a definition.
Room transition destroys the editor	Use PCA ChangeScene, not SceneTree.change_scene_to_file, for adventure rooms.
A stale node fails after room change	Use is_instance_valid and reacquire the native node when the room/actor rebuilds.
TryCall did not recover an error	It validates a Callable; it cannot catch arbitrary GDScript runtime errors.

Reference basis and practical limits

The PCA API was checked against PCAScriptBase, the shared action catalog/runner, script source preparation/router, registries/proxies and authored GUI runtime in the installed 0.86.3.4 build. The Godot global-function inventory came from the installed 4.7.2 executable’s generated API documentation.

All 206 normal gameplay wrappers and 78 public non-action helpers are documented. The registry/proxy surface, property catalog, shared picker keys and 18 event hooks are included. The 130 native global functions and 30 class-map entries give you a route into the broader Godot API.

The book does not claim to reproduce every method on every Godot class. Use the linked official reference and local editor help for that library, and validate against the runtime your players will receive. A later stable documentation page can describe changes not present in an older packaged engine.

Most entity actions operate on the loaded room’s instances, even where a shared schema includes a room-selection field. Supplying an ID does not load another room. Use the dedicated offscreen/character-room route when that is the intended job.

No engine source or user project was modified to create this guide. The deliverable is this PDF; the examples and screenshots do not require downloadable practice files.

Alphabetical API and event index

Click an entry to jump to its reference. The PDF outline/bookmarks provide a second route through tutorial chapters, APIs, designer records and Godot classes.

Entry	Kind	Entry	Kind
abs	Godot global	absf	Godot global
absi	Godot global	acos	Godot global
acosh	Godot global	Action	PCA helper
ActivateCamera	Action	ActivePlayer	PCA helper
AddAssetToScene	Action	AddCharacterToScene	Action
AddCharacterWaypoint	Action	AddDialogueChoice	Action
AddDialogueChoiceNode	Action	AddDialogueLine	Action
AddInventory	Action	AddScore	Action
AlignAttachPoints	Action	angle_difference	Godot global
AnimationPlayer	Godot class	asin	Godot global
asinh	Godot global	assert	Godot global
Assets	Designer	atan	Godot global
atan2	Godot global	atanh	Godot global
AttachObject	Action	AttachParticleEffect	Action
AttachPoints	Action	AttachPointsTouching	Action
AudioStreamPlayer	Godot class	BASIC script records	Designer
bezier_derivative	Godot global	bezier_interpolate	Godot global
Breakpoint	PCA helper	Button	Godot class
bytes_to_var	Godot global	bytes_to_var_with_objects	Godot global
CallScript	PCA helper	CallScriptFunction	Action
CameraOrbit	Action	Cameras	Designer
CameraShake	Action	CancelPreposition	Action
CarouselAddImage	Action	CarouselClearImages	Action
CarouselNext	Action	CarouselPrevious	Action
CarouselRemoveImage	Action	ceil	Godot global
ceilf	Godot global	ceili	Godot global
CenterGUIModel	Action	ChangeCamera	Action
ChangeCharacterDescription	Action	ChangeCharacterProperty	Action
ChangeGameCursor	Action	ChangeGameSetting	Action
ChangeGUIElementProperty	Action	ChangeGUIProperty	Action
ChangeInventoryDescription	Action	ChangeInventoryQuantity	Action
ChangeObjectDescription	Action	ChangeObjectProperty	Action
ChangeProjectProperty	Action	ChangeQuestObjectiveProperty	Action
ChangeQuestProperty	Action	ChangeQuestStageProperty	Action

Entry	Kind	Entry	Kind
ChangeRegionDescription	Action	ChangeScene	Action
ChangeSceneDescription	Action	ChangeSubtitleLanguage	Action
ChangeVariable	Action	ChangeVerb	Action
ChangeVoiceLanguage	Action	char	Godot global
Character definitions	Designer	CharacterTeleport	Action
CharacterWalkPath	Action	CharacterWalkTo	Action
CharacterWalkWaypoints	Action	Choose	PCA helper
Clamp	PCA helper	clamp	Godot global
clampf	Godot global	clampi	Godot global
ClassDB	Godot class	ClearCharacterWaypoints	Action
CloseInteractionCoin	Action	Color8	Godot global
CompleteObjective	Action	CompleteQuest	Action
ConfigFile	Godot class	Contains	PCA helper
Control	Godot class	convert	Godot global
cos	Godot global	cosh	Godot global
CreatePrimitive	Action	CreateProcedural	Action
CreateVariable	Action	cubic_interpolate	Godot global
cubic_interpolate_angle	Godot global	cubic_interpolate_angle_in_time	Godot global
cubic_interpolate_in_time	Godot global	CurrentScene	PCA helper
Cursors	Designer	CycleInteractionVerb	Action
DateTimeNow	PCA helper	DateTimeString	PCA helper
db_to_linear	Godot global	deg_to_rad	Godot global
DespawnObject	Action	DetachObject	Action
DetachParticleEffect	Action	Dialogue graphs	Designer
dict_to_inst	Godot global	DirAccess	Godot class
DirectoryExists	PCA helper	DisableGUI	Action
DisableGUIElement	Action	Distance	PCA helper
DoWhile	PCA helper	ease	Godot global
EmitEvent	PCA helper	EnableGUI	Action
EnableGUIElement	Action	End	PCA helper
EndCutscene	Action	EndDialogue	Action
EndGame	PCA helper	EndsWith	PCA helper
EndVisualScript	PCA helper	Engine	Godot class
error_string	Godot global	ExecutePreposition	Action
exp	Godot global	FaceCharacter	Action
FaceCoordinates	Action	FaceObject	Action
FadeMusic	Action	FadeScreen	Action
FadeSound	Action	FailObjective	Action
FailQuest	Action	FileAccess	Godot class

Entry	Kind	Entry	Kind
FileExists	PCA helper	floor	Godot global
floorf	Godot global	floori	Godot global
fmod	Godot global	FollowCharacter	Action
Fonts	Designer	ForEach	PCA helper
ForLoop	PCA helper	Format	PCA helper
fposmod	Godot global	FromJSON	PCA helper
get_stack	Godot global	GetAttachPoint	Action
GetAttachPointContacts	Action	GetControlState	Action
GetGUIElementProperty	PCA helper	GetGUIProperty	PCA helper
GetIncludedVariable	PCA helper	GetProjectProperty	PCA helper
GetSetting	PCA helper	GUIs	Designer
hash	Godot global	HasInventory	PCA helper
HasVisited	PCA helper	HideGUI	Action
HideGUIElement	Action	HidePlayerCharacter	Action
HTTPRequest	Godot class	IfCharacterMoving	PCA helper
IfInventoryItemWasUsed	PCA helper	IfPlayerWasHere	PCA helper
IfPlayerWasNeverHere	PCA helper	IfVariable	PCA helper
IncludeScript	PCA helper	Input	Godot class
InputMap	Godot class	inst_to_dict	Godot global
instance_from_id	Godot global	InteractNearest	Action
Inventory definitions	Designer	InventoryCount	PCA helper
inverse_lerp	Godot global	is_equal_approx	Godot global
is_finite	Godot global	is_inf	Godot global
is_instance_id_valid	Godot global	is_instance_of	Godot global
is_instance_valid	Godot global	is_nan	Godot global
is_same	Godot global	is_zero_approx	Godot global
IsCurrentRoom	Action	IsValid	PCA helper
Join	PCA helper	JSON	Godot class
Jump	Action	len	Godot global
Lerp	PCA helper	lerp	Godot global
lerp_angle	Godot global	lerpf	Godot global
linear_to_db	Godot global	LineEdit	Godot class
ListBoxAddItem	Action	ListBoxClearItems	Action
ListBoxDeleteItem	Action	ListBoxModifyItem	Action
load	Godot global	LoadGame	Action
LoadGameSlot	Action	Log	PCA helper
log	Godot global	Lower	PCA helper
Lua script records	Designer	max	Godot global
maxf	Godot global	maxi	Godot global

Entry	Kind	Entry	Kind
MeltSnow	Action	MeshInstance3D	Godot class
min	Godot global	minf	Godot global
mini	Godot global	move_toward	Godot global
MoveNPCToRoom	Action	nearest_po2	Godot global
Node	Godot class	Node3D	Godot class
NowUnix	PCA helper	Object	Godot class
Objectives	Designer	ObjectiveState	PCA helper
Objects and character instances	Designer	on_attach_point_contact	Handler
on_character_control	Handler	on_custom_event	Handler
on_dialogue_ended	Handler	on_dialogue_started	Handler
on_game_start	Handler	on_global_ready	Handler
on_interaction	Handler	on_inventory_changed	Handler
on_item_found	Handler	on_music_end	Handler
on_player_changed	Handler	on_save_loaded	Handler
on_scene_leaving	Handler	on_scene_loaded	Handler
on_trigger_entered	Handler	on_trigger_exited	Handler
on_video_end	Handler	OnAttachPointContact	Event
OnCharacterControl	Event	OnCustomEvent	Event
OnDialogueEnded	Event	OnDialogueStarted	Event
OnGameStart	Event	OnGlobalReady	Event
OnInteraction	Event	OnInventoryChanged	Event
OnItemFound	Event	OnMusicEnd	Event
OnPlayerChanged	Event	OnSaveLoaded	Event
OnSceneLeaving	Event	OnSceneLoaded	Event
OnTriggerEntered	Event	OnTriggerExited	Event
OnVideoEnd	Event	OpenInteractionCoin	Action
ord	Godot global	PackedScene	Godot class
PauseAnimation	Action	PauseFMV	Action
PauseMusic	Action	PauseSound	Action
PauseVoice	Action	PCA Script records	Designer
PCAAssert	PCA helper	PhysicsDirectSpaceState3D	Godot class
pingpong	Godot global	PlayAnimation	Action
PlayGameplayRecording	Action	PlayGUIModelAnimation	Action
PlayMusic	Action	PlaySound	Action
PlayVideo	Action	PlayVoice	Action
posmod	Godot global	pow	Godot global
preload	Godot global	print	Godot global
print_debug	Godot global	print_rich	Godot global
print_stack	Godot global	print_verbose	Godot global

Entry	Kind	Entry	Kind
printerr	Godot global	printraw	Godot global
prints	Godot global	printt	Godot global
proxy.AddDestinationPath	Record method	proxy.AddInventory	Record method
proxy.ChangeRoom	Record method	proxy.ClearDestinationPath	Record method
proxy.Exists	Record method	proxy.FaceCharacter	Record method
proxy.FaceCoordinates	Record method	proxy.Get	Record method
proxy.PlayAnimation	Record method	proxy.RemoveInventory	Record method
proxy.Say	Record method	proxy.Set	Record method
proxy.SetEnabled	Record method	proxy.SetVisible	Record method
proxy.StopAnimation	Record method	proxy.Teleport	Record method
proxy.ToDictionary	Record method	proxy.WalkPath	Record method
proxy.walkpath	Record method	proxy.WalkTo	Record method
push_error	Godot global	push_warning	Godot global
Quests	Designer	QuestState	PCA helper
QuitToMenu	PCA helper	rad_to_deg	Godot global
rand_from_seed	Godot global	randf	Godot global
randf_range	Godot global	randfn	Godot global
randi	Godot global	randi_range	Godot global
RandomFloat	PCA helper	RandomInt	PCA helper
RandomInteger	PCA helper	Randomize	PCA helper
randomize	Godot global	RandomNumber	PCA helper
RandomNumberGenerator	Godot class	range	Godot global
ReadJSONFile	PCA helper	ReadTextFile	PCA helper
RegexSearch	PCA helper	remap	Godot global
RemoveAttachPoint	Action	RemoveDialogueChoice	Action
RemoveDialogueNode	Action	RemoveInventory	Action
Replace	PCA helper	ResetDialogueHistory	Action
ResetObjective	Action	ResetQuest	Action
Resource	Godot class	ResourceLoader	Godot class
ResumeAnimation	Action	ResumeFMV	Action
ResumeMusic	Action	ResumeSound	Action

Entry	Kind	Entry	Kind
ResumeVoice	Action	rid_allocate_id	Godot global
rid_from_int64	Godot global	Rooms	Designer
rotate_toward	Godot global	round	Godot global
roundf	Godot global	roundi	Godot global
RunVisualGraph	PCA helper	RunVisualScript	Action
SaveGame	Action	SaveGameSlot	Action
SaveGameWidgetCommand	Action	Say	Action
SceneTree	Godot class	ScreenTransition	Action
ScriptError	PCA helper	seed	Godot global
SeekFMV	Action	SeekMusic	Action
SeekVideo	Action	SeekVoice	Action
Set2DCamera	Action	Set2DEntityDepth	Action
Set2DLayerVisible	Action	Set2DParallax	Action
SetAttachPoint	Action	SetAutoInputSwitching	Action
SetCameraBehavior	Action	SetCameraPriority	Action
SetCameraTargets	Action	SetCharacterActionEnabled	Action
SetControlSetting	Action	SetDialogueChoiceEnabled	Action
SetFlashlightMode	Action	SetGameLanguage	Action
SetGlow	Action	SetGUIElementProperty	PCA helper
SetGUIProperty	PCA helper	SetIncludedVariable	PCA helper
SetInputDevice	Action	SetInteractionApproachEnabled	Action
SetInteractionMode	Action	SetLipSyncEnabled	Action
SetMotion	Action	SetMouseClicksEnabled	Action
SetMousePointerVisible	Action	SetMousePointingEnabled	Action
SetMovementProfile	Action	SetObjectActionEnabled	Action
SetObjectEnabled	Action	SetObjectInteractionApproach	Action
SetObjectVisible	Action	SetPlayerInputEnabled	Action
SetProjectProperty	PCA helper	SetProximityInteractionEnabled	Action
SetQuestHidden	Action	SetQuestStage	Action
SetQuestTracked	Action	SetRetroStyle	Action
SetRoomCameraBounds	Action	SetRoomDimension	Action
SetRuntimeControlsEnabled	Action	SetSaveSlotImage	Action
SetSceneBrightness	Action	SetSceneTint	Action
SetSetting	PCA helper	SetSnowAccumulation	Action
SetTriggerEnabled	Action	SetVariable	Action
SetVerbUsageEnabled	Action	SetVoiceSpeaker	Action
ShaderMaterial	Godot class	ShakeScreen	Action
ShowGUI	Action	ShowGUIElement	Action
ShowMessage	Action	Shuffle	PCA helper

Entry	Kind	Entry	Kind
sign	Godot global	signf	Godot global
signi	Godot global	sin	Godot global
sinh	Godot global	smoothstep	Godot global
snapped	Godot global	snappedf	Godot global
snappedi	Godot global	Spawn points	Designer
SpawnObject	Action	Split	PCA helper
sqrt	Godot global	StandardMaterial3D	Godot class
StartCutscene	Action	StartDialogue	Action
StartFMV	Action	StartGameplayRecording	Action
StartQuest	Action	StartsWith	PCA helper
step_decimals	Godot global	StopAnimation	Action
StopAttachPointWatch	Action	StopCameraOrbit	Action
StopFMV	Action	StopFollowCharacter	Action
StopGameplayPlayback	Action	StopGameplayRecording	Action
StopMusic	Action	StopScreenShake	Action
StopSound	Action	StopStartupSplashAudio	Action
StopVideo	Action	StopVoice	Action
StopZoomBlur	Action	str	Godot global
str_to_var	Godot global	StringLength	PCA helper
StringSplit	PCA helper	SwitchCharacter	Action
tan	Godot global	tanh	Godot global
Thread	Godot class	Time	Godot class
Timer	Godot class	ToggleGlow	Action
ToggleMotion	Action	ToggleRGBString	Action
ToJSON	PCA helper	TransferInventory	Action
Triggers	Designer	Trim	PCA helper
TryCall	PCA helper	Tween	Godot class
type_convert	Godot global	type_exists	Godot global
type_string	Godot global	TypeOf	PCA helper
typeof	Godot global	UnixToDateTime	PCA helper
UnlockHint	Action	Upper	PCA helper
var_to_bytes	Godot global	var_to_bytes_with_objects	Godot global
var_to_str	Godot global	Verbs	Designer
Visual script records	Designer	Warn	PCA helper
WatchAttachPointContacts	Action	weakref	Godot global
WhileLoop	PCA helper	wrap	Godot global
wrapf	Godot global	wrapi	Godot global
WriteJSONFile	PCA helper	WriteTextFile	PCA helper
ZoomBlur	Action		