PCA 3D ADVENTURE STUDIO

PCA Script

Programming in depth

A practical tutorial and complete language reference

From your first instruction to reusable puzzle logic, live interfaces and cross-language libraries.

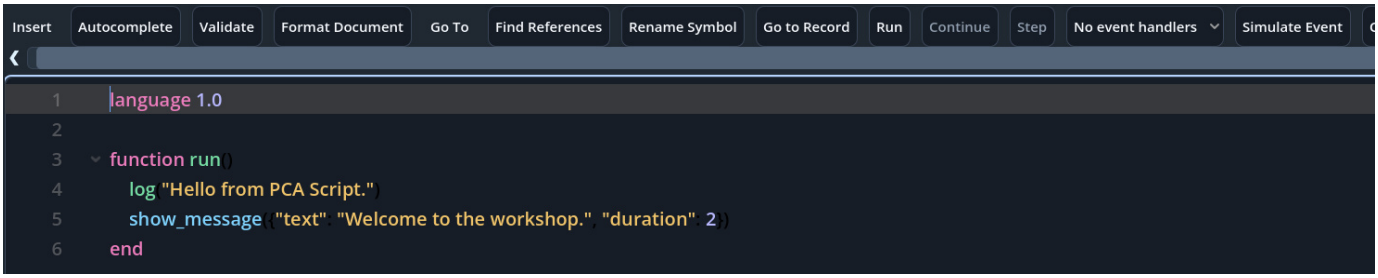

Figure 1. The PCA Script workspace, using a real script in the editor.

PCA Script 1.0 • PCA 3D 0.86.3.4

Windows · macOS · Linux

About the examples and the reference

You can work through this book in an existing test project. There are no downloadable practice projects to install. The workshop shown in the screenshots simply gives the examples a consistent set of names.

Example ID	What it represents
lab / archive	Two rooms
ada / elias	Player and NPC character definitions
archive_door / key_pickup / welcome_trigger	A door, a visible pickup and a trigger in the room
key / chime / wide	An inventory definition, audio asset and camera
hud / name_entry	A gameplay interface and an input panel
utility_pca / rewards_pca	Reusable PCA Script records

Replace those IDs with your project’s actual IDs. The source language version is 1.0; the application version is 0.86.3.4. Those numbers describe different things.

The tutorial code was checked with the engine’s compiler. Numeric, string, array, dictionary, file and control-flow examples were also run, and the input and language bridges were tested in the live runtime. Gameplay reference calls are patterns: you still need real assets, actors and valid project configuration.

> This guide documents the implemented API in this version. It labels current limitations and does not treat requested or planned features as existing language syntax. A shorter spelling in another language is not proof that PCA Script supports it.

Contents

About the examples and the reference 2

1. Start here: what PCA Script is 8

2. Your first script in the editor 9

3. Lines, indentation and comments 10

4. Quotes, escapes and record names 11

5. Values and types 12

6. Variables, constants and saved state 13

7. Assignment and arithmetic 14

8. Comparisons, logic and precedence 15

9. If, else if and else 16

10. Functions and parameters 17

11. Repeat, while and for each 18

12. Break, continue and safe waiting 19

13. Arrays: lists you can build and edit 20

14. Dictionaries and your own records 21

15. Strings and useful text operations 22

16. Regular expressions and text extraction 23

17. Math, vectors and random choices 24

18. JSON and safe file editing 25

19. Dates, time and timed jobs 26

20. Find your game’s records 27

21. Properties: live changes and definitions 28

22. Gameplay calls and readable commands 29

23. A key, a lock and a one-time reward 30

24. Player input through a GUI text box 31

25. Events: respond when the game changes 33

26. Event data and custom signals 34

27. This, the caller and the active player 35

28. Movement, animation and concurrent work 36

29. Errors, assertions and recovery 37

30. Headers, includes and shared state 38

31. Call PCA Script from PCA Script 39

32. Make a function into a visual node 40

33. Work with GDScript, Lua and BASIC 41

34. BASIC, visual graphs and calls back into PCA 42

35. Debugging in the editor 43

36. A complete, reusable interaction pattern 44

37. Speech, portraits and typewriter messages 45

38. Cameras, effects and clean restoration 46

39. Configure the project and authored interfaces 47

40. What “all designer functions” means 48

41. Conversion, organization and testing 49

Built-in functions: how to read this reference 50

Builtins: Values and conversion 51

Builtins: Math and randomness 53

Builtins: Text and JSON 57

Builtins: Validity and gameplay queries 60

Builtins: Interfaces and project settings 65

Builtins: Scripts, events and execution 67

Builtins: Diagnostics 71

Builtins: Calendar and files 72

Builtins: Ranges 75

Gameplay functions: the complete shared catalog 76

Gameplay: Variables, rewards and reusable logic 77

Gameplay: Inventory and two-object interactions 82

Gameplay: Speech, messages, voice and dialogue 86

Gameplay: Characters, movement and animation 100

Gameplay: Objects, descriptions and scene life 116

Gameplay: Cameras, transitions and screen effects 128

Gameplay: Room changes, 2D controls and environmental state 140

Gameplay: Continuous object effects and attachments 154

Gameplay: GUIs, element state and data-driven widgets 169

Gameplay: Control profiles, verbs and pointer behavior 184

Gameplay: Audio, video, FMV and recorded gameplay 193

Gameplay: Quests and objectives 206

Gameplay: Saving, language and project settings 215

Gameplay: Cutscene control and timeline-specific actions 220

Designer access: namespaces and property rules 221

Designer: Characters 222

Designer: Objects 229

Designer: Triggers 239

Designer: Rooms 247

Designer: Inventory 252

Designer: GUIs 254

Designer: Dialogues 257

Designer: Quests 258

Designer: Objectives 259

Designer: Assets 260

Designer: Cameras 261

Designer: Spawn Points 263

Designer: Cursors 264

Designer: Verbs 265

Designer: Fonts 266

Designer: PCA Scripts 267

Designer: Lua Scripts 268

Designer: BASIC Scripts 269

Designer: Visual Scripts 270

Nested effects: a raw-path workaround 271

Custom properties and nested containers 272

Shared character-property action keys 273

Game setting keys 274

GUI element property keys 278

Whole GUI property keys 284

Shared object-property action keys 286

Project property paths 287

Quest-objective action keys 290

Quest-property action keys 291

Quest-stage action keys 292

Events: phrases, shared hooks and payloads 293

Event filters and caller context 300

Readable adventure commands 301

Paths, following and record methods 305

Language quick reference 306

Common mistakes and their fixes 307

Boundaries and compatibility notes 308

Reference basis and version coverage 309

Alphabetical function and event index 310

1. Start here: what PCA Script is

PCA Script is the adventure language built into PCA 3D. It can read your game’s records, change puzzle state, move actors, run interfaces and call the same gameplay actions used by visual scripts. You can write a short instruction such as `say "The door is locked."`, or build a reusable function with parameters and a return value.

This book uses PCA 3D Adventure Studio 0.86.3.4 and PCA Script language 1.0. The screenshots come from the running editor. The explanations apply on Windows, macOS and Linux; the placement of a dock can change with window size and layout.

Read chapters 1-14 in order if programming is new to you. Chapters 15-29 turn the basics into game logic. Chapters 30 onward explain integration, debugging and the complete reference. The reference is meant to be searched while you work, not memorized before you make a door open.

You want to…	Read first
Understand punctuation, values and decisions	Chapters 3-10
Use loops, arrays, text, math or files	Chapters 11-19
Make a playable puzzle or read a text box	Chapters 20-29
Share functions with visual scripts or another language	Chapters 30-34
Find a gameplay function or designer property	The function, action and designer reference sections

Scope: this is the authored PCA Script API in this build. A setting in an editor does not automatically become a callable function. Unsupported facilities are called out, and the reference distinguishes record properties, shared actions and editor-only operations.

2. Your first script in the editor

1. Open a game project. In the Adventure Database, add a PCA Script record and select it. The PCA Script workspace opens. If the database is hidden, restore it through the Panels controls.
2. Give the script an ID such as `hello_pca` and a name such as First Greeting. An ID is the stable name other records use to call it.
3. Replace the source with the example below. Click Validate. A clean result should say there are no errors or warnings.
4. Click Apply Changes. Save the project. Start Test Scene or Play so that a running game exists, then use Run to try the script. A gameplay instruction needs that runtime.

```
language 1.0

function run()
    log("Hello from PCA Script.")
    show_message({"text": "Welcome to the workshop.", "duration": 2})
end
```

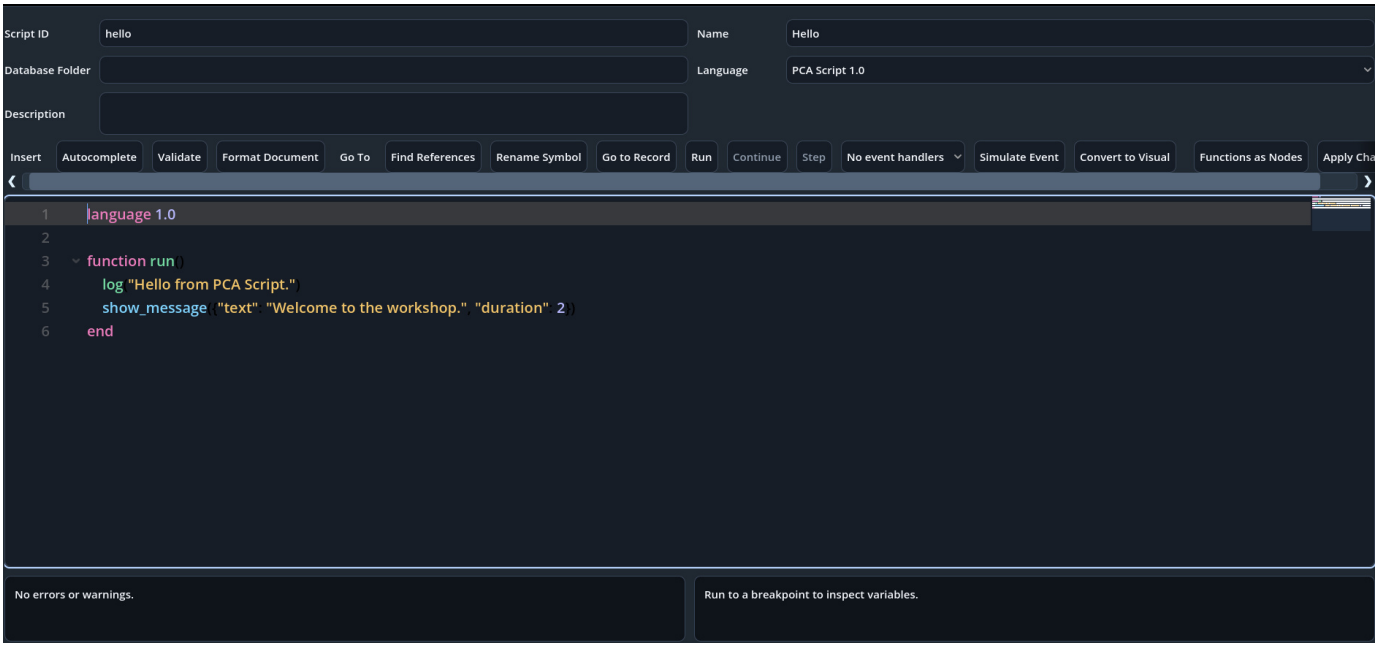

Figure 2. A PCA Script record in the editor, with source, validation results and the script toolbar.

Keep IDs unique across scripting languages. `hello_pca` should not also be the ID of a Lua or visual script. A visible Name (ID) label is convenient, but the ID is what the router needs.

3. Lines, indentation and comments

Write one instruction on each line. A block opens with a word such as `if` or `function` and closes with `end`. Indentation makes the block readable; it does not replace `end`. There are no C-style braces around a block and no semicolons at the ends of instructions.

```
language 1.0
# This comment takes up a whole line.
function run()
    let points = 5  // This comment follows an instruction.
    let address = "https://example.test/#door"
    return points
end
```

Both `#` and `//` begin a comment outside a quoted string. The address above keeps its slashes and hash because they are inside quotes. There is no separate multiline comment delimiter: put a comment marker on each line.

Keep a function call, array or dictionary on one physical line. The parser is line-oriented; spreading its arguments over several lines is not a supported continuation style. A long action is easier to read if you build its options dictionary a field at a time.

```
function run()
    let options = {}
    options["text"] = "The workshop is quiet."
    options["duration"] = 2
    show_message(options)
end
```

Names are case-insensitive when the language looks up variables, functions and records. Text inside quotes is preserved. Use a consistent spelling anyway: `door_open` is much easier to follow than three versions of the same name.

4. Quotes, escapes and record names

Single and double quotes make text values. Choose the outer quote that leaves the sentence easiest to read. Backticks have a different job: they quote a record name or ID that would otherwise contain spaces, punctuation or a reserved word.

```
function run()
    let line1 = "It's locked."
    let line2 = 'The label says "Clockmaker".'
    let line3 = "He said \"come in\"."
    let lines = "First line\nSecond line"
    let tabbed = "Name\tScore"
    let slash = "one\\two"
    return line1
end
```

Escape in source	Stored character
`\n`	New line
`\r`	Carriage return
`\t`	Tab
`\\`	Backslash
`\"`	Double quote
`\'`	Single quote

A string cannot continue through a literal newline in the source. Use
inside it. For a regular expression, the language consumes one level of backslashes: a regex digit group is written `"\d+"` in PCA source.

```
function run()
    let door = objects.`Archive Door`
    return door.name
end
```

These three are different: `"archive_door"` is text containing an ID; `objects.archive_door` is a record reference; a backtick-quoted identifier lets the parser recognize a name. A backtick does not display a speech line.

5. Values and types

Type	Example	Use
integer	`5`	Whole counts, indices, slots
number	`2.5`	Seconds, speeds, measurements
text	`"Archive Key"`	Speech, IDs, entered text
boolean	`true`, `false`	A yes/no puzzle fact
null	`null`	No value / not found
array	`["key", "note"]`	Ordered values
dictionary	`{"open": false}`	Named values
vector2	`vector2(10, 20)`	Two coordinates
vector3	`vector3(2, 0, -1)`	3D coordinates
color	`color("#88bbff")`	A color value
record	`characters.elias`	A live reference to a project record

```
function run()
    let count: integer = 3
    let delay: number = 0.5
    let title: text = "Parts tray"
    let ready: boolean = true
    let items: array = ["key", "note"]
    return type_of(items)
end
```

Type annotations are optional, but useful on public functions. A typed declaration can catch the accidental text `"3"` where you intended the number `3`. Use `str()`, `integer()`, `number()` and `boolean()` when you intend a conversion. Do not expect `boolean("false")` to parse an English answer: compare the entered text with `"true"` or `"yes"` yourself.

6. Variables, constants and saved state

A variable gives a value a name. Use `let` for a working value in the current call. Use `global` for a runtime game variable, such as whether a puzzle is solved. Use `const` for a name that must not be reassigned.

```
global door_open: boolean = false
global score: integer = 0
const KEY_REWARD = 5

function run()
    let message = "The lock clicks."
    if not door_open
        set door_open to true
        add KEY_REWARD to score
        log(message)
    end
    return score
end
```

A global initializer only fills a missing runtime variable. Calling this script a second time does not reset `door_open` to false or `score` to zero. Globals use the same runtime variable store as the Variable Designer, GUI bindings and saved game state.

A top-level `let` is initialized for each invocation; it is not a private persistent module variable in PCA Script. A local `let` inside a function is also call-local. For private session-only shared data, use the included-variable store described later. For story progress that must survive save/load, use a game variable.

Constants prevent rebinding the name. Do not treat a constant array or dictionary as deeply frozen data; keep mutable containers in ordinary variables. Avoid giving a local the same name as a global, a function, `player`, `this` or a namespace.

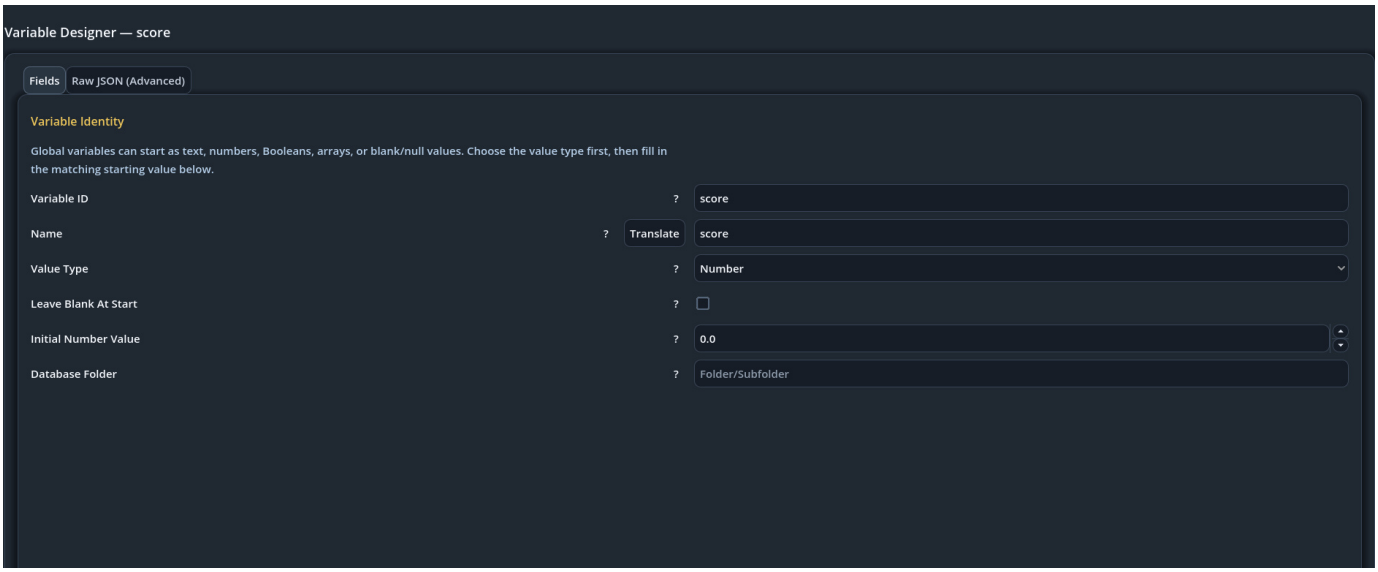

Figure 3. Game variables provide the saved puzzle state that PCA Script globals read and change.

7. Assignment and arithmetic

Meaning	Readable form	Compact form
Assign	`set score to 10`	`score = 10`
Add	`add 5 to score`	`score += 5`
Subtract	`subtract 1 from score`	`score -= 1`
Multiply	Use compact form	`score *= 2`
Divide	Use compact form	`score /= 2`
Reverse a Boolean	`toggle ready`	`ready = not ready`

```
function run()
    let score = 10
    add 5 to score
    score *= 2
    let average = score / 4
    let remainder = 11 % 3
    return [score, average, remainder, -score]
end
```

`+`, `-`, `*`, `/` and `%` mean add, subtract, multiply, divide and integer remainder. Division returns a number with a fractional part when necessary. `%` works with integer values. Avoid a zero divisor. There are no `++`, `--`, exponentiation or bitwise operators in PCA Script 1.0.

`+` also joins text, and joins two arrays into a new array. If either operand is text, the other is converted to text. Add parentheses when a line mixes text and arithmetic: `"Score: " + str(score + bonus)` makes your intention clear.

Assignment uses `=`. Equality uses `==` or `is`. Accidentally writing an assignment where you meant a test is a good reason to validate before you press Run.

8. Comparisons, logic and precedence

Operator	Meaning
`==`, `is`	Equal
`!=`, `is not`	Not equal
`<`, `is less than`	Less than
`<=`, `is at most`	Less than or equal
`>`, `is greater than`	Greater than
`>=`, `is at least`	Greater than or equal
`and`	Both tests true; stops early on false
`or`	Either test true; stops early on true
`not`	Reverse a test
`player has key`	Inventory possession, not array membership
`objects.archive_door exists`	The referenced record exists

```
function run()
    let power = true
    let code_entered = false
    let override = true
    let allowed = power and (code_entered or override)
    return allowed
end
```

From weaker to stronger: `or`; `and`; comparisons and `has`; `+ -`; `* / %`; unary `not + -`; property, index and call expressions. Parentheses are cheap insurance. `not a == b` is easier to understand when written `not (a == b)`.

Use `contains(text, needle)` for text searches. The ordinary PCA parser does not provide Python's `in` expression, a general array `contains` operator, or C-style `&&` and `||`. To test whether an array contains an item, loop through it and compare each value.

9. If, else if and else

```
function describe_charge(charge: number): text
    if charge >= 80
        return "Fully charged."
    else if charge >= 30
        return "Enough for one attempt."
    else
        return "The battery is flat."
    end
end

function run()
    return describe_charge(45)
end
```

The first true branch runs. The remaining branches are skipped. `else` covers everything left over. One `end` closes the entire decision; a second closes the function.

Put specific conditions before broad ones. If you test `charge >= 30` first, a charge of 90 satisfies it and never reaches a later `>= 80` branch. You can nest another `if` inside a branch, but a few clear returns often make a puzzle script easier to maintain.

```
global powered = false
function can_open(): boolean
    if not powered
        return false
    end
    return has_inventory("key")
end
```

`return` ends the current function. A false return value is still just data: a caller has to inspect it. Returning false does not automatically cancel an unrelated action list. There is no `switch` statement or ternary operator in PCA Script 1.0; use an `else if` chain or a dictionary lookup.

10. Functions and parameters

```
function total_price(price: number, count: integer = 1): number
    return price * count
end

function run()
    let one = total_price(2.5)
    let three = total_price(2.5, 3)
    return [one, three]
end
```

A function packages a job under a name. Parameters are inputs. The returned value is its output. A default value makes the last parameter optional. Calls to ordinary functions use positional arguments in the declared order.

Declare functions at the top level, not inside another function. Use `function run()` as the normal entry point when the record is called directly. A script can also contain top-level instructions, but when `run` exists the default call selects that function. Keep initialization in declarations and the actual job in `run` to avoid ambiguity.

A function can call another function, including itself, but the runtime limits nested calls to 64. A loop is usually better than recursion for walking a list of inventory items. Type annotations describe supported value types; they do not turn a function into an exposed visual node by themselves.

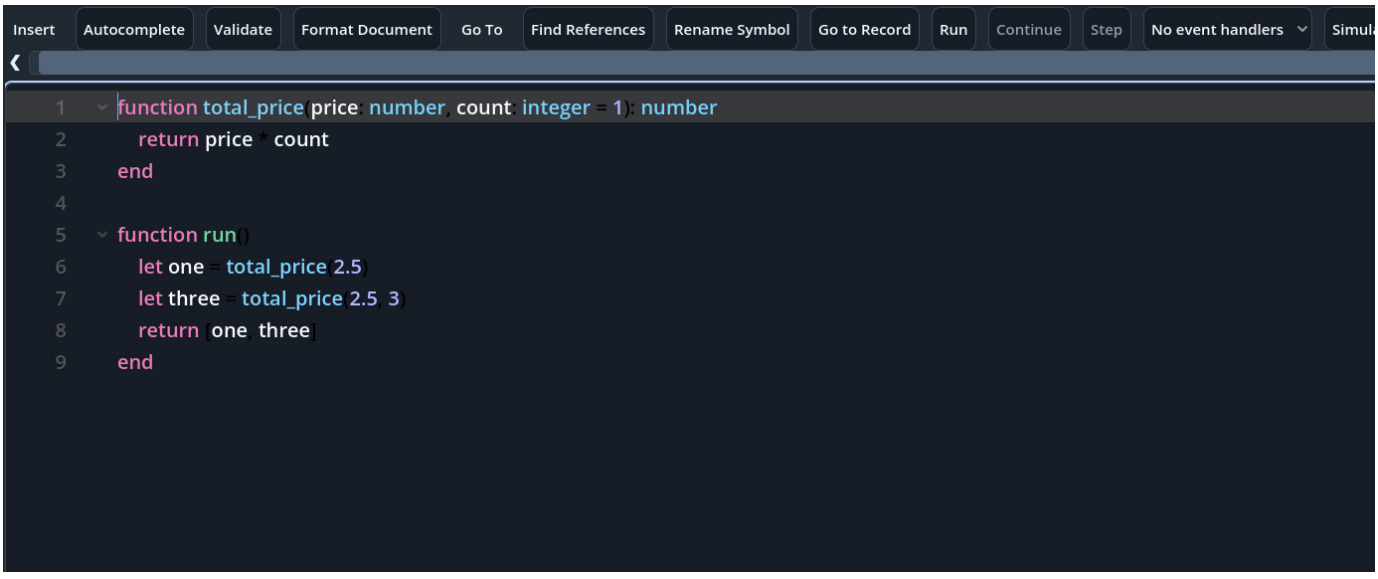

Figure 4. Small named functions keep puzzle checks separate from the response that uses them.

11. Repeat, while and for each

```
function run()
    let total = 0
    repeat 3 times
        total += 2
    end
    let tries = 0
    while tries < 3
        tries += 1
    end
    for each value, index in [10, 20, 30]
        total += value
    end
    return [total, tries]
end
```

`repeat` is for a fixed count. `while` keeps going while a test is true. `for each` visits every value in an array; the optional second name receives its zero-based index. When given a dictionary, `for each` visits its keys.

A numeric for-loop is written with `range()`: `for each n in range(1, 6)` produces 1, 2, 3, 4 and 5. The stop value is excluded. `range_inclusive(1, 5)` includes 5. A negative step counts down. A zero step is an error.

```
function run()
    let result = []
    for each n in range(5, 0, -2)
        result = result + [n]
    end
    return result
end
```

Ordinary loops have a 100,000-iteration guard. They do not automatically spread work over frames. A loop that waits for player action needs a `wait` in its body; otherwise it can freeze the current frame until the guard stops it.

12. Break, continue and safe waiting

```
function run()
    let found = []
    for each n in range(0, 10)
        if n == 2
            continue
        end
        if n == 5
            break
        end
        found = found + [n]
    end
    return found
end
```

`continue` skips the remaining instructions in the current iteration. `break` leaves the nearest loop. `return` leaves the function, even when it is inside several loops.

```
global panel_open = false
function run()
    let attempts = 0
    while not panel_open and attempts < 50
        wait 0.1 seconds
        attempts += 1
    end
    return panel_open
end
```

This waits for up to roughly five seconds without busy-looping. `wait until panel_open` is shorter, but can wait indefinitely if nothing sets the flag. Use a bounded loop when failure to arrive needs a useful response. `wait` uses seconds; there is no need to write GDScript's `await` around PCA instructions.

13. Arrays: lists you can build and edit

```
function run()
    let clues = ["key", "note"]
    clues = clues + ["watch"]
    clues[1] = "torn_note"
    let count = len(clues)
    return [clues[0], clues[1], count]
end
```

An array keeps values in order. Its first index is 0 and its last is `len(array) - 1`. Check the length before reading an index. Do not rely on a missing index returning a useful value.

```
function without(values: array, unwanted): array
    let kept = []
    for each value in values
        if value != unwanted
            kept = kept + [value]
        end
    end
    return kept
end

function run()
    return without(["key", "note", "watch"], "note")
end
```

There is no general `.append()`, `.remove()` or `.sort()` method API on PCA arrays. Concatenate another array to append; rebuild the list to filter it; use a loop for a search. `shuffle()` returns a shuffled copy. For a designer-owned array, edit a working value and write the whole array back, rather than assuming that a nested element write will notify the live runtime.

14. Dictionaries and your own records

```
record Reward
    item: text
    points: integer
end

function run()
    let reward = Reward(item: "key", points: 5)
    let facts = {"unlocked": false, "attempts": 0}
    facts["attempts"] += 1
    facts["unlocked"] = true
    return [reward["points"], facts["attempts"]]
end
```

A dictionary stores values under keys. Use quoted text keys, and use brackets to read a key held in another variable: `facts[key_name]`. A simple named field can also use dot access. For a missing dictionary key, the value is null; check before doing arithmetic with it.

A `record` declaration defines the fields of a data value. The named-argument constructor above produces a dictionary with those fields. It does not create a character, import an asset or register a new designer object. A project record reference, such as `objects.archive_door`, is a different kind of value.

Arrays and dictionaries are mutable containers. For an independent deep copy of JSON-compatible data, `from_json(to_json(value))` is a useful technique. Vectors, colors and live record references are not a promise of lossless JSON round-tripping.

15. Strings and useful text operations

```
function run()
    let answer = lower(trim("  Open The Door  "))
    let words = split(answer, " ", false)
    let tidy = join(words, " ")
    let changed = replace(tidy, "door", "archive")
    return [answer, changed, starts_with(answer, "open")]
end
```

`trim()` removes whitespace at the ends. `lower()` and `upper()` change case. `contains()`, `starts_with()` and `ends_with()` accept an optional case-sensitive flag; it defaults to true. `replace()` replaces matching text with case-sensitive matching.

`split(text, delimiter, allow_empty)` makes an array. Its default delimiter is a space, and empty pieces are kept unless you pass false. `join(array, delimiter)` goes the other way. `len(text)`, `length(text)` and `string_length(text)` report length.

```
function run()
    let line = format("{name} found {count} gears.", {"name": "Ada", "count": 3})
    let numbered = format("{0}: {1}", ["Score", 25])
    return [line, numbered]
end
```

`format()` replaces braces with dictionary keys, zero-based array positions, or `{value}` for a single value. It is not a printf-style format engine. GUI wildcards such as `$$score` belong to the GUI text substitution system; ordinary PCA string values do not automatically expand them.

16. Regular expressions and text extraction

```
function run()
    let found = regex_search("([A-Z]+)-(\\d+)", "KEY-204")
    if found == null
        return "No code found"
    end
    return found["groups"][2]
end
```

The result of a successful single search is a dictionary with `text`, `start`, `end` and `groups`. Group 0 is the complete match; group 1 is the first parenthesized capture. Start is inclusive and end is exclusive. A failed single search returns null.

Pass true as the third argument to get an array of all matches. A failed all-matches search returns an empty array. A malformed pattern reports a diagnostic and also returns a no-match-style result; check your pattern rather than treating every empty result as valid input.

PCA Script has no general substring or string-method API. Use split/replace for straightforward transformations and a captured regex group when you need a portion of text. Avoid writing a complicated regular expression for a simple yes/no answer.

17. Math, vectors and random choices

```
function run()
    let health = clamp(135, 0, 100)
    let halfway = lerp(10, 20, 0.5)
    let span = distance(vector3(0, 0, 0), vector3(3, 0, 4))
    return [health, halfway, span, abs(-3), round(2.6)]
end
```

`min()` and `max()` take separate numeric arguments, not an array to reduce. `floor()` rounds down; `ceil()` rounds up; `round()` chooses the nearest integer value. `clamp(value, low, high)` limits a value. `lerp(start, finish, weight)` interpolates numbers; 0 is the start and 1 the finish.

```
function run()
    randomize(1234)
    let die = random_int(1, 6)
    let amount = random_number(0, 1)
    let reply = choose(["Locked.", "Still locked."])
    let order = shuffle(["red", "green", "blue"])
    return len(order)
end
```

Integer random endpoints are inclusive. Use no seed for normal unpredictable play; a fixed seed makes a test sequence repeatable. The random generator is shared, so seeding it can affect other random choices. `shuffle()` makes a copy. `choose([], fallback)` is useful when a list may be empty.

`distance()` accepts two Vector2 values, two Vector3 values, or two numbers. It does not find a character’s position from a record reference. Read its position first. Advanced trigonometry, square roots and other native math APIs require a helper in another language; do not assume every Godot math function is a PCA builtin.

18. JSON and safe file editing

```
function run()
    let path = "user://tutorial/preferences.json"
    let prefs = read_json_file(path, {"music": 0.7})
    prefs["music"] = 0.5
    let saved = write_json_file(path, prefs, "  ")
    if not saved
        warn("Preferences could not be saved.")
    end
    return saved
end
```

`user://` is the game's writable user-data area. It is a logical path, so you do not need a Windows drive letter or a Linux home path. The write helpers create parent directories and overwrite the complete target file. They do not open an interactive file editor.

To edit text, read it with `read_text_file()`, change the string, then write the complete new string with `write_text_file()`. There is no append mode, line insertion, rename or delete builtin. Read/write JSON helpers suit settings, custom notes or exported progress; use the save-game actions for a full game save.

```
function run()
    let path = "user://tutorial/notes.txt"
    let old = read_text_file(path, "")
    return write_text_file(path, old + "Visited the archive.\n")
end
```

Writing is restricted to `user://` paths. Read helpers can also read packaged resources when a valid resource path is available, but your source project's absolute paths are not portable. Always check the returned Boolean before telling the player that a file was saved.

19. Dates, time and timed jobs

Use `wait` for a pause in a sequence. Use a small waiting loop for a countdown. Use `every` for a detached repeated job. Date/time helpers are for the computer's calendar and clock, not a game-loop stopwatch.

```
global countdown_running = false
global seconds_left = 0
function run()
    if countdown_running
        return
    end
    countdown_running = true
    seconds_left = 10
    while countdown_running and seconds_left > 0
        wait 1 second
        if countdown_running
            seconds_left -= 1
        end
    end
    countdown_running = false
end
```

Set `countdown_running` to false from another script to stop future ticks. A waiting loop counts its requested waits and can drift slightly under load. A saved countdown should save a deliberate game value and have a clear restart policy after load.

```
global pulse_active = true
function run()
    every 1 second
        if not pulse_active
            break
        end
        log("tick")
    end
end
```

An `every` block starts a separate periodic job. Its body runs once immediately, then waits the interval before the next pass. It does not hold up the following instruction in `run`. There is no named timer handle builtin. Guard against starting duplicates, and stop or deactivate the job when its owner no longer needs it. Script/cache invalidation cancels its execution generation.

`now_unix()` returns whole Unix seconds. `date_time_now()` returns a calendar dictionary. `date_time_string()` returns a display string. A system-clock adjustment can move wall-clock time, so avoid using it as an unquestioned physics timer.

20. Find your game’s records

A namespace groups designer records. `objects.archive_door` is an object; `characters.elias` is a character definition; `rooms.lab` is a room. A record reference lets the runtime read a property or route a supported change. It is not a raw Godot Node.

```
function run()
    let door = objects.archive_door
    if door exists
        log(door.name)
    end
    let room_id = current_scene()
    let who = active_player()
    return [room_id, who]
end
```

A string ID is usually the safest argument for action dictionaries. Direct references are useful for properties and the small record-method API. The compiler binds recognized project names to IDs; use autocomplete to avoid misspellings.

For names that are not legal simple identifiers, completion can insert backticks. For a computed ID, use the action/helper’s string parameter instead of trying to construct a dotted source expression. Check `exists()` before using an optional record.

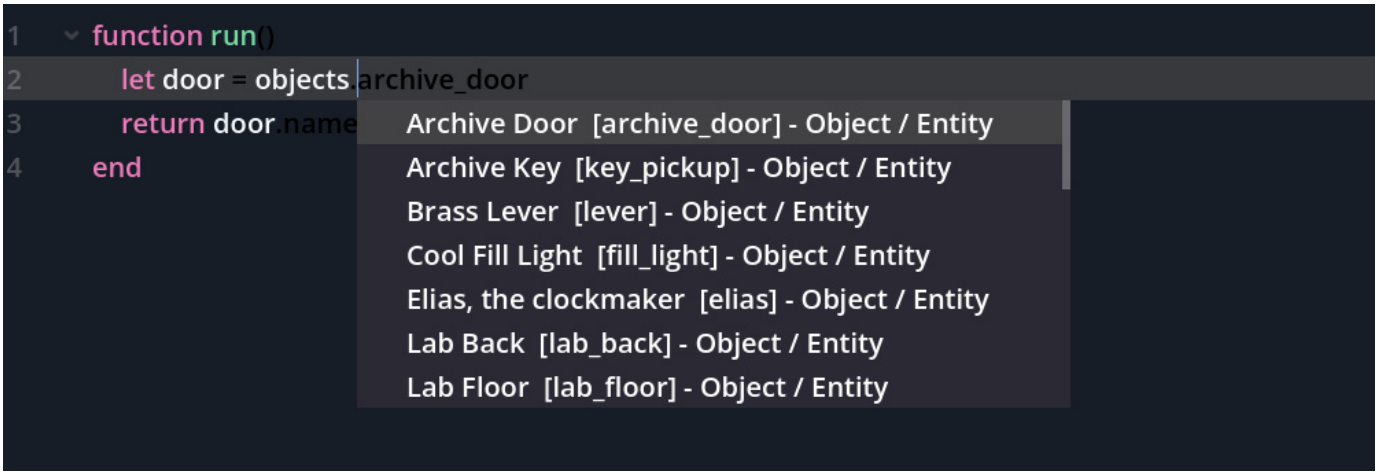

Figure 5. The editor’s completion list helps find actual records and supported property paths.

There are 19 direct namespaces in this build. Project settings, runtime variables and cutscenes use helpers/actions rather than a `Project`, `Variable` or `Cutscene` PCA namespace. Other scripting languages expose a different surface; their examples cannot be pasted into PCA unchanged.

21. Properties: live changes and definitions

```
function run()
    set objects.archive_door.visible to true
    set objects.archive_door.opacity to 0.8
    set objects.archive_door.transform.position to vector3(2, 0, -3)
    set characters.elias.name to "Professor Elias"
end
```

A definition is the stored description of a character or asset. A loaded scene object is the instance the player sees now. A live property can update that instance. A definition-only property may be saved in the runtime project data but need an object rebuild or a later lookup before its visible effect changes.

The reference marks properties as Live or Definition. “Definition” does not mean your source game project has been permanently rewritten on disk. Runtime record changes and full save-game persistence are separate concerns. Use game variables for durable puzzle facts and reapply visual state from those facts when a room loads.

Do not rename a record by assigning its `id`. ID and type fields are read-only where marked. Change an ID through the editor so it can update references. To edit a nested array such as GUI elements, dialogue nodes or animation states, change a working copy and assign the full container, or use a dedicated action.

Character definitions and scene actors can share an ID but are not interchangeable. `characters.elias.name` changes the shared character definition; `objects.elias.character.displayName` addresses a loaded actor’s character data when that actor exists in the scene.

22. Gameplay calls and readable commands

PCA Script offers two useful spellings for many jobs. Readable commands are quick for common adventure work. The shared action functions cover the full catalog and put their settings in one dictionary. Both routes reach the gameplay runtime.

```
function run()
    say "The mechanism is silent."
    give key to player
    hide object key_pickup
    play sound chime
    set camera to wide
end
```

```
function run()
    add_inventory({"itemId": "key", "count": 1})
    set_object_visible({"entityId": "key_pickup", "visible": false})
    play_sound({"assetId": "chime", "channel": "puzzle"})
    action("CameraShake", {"duration": 0.3, "power": 0.1})
end
```

Function names use snake_case. Dictionary field names keep the catalog’s spelling: `characterId`, not `character_id`; `assetId`, not `asset`. A typo inside a dictionary can silently leave the action using a default. Copy the field key from the reference.

A direct action call normally waits for the action routine to return. That does not mean the visible effect has ended. Playing an animation or music can return after playback starts. A camera blend or fade may wait, and some actions expose a `wait` option. Each reference entry includes the timing rule.

Use a quoted ID such as `"archive_door"` in an options dictionary. An omitted dictionary uses the catalog defaults; it does not discover the object you meant.

23. A key, a lock and a one-time reward

This is the basic adventure pattern: reject an already-completed attempt, check the requirement, set the saved fact, then present the result. Set the fact before any long speech or movement so a second click cannot award the same result twice.

```
global door_open = false
global score = 0

function run()
    if door_open
        say "It is already unlocked."
        return
    end
    if not has_inventory("key")
        say "I need the archive key."
        return
    end
    door_open = true
    score += 5
    play_sound({"assetId": "chime", "channel": "puzzle"})
    show_message({"text": "The archive is unlocked.", "duration": 2})
end
```

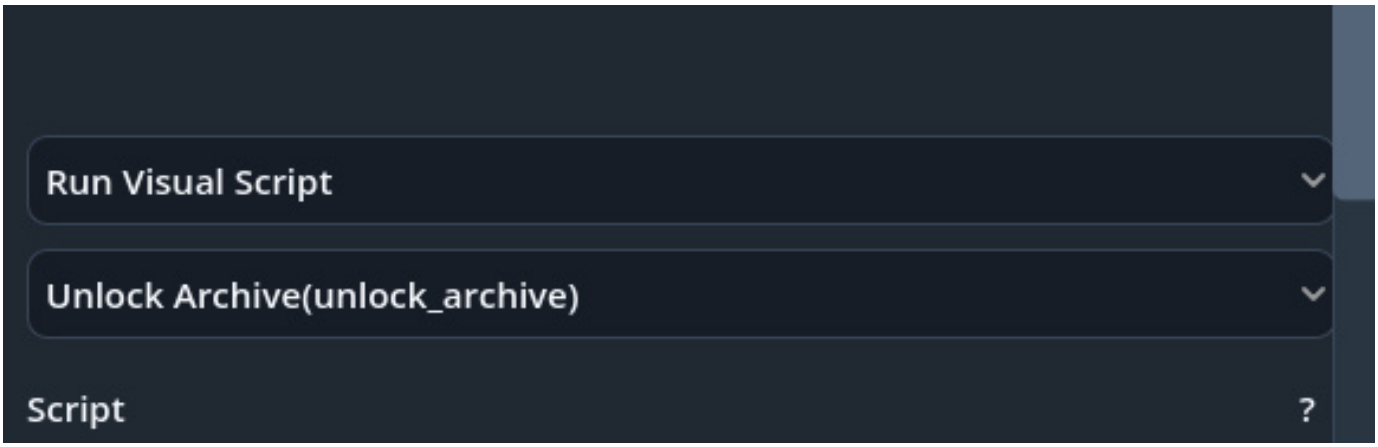

Figure 6. A script belongs behind the door’s Use response; the object still supplies the hotspot, verb and approach settings.

1. Create the `key` inventory definition. Import the chime as an audio asset if you want the sound.
2. Create a Boolean variable `door_open`, initially false, and a numeric `score`, initially 0.
3. Create a PCA Script with ID `unlock_archive` and this source. Validate and apply it.
4. On the door’s Use response, add Run Script / Run Visual Script and choose `unlock_archive`. Test with and without the key.
5. Make room-entry logic restore the open-door appearance from `door_open`. The saved fact and the displayed model must agree.

24. Player input through a GUI text box

PCA Script does not have a terminal-style `input()` or blocking prompt builtin. Make a GUI with a text box and a Confirm button. The button runs a script, and the script reads the live text from the element.

```
global player_name = "Ada"
function run()
    let typed = get_gui_element_property("name_entry", "name_box", "text", "")
    typed = trim(typed)
    if len(typed) < 1
        set_gui_element_property("name_entry", "feedback", "text", "Enter a name.")
        return false
    end
    player_name = typed
    set_gui_element_property("name_entry", "feedback", "text", "Hello, " + typed + ".")
    return true
end
```

1. Create a GUI with ID `name_entry`. Add a single-line Text Box with ID `name_box`, a Label `feedback`, and a Button `confirm`.
2. Create this script as `accept_name`. On Confirm → Events → Click, choose Run Visual Script / Run Script behavior and set Target to `accept_name`.
3. Leave the event's Verb empty. Validate/apply the script and save the GUI. Show the GUI in a running game.
4. Type a name, then click Confirm. `text` reads the live control, not just the text originally stored in the designer.

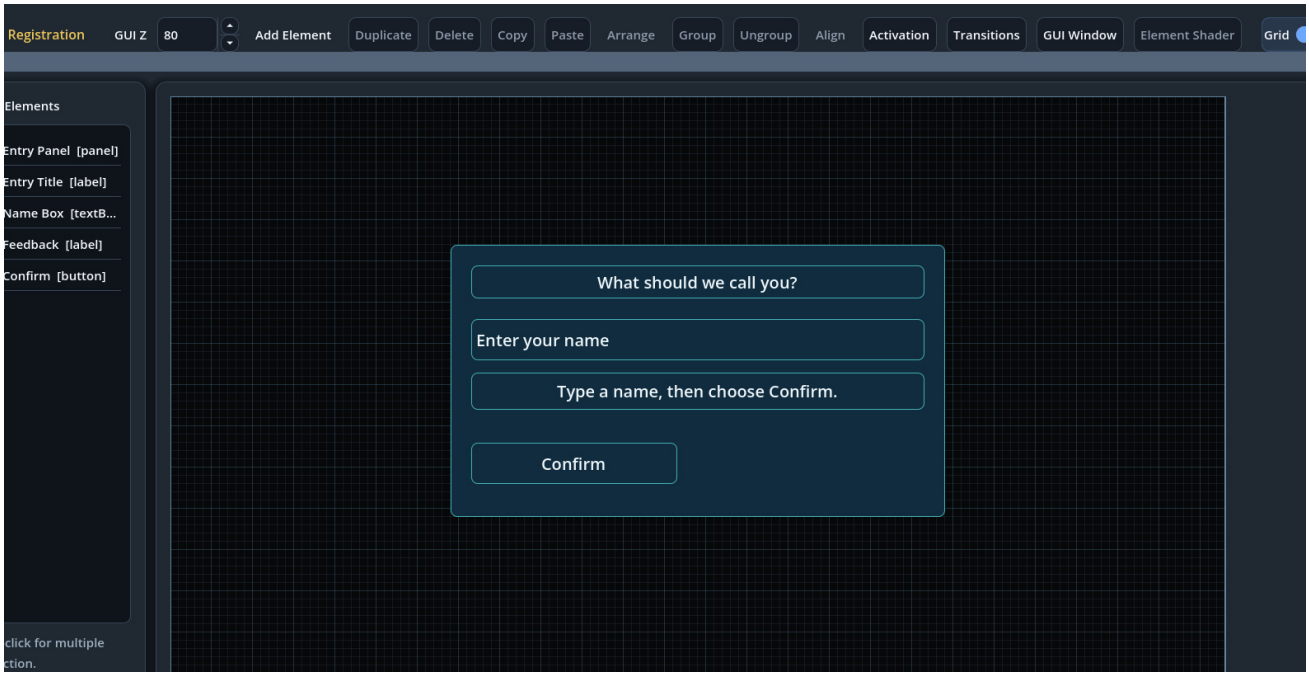

Figure 7. The input panel in the GUI Designer. The text box collects input and the button calls the PCA script.

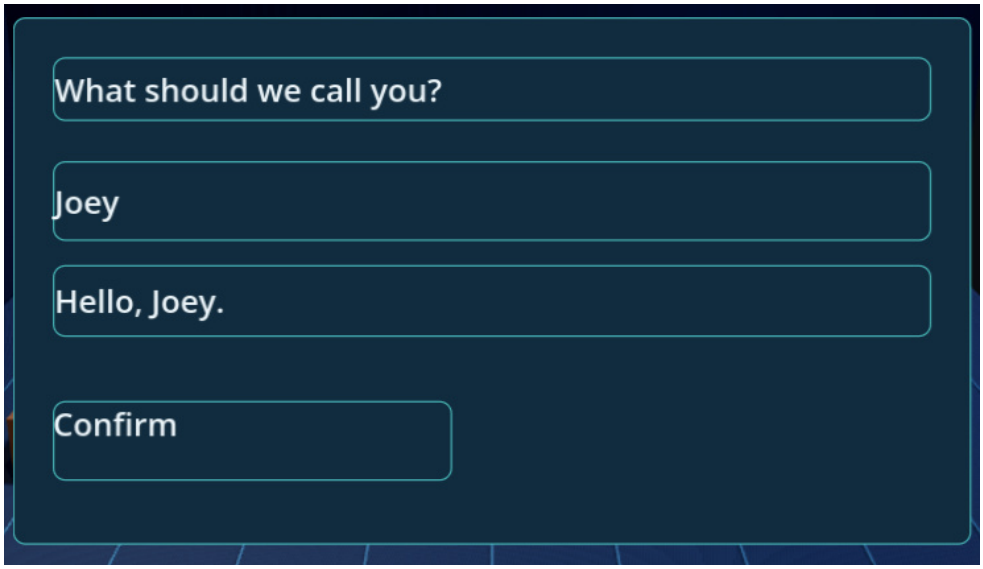

Figure 8. A real runtime test: Confirm reads the typed name and updates the feedback label.

Checkboxes and sliders can bind to game variables through their type settings. Text parser commands are another input route: they call the configured script with command context. Use `log(to_json(event))` in a suitable event handler to inspect the payload rather than guessing its keys.

25. Events: respond when the game changes

```
global visits = 0

when game starts
    log("A new play session has started.")
end

when room lab loads
    visits += 1
    log("Room visits: " + str(visits))
end

when player enters trigger welcome_trigger
    log("The player crossed the welcome area.")
end
```

A `when` block is an event handler. The runtime finds matching handlers in enabled PCA Script records and supplies the event context. You do not need to call `run()` to make a normal `when room lab loads` handler listen. Be careful about also calling the same work from a room action list; that can run it twice.

Built-in event phrases can filter a room, trigger, character, dialogue or media asset. The full event reference gives the exact phrase and payload. A handler runs for the event actually emitted; inventing an English phrase does not create an engine hook.

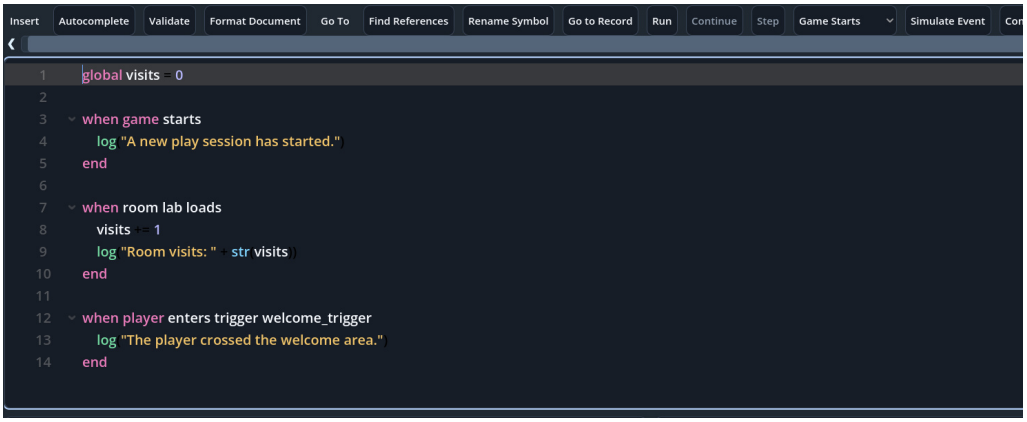

Figure 9. PCA event handlers are declared in the source. Simulate Event can help test a handler without repeating a full playthrough.

A plain function named `on_scene_loaded` is not the normal PCA event declaration. Use `when room lab loads`, or the exact-name event form in the reference. Native GDScript/Lua/BASIC subscription controls have their own conventions.

26. Event data and custom signals

```
when event signal_powered
    log("Source: " + str(event["source"]))
end

function run()
    emit_event("signal_powered", {"source": "lever"})
end
```

`emit_event()` broadcasts the exact name passed to it and includes your data in the event context. The matching listener uses `when event signal_powered`. Use a name that describes what happened. This keeps the lever from having to know every lamp, door and journal entry that might care about power.

Within a dispatched handler, `event` is a dictionary snapshot of the context. Standard keys depend on the emitter: an interaction may include `entityId`, `verb` and `itemId`; a trigger includes its trigger ID. Check a missing value against null before you use it.

An exact shared hook can be written as `when event OnInteraction`. Filter its context in the body when the readable header has no suitable filter. Custom data should be a dictionary. A non-dictionary data value is placed under `value`.

> Do not emit the same event from inside its own matching handler without a guard. A notification system can recurse just as effectively as a function can. Set one-time state before broadcasting and avoid a circular chain of listeners.

27. This, the caller and the active player

```
function run()
    if this == null
        warn("This script needs an object caller.")
        return false
    end
    log("Called by " + str(this.id))
    return true
end
```

`this` means the object or record that called the script, when the call supplied an owner. It does not mean whichever object is selected in the editor. A direct Run button or global event can have no owner, so check for null in reusable code.

`player` resolves the active player. A function called by a door can therefore use `this` for the door and `player` for the actor standing in front of it. Capture an ID before a delayed operation if later player switching would change what you intend to address.

A shared GUI element action may also use special caller IDs such as `this` in the action's own fields. That is context routing, not proof that a GUI element is a direct PCA namespace record. Use `guiId` and `elementId` from the event/context when you need a particular element.

A nested call normally inherits caller context through the router. If you supply a new context dictionary, preserve the keys your helper needs. The phrase "this script" is not the same as the language value `this`.

28. Movement, animation and concurrent work

```
function run()
    player walks to archive_door
    player faces archive_door
    play player animation "nod"
    wait until player animation ends
end
```

Walking uses navigation and approach settings; moving/teleporting is a different operation. An imported animation name must match a clip or configured state. Starting playback is not the same as waiting for it to finish. A looping idle clip never naturally ends, so the last wait above is suitable only for a finite clip.

```
function pan_notice()
    camera_shake({"duration": 0.3, "power": 0.1})
    log("Camera beat finished.")
end

function run()
    parallel
        pan_notice()
        show_message({"text": "The machine wakes.", "duration": 2})
    end
    log("Both branches finished.")
end
```

Each immediate instruction in `parallel` is a branch. They start together, and the enclosing block waits for all branches. Wrap a sequence in a function when one branch needs several ordered steps. Parallel branches share local containers; avoid having both branches write the same state.

Readable `start` commands offer non-blocking versions of supported adventure actions. They are not a universal “make any function asynchronous” keyword. For staged camera, speech and animation sequences with many timings, the Cutscene Designer remains useful.

29. Errors, assertions and recovery

```
function run()
    try
        let result = 10 / 0
        return result
    catch problem
        log(problem["message"])
        return "Recovered"
    end
end
```

A runtime error stops the failing frame unless a surrounding `try` catches it. The caught dictionary includes `message`, `line` and `scriptId`. Use recovery for an expected failure, not to hide a typo in every line.

A shared action result with `ok: false` is promoted to a PCA runtime failure and can reach a catch block. An ordinary false or null return is not automatically an exception. Check the documented result and, where relevant, confirm the resulting game state.

`assert condition` is a statement that can fail the script. `pca_assert(condition, message)` is a diagnostic helper that reports and returns a Boolean; it does not have the same stop behavior. `script_error(text)` logs an error message; it is not a throw statement.

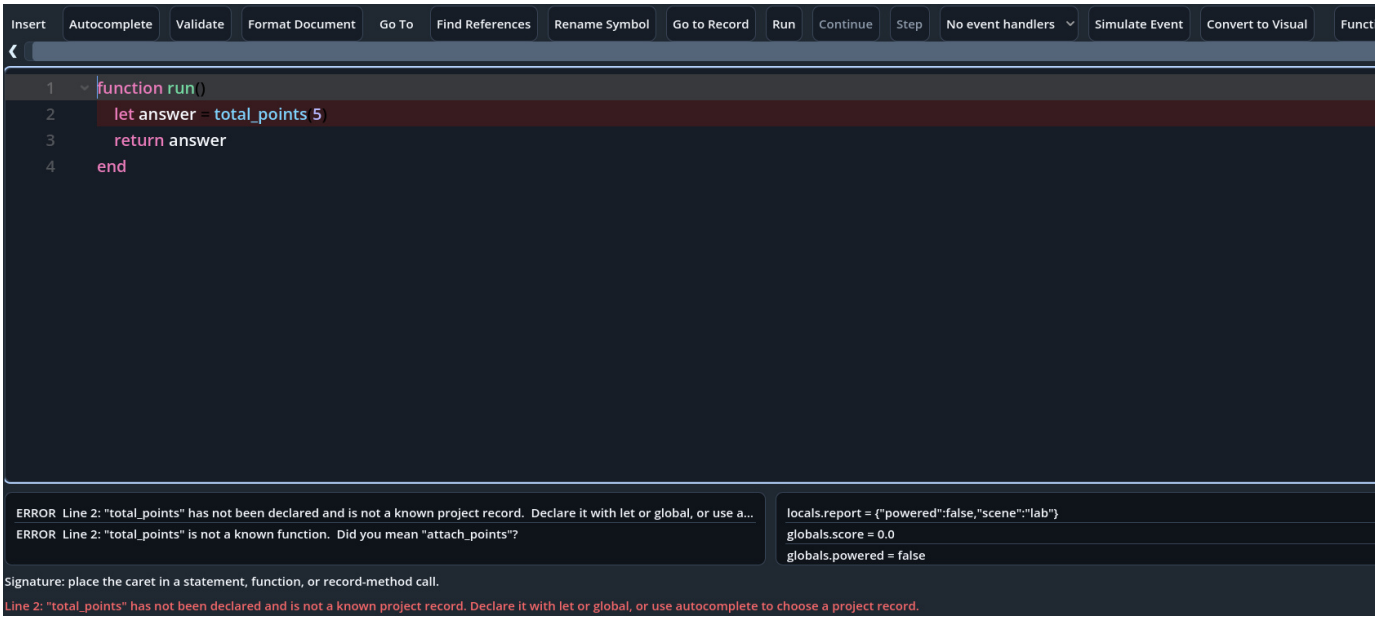

Figure 10. Validate reports a source line and a useful explanation before you run the script.

`try` cannot repair a syntax error: invalid source never becomes a runnable program. Fix validation errors first, then use runtime logs or breakpoints to investigate behavior.

30. Headers, includes and shared state

The only language header is `language 1.0`. PCA Script does not have C-style header files, `#include`, textual macros or an import statement that pastes another file into this one. A leading `#include` would be a comment.

```
function run()
    let library = include_script("utility_pca")
    let total = call_script(library, "add_points", [10, 5])
    set_included_variable(library, "last_total", total)
    return get_included_variable(library, "last_total", 0)
end
```

With one argument, `include_script()` checks that a script ID exists and returns that ID. It does not run the other script's top-level code or import its functions into your local scope. With additional arguments it uses the same call route as `call_script()`.

```
function add_points(base: integer, bonus: integer): integer
    return base + bonus
end

function run()
    return add_points(10, 5)
end
```

Put the second example in a separate PCA Script called `utility_pca`. The included-variable helpers use a router-owned dictionary keyed by script ID and variable name. They do not read the other script's local `let` declarations. This store is session/cache state, not a saved-game variable system.

A header can describe a public function's parameters in comments, but its executable definition still belongs in a script record. Use the shared router for libraries and the Variable Designer for saved story state.

31. Call PCA Script from PCA Script

```
function run()
    let value = call_script("utility_pca", "add_points", [10, 5])
    log("Total: " + str(value))
    return value
end
```

The argument array is positional: its first value goes to the first parameter. The optional fourth argument is a context dictionary. Use it for caller information or a reason code, not as a substitute for declared parameters.

Keep the called function small and explicit. Return a number, text, Boolean, array or dictionary that the caller understands. If it changes global puzzle state, say so in a comment at the top of the function.

To run a visual script record by ID, use the readable `run visual script` statement or the shared script action. `run_visual_graph()` instead expects graph data as a dictionary; it is the lower-level route, not an ID lookup.

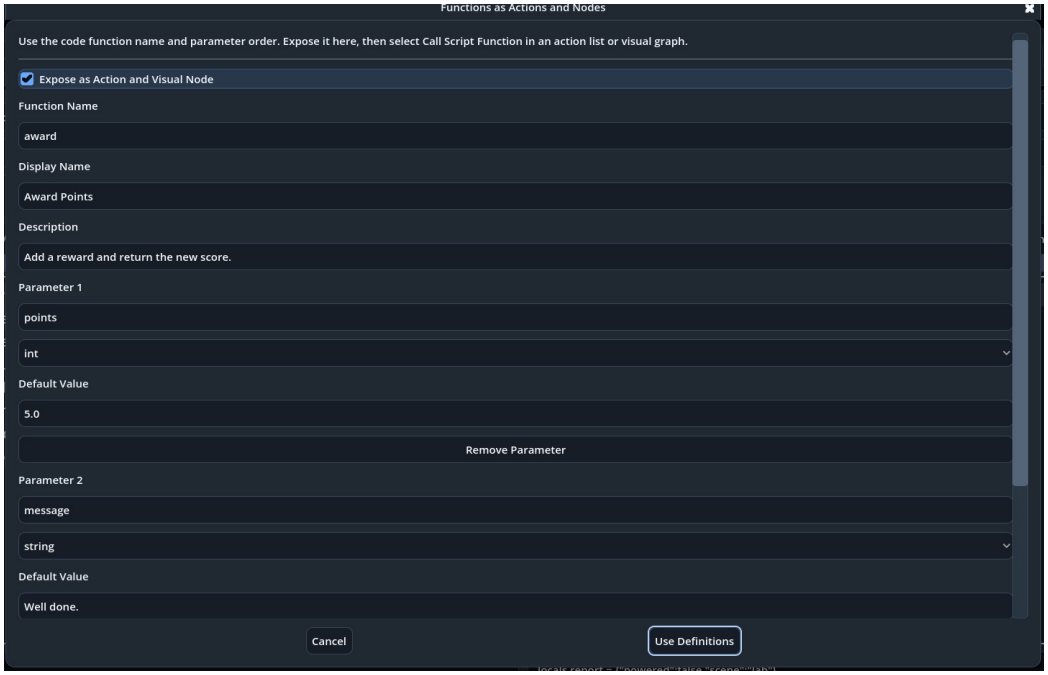

Figure 11. Functions as Nodes describes the public parameter contract that the visual editor will show.

32. Make a function into a visual node

```
global score = 0
function award(points: integer, message: text = "Well done."): integer
    score += points
    log(message)
    return score
end
```

1. Create a PCA Script with ID `rewards_pca`, enter this function, Validate and Apply Changes.
2. Click Functions as Nodes. Add/select the `award` definition. Its function name must match the source exactly in meaning, and its parameters must match the source order.
3. Add the parameters manually in source order: points as int, then message as string. Enter defaults of 5 and Well done. The definition UI uses int/string where the PCA source uses integer/text. Use Definitions accepts this metadata; it does not scan the source, create the function or run it. Then Apply Changes to save the script.
4. In a visual script, add Call Script Function. Choose the published script/function and fill in points and message. Choose a result variable if you need the returned score.
5. The same published function can be used from an action list that exposes Call Script Function. Test one call with 5 points and a second with 0 so you can confirm the return and side effects.

Changing a source signature later does not magically repair every existing node’s saved parameter mapping. Update the exposed definition and revisit callers. If you click Use Definitions with no functions added, you have accepted an empty list. Nothing will appear in Call Script Function until you add and save a definition.

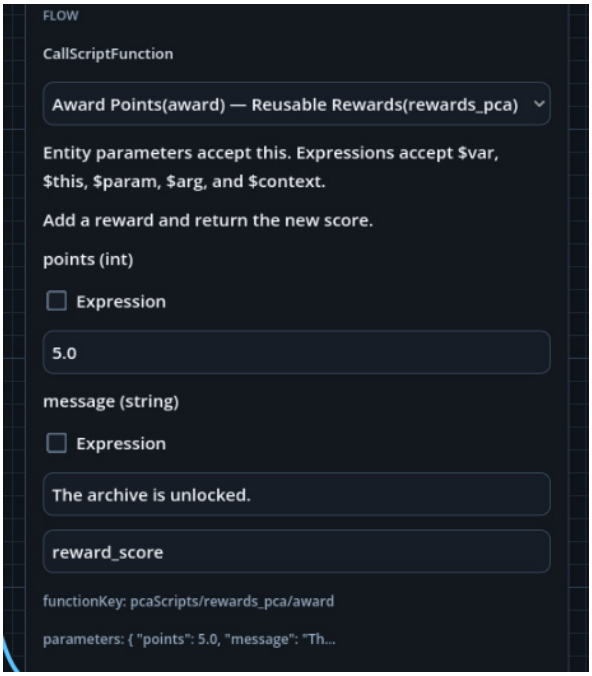

Figure 12. A visual function-call node exposes the published parameters as editable fields.

33. Work with GDScript, Lua and BASIC

PCA Script, native GDScript, Lua, BASIC and visual graphs share a script router and gameplay runtime. They do not share one syntax. Use `call_script(id, function, arguments, context)` at the boundary and pass ordinary data.

A native GDScript helper can supply an operation PCA does not have. Create a GDScript record `math_native` using the native Script workspace. A record without an explicit extends line receives the engine's script base.

```
function run()
    return call_script("math_native", "square_root", [81])
end
```

```
func square_root(value: float) -> float:
    return sqrt(value)
```

The small block above is GDScript, not PCA Script. A call to square_root with 81 returns 9. A native helper can use supported Godot APIs; it must still be written and tested for the platforms you intend to export.

For a Lua record `math_lua`, use this module. Lua's ordinary numeric array syntax starts at 1 inside Lua; data crossing the bridge is normalized to PCA/Godot arrays, which are zero-based in PCA.

```
local Script = {}
function Script.add_points(base, bonus)
    return base + bonus
end
return Script
```

```
function run()
    return call_script("math_lua", "add_points", [10, 5])
end
```

Lua action wrappers that return a signal need Lua's `await(...)`. PCA calls handle their own suspension; do not paste that Lua await syntax into PCA source.

34. BASIC, visual graphs and calls back into PCA

Create a BASIC record `math_basic` with this source. Its entry point conventions differ from PCA, but a named function can still receive the same positional arguments.

```
OPTION PCA3D BASIC 1.0
FUNCTION AddPoints(base AS Integer, bonus AS Integer) AS Integer
    RETURN base + bonus
END FUNCTION
```

```
function run()
    return call_script("math_basic", "AddPoints", [10, 5])
end
```

GDScript can call a PCA function through `CallScript`. This is a native script helper, with capitalized spelling and the native await keyword:

```
func run(context: Dictionary = {}) -> void:
    var total = await CallScript("utility_pca", "add_points", [10, 5])
    Log(str(total))
```

A Lua caller uses `await(CallScript("utility_pca", "add_points", {10, 5}))`. A BASIC caller uses `AWAIT CallScript("utility_pca", "add_points", [10, 5])`. In a visual graph, use Call Script / Call Script Function and its argument or parameter fields.

Boundary rule	Why it matters
Unique script IDs	The router resolves an ID across language collections.
Ordinary data	Text, numbers, arrays and dictionaries are the clearest contract.
Explicit results	Do not rely on another language’s private local variables.
Shared state chosen deliberately	Globals are game state; included variables are router session state.
Own language syntax	A GDScript record proxy is not a PCA namespace or method API.

This build’s supported authored language routes are PCA Script, GDScript, Lua, BASIC and visual graphs. This guide does not claim C# or Python support.

35. Debugging in the editor

1. Validate first. Double-click a diagnostic to go to its line. Read the first error before chasing later errors that may be consequences.
2. Add `log()` calls at the start of important branches. Log the value you are testing, not just “here”. Use `to_json()` for a dictionary of ordinary data.
3. Use the code editor breakpoint gutter, then run with the runtime debugger available. When paused, inspect locals, globals and context in the watch pane.
4. Use Step to advance and Continue to resume. Remove or disable a diagnostic breakpoint before the final player-facing build.
5. Test the unsuccessful branch, the successful branch, a second click, room re-entry and save/load. A puzzle that works only on its first attempt is unfinished.

```
global powered = false
function run()
    let report = {"powered": powered, "scene": current_scene()}
    log(to_json(report))
    if powered
        return "The machine is already running."
    end
    return "No power yet."
end
```

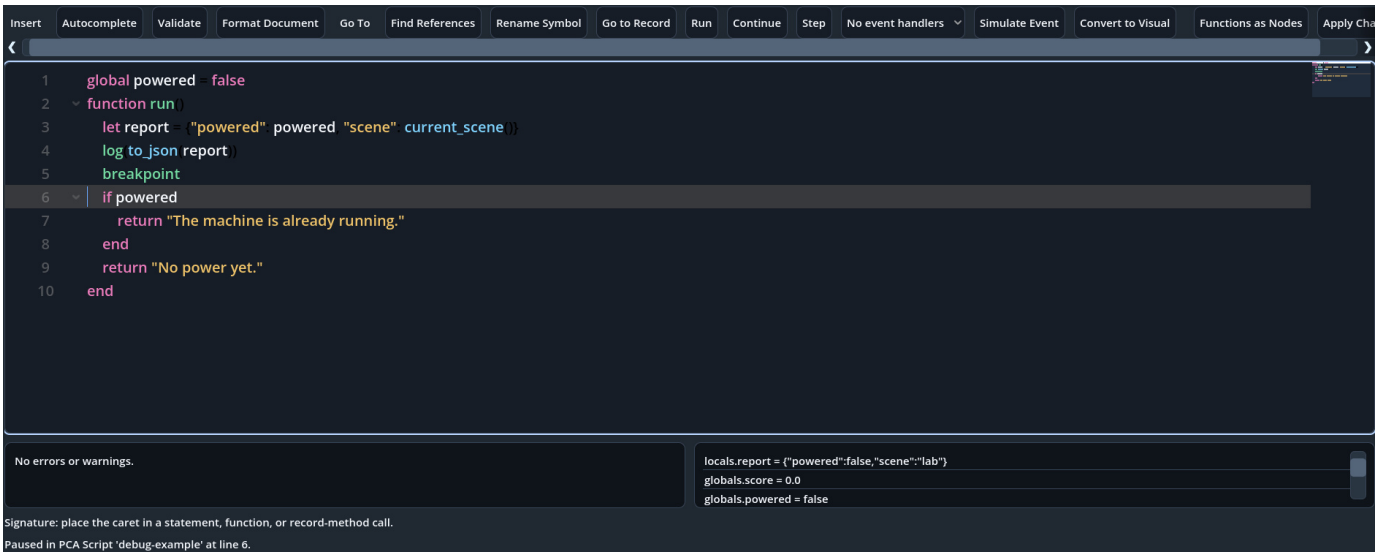

Figure 13. The script editor’s watch pane shows state while execution is paused.

The statement `breakpoint` is a debugger pause point when debugging is enabled. The function helper `breakpoint()` is a diagnostic compatibility helper; use the statement or gutter for stepping. A breakpoint does not replace a condition or set a story variable.

36. A complete, reusable interaction pattern

This example puts the pieces together without hiding a large framework behind them. It reads saved state, checks inventory, prevents a second simultaneous attempt, calls gameplay actions and reports a useful return value.

```
global archive_open = false
global archive_busy = false
global score = 0

function unlock_archive(): boolean
    if archive_open or archive_busy
        return false
    end
    if not has_inventory("key")
        say "The archive key should fit this lock."
        return false
    end
    archive_busy = true
    try
        play_sound({"assetId": "chime", "channel": "puzzle"})
        archive_open = true
        score += 5
        show_message({"text": "Unlocked.", "duration": 1.5})
    catch problem
        warn(problem["message"])
    end
    archive_busy = false
    return archive_open
end

function run()
    return unlock_archive()
end
```

Use `archive_open` to restore the door’s appearance on room entry. Clear transient busy flags deliberately when loading or restarting a puzzle; saving in the middle of an interaction should not leave it locked forever. If an action fails after the fact is set, decide whether the story should stay completed or roll back. That is a design decision, not something try/catch chooses for you.

The later reference chapters are lookup material. Look up the exact function, property key or event phrase you need, then return to this small interaction pattern. Most adventure logic is still a requirement, a state change and a response.

37. Speech, portraits and typewriter messages

Speech, narrator messages and a branching dialogue are different jobs. `say` is a convenient spoken line; `show_message` presents a message; `start_dialogue` runs a graph authored in the Dialogue Designer. A portrait and its mouth animation come from the speaker's configured portrait/lip-sync setup.

```
function run()
    let line = {}
    line["text"] = "Someone has been here before us."
    line["duration"] = 2
    line["typewriter"] = true
    line["typewriterSpeed"] = 35
    line["typewriterSound"] = "default"
    show_message(line)
end
```

For silent typing choose `none`. For an imported typing sound choose `custom` and set `typewriterSoundAssetId` to the audio asset ID. Duration and continuation settings still control the surrounding presentation. Test a short line and a long line so the player is not forced to wait after the text has finished.

For character speech, use the Say action's speaker/portrait/voice fields shown in the reference. A plain text string does not automatically create voice audio, visemes or a portrait mouth mapping. Configure the character and portrait first, then test the same speech route used by the game.

> Keep translated player-facing text in the game's translation workflow. `format()` only assembles a string; it is not a request to translate it or to look up an arbitrary new translation key.

38. Cameras, effects and clean restoration

```
function run()
    activate_camera({"cameraId": "wide", "duration": 0.8, "wait": true})
    let blur = {}
    blur["strength"] = 0.2
    blur["duration"] = 0.5
    blur["includeGUI"] = false
    blur["wait"] = true
    zoom_blur(blur)
end
```

Activate Camera uses an authored camera, including its configured blend behavior when duration is -1. Use an explicit positive duration for a predictable test. Camera property configuration and selecting the active view are different operations.

Zoom Blur suits a brief shock, impact or attention beat. A small strength and short duration are easier to read than a permanent blur. Use screen shader/effect actions for a full-screen treatment and region/object properties for local presentation; they are different scopes.

Before changing an ongoing effect, read/store the original setting if the player must return to it. Restore input locks, cursor visibility, camera follow and temporary effects on normal completion and cancellation. For many cinematic beats, author the sequence as a cutscene and call it from PCA.

A field in the reference is not a promise that every renderer implements the same effect. Confirm the appearance in the render mode and target build your players will use. Material shine is not automatically a dynamic mirror of the character.

39. Configure the project and authored interfaces

```
function run()
    let previous = get_setting("musicVolume", 0.7)
    set_setting("musicVolume", 0.5)
    set_gui_element_property("hud", "topline", "text", "Workshop powered")
    set_gui_property("hud", "visible", true)
    return previous
end
```

Project settings are reached with `get_project_property()` / `set_project_property()`. The shorter setting helpers address values below `settings`. They operate on the running game's data. They are not commands to open the Project Settings Designer or permanently save the editor document.

Use the exact key list in the reference. Preserve the property's type: a Boolean checkbox expects true/false, an asset selector expects an ID, a radius expects a number and a shader layer stack expects structured data. A new key with a plausible name does nothing unless the runtime uses it.

GUI event behavior is configured in the GUI Designer. A button can call a PCA script and let that script set text, visibility, images, widget state and gameplay actions. The designer's interaction preview is for appearance and pointer state; use a running game to test scripts and event chains.

For inventory/save widgets, target the intended outer panel, slot, thumbnail or navigation part. Their image and corner-radius overrides can be separate. Use a setter for the specific key rather than writing the same whole-element radius to every nested part.

40. What “all designer functions” means

PCA Script controls game records and the running game. It is not an automation language for clicking every editor button. This distinction matters when you move from a designer tutorial to a script reference.

Designer / system	PCA route
Scene Designer	Room/entity/camera/spawn/trigger properties; movement, room, material, effect and camera actions.
Character Designer	Character definition properties; actor movement, animation, speech, lip-sync, attachment and switching actions.
GUI Designer	Whole-GUI and element helpers; list-box, carousel, model, video, inventory/save-widget actions.
Dialogue / Cutscene	Run/end sessions and supported dialogue mutations; author complex graphs/timelines in their designers.
Inventory / Quests / Variables	Definition namespaces plus runtime bag/progress/global-variable actions and queries.
Verb / Cursor / Text Parser	Verb/cursor properties and selection/control actions; parser configuration and script caller context.
Assets / Fonts / Translations	Read supported imported-record data; use the editor’s import/font/translation workflows to create content.
Project / Global Settings	Project and setting helpers; dedicated actions for immediate runtime control.
Visual / PCA / Lua / BASIC / native scripts	Router calls and exposed function metadata; conversion creates supported new records through the editor.

There is no built-in PCA command to import an arbitrary model, bake navigation, drag a designer gizmo, export a desktop build, rewrite the translation spreadsheet or edit the editor theme. Use the corresponding designer/tool. A native helper is an explicit extension, not a hidden PCA function.

41. Conversion, organization and testing

The PCA editor includes autocomplete, validation, formatting, Go To, references, rename, Go to Record, event simulation and conversion to visual scripting. Use them to keep a script readable before it becomes large.

Conversion creates a new record and preserves the original. It supports a useful subset of constructs; warnings matter. A complex loop, custom data operation or cross-language call is not guaranteed to turn into an equivalent editable graph. Keep the source version as the reference until you have tested the converted behavior.

Check	What to try
Branch coverage	No key, correct key, already solved, wrong room/actor.
Repeatability	Double click, repeated trigger entry, duplicate event listeners.
Lifecycle	Leave/re-enter a room, stop/restart a timer, switch player.
Persistence	Save before and after the change; reload both.
Interface	Empty input, extra spaces, long text, pointer release, keyboard focus.
Distribution	Run the exported build with its actual assets and renderer.

Keep one job per script/function where practical. Names such as `unlock_archive`, `accept_name` and `restore_lab_state` tell a future reader what the call is supposed to do. A small library of clear functions is easier to repair than a single script that owns the entire game.

> No practice project is required for this book. The examples name a small workshop so the references stay consistent. Adapt the IDs to your existing project and test on a copy before adding save deletion or quit operations.

Built-in functions: how to read this reference

The following 89 entries cover all 103 builtin spellings recognized by this build, with aliases grouped together. Signatures show the ordinary supported parameters. A value after = is the default; a question mark marks an optional argument.

A function used in an expression can return a value. A statement such as wait, return or when is a language construct, not a missing function from this list. The 206 gameplay wrappers have their own section after the builtins.

> Call examples illustrate the argument order. Replace project IDs and file names with your own. A file write, quit request or gameplay call has a real effect when executed; do not run the whole reference as one script.

Builtins: Values and conversion

vector2()

`vector2(x = 0, y = 0)`

Construct a two-coordinate value. Use it for 2D positions and offsets.

Returns: Vector2.

```
vector2(120, 80)
```

vector3()

`vector3(x = 0, y = 0, z = 0)`

Construct a three-coordinate value. In a normal 3D room, Y is height and X/Z span the floor.

Returns: Vector3.

```
vector3(2, 0, -3)
```

color()

`color(text) or color(r, g, b, a = 1)`

Construct a color from an HTML-style string or normalized numeric components.

Returns: Color.

```
color("#88bbff")
```

Invalid color text falls back to white. Components normally use 0 to 1; a hex string is often easier to copy from a designer.

len() and aliases

`len(value)`

Count text characters or array/dictionary entries.

Returns: Integer size; 0 for null or an unsupported value.

Aliases: `length()`

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

str() and aliases

`str(value)`

Convert a value to its display text.

Returns: Text.

Aliases: `string()`, `text()`

```
str(25)
```

This is a conversion, not a translation lookup or automatic GUI wildcard expansion.

number() and aliases

`number(value)`

Convert to a floating-point number.

Returns: Number.

Aliases: `float()`

```
number("2.5")
```

Validate player-entered numeric text first if malformed input must be rejected.

integer() and aliases

`integer(value)`

Convert to a whole integer. Fractional parts are discarded toward zero.

Returns: Integer.

Aliases: `int()`

```
integer(3.8)
```

boolean() and aliases

`boolean(value)`

Convert using runtime truthiness.

Returns: Boolean.

Aliases: `bool()`

```
boolean(1)
```

A nonempty text such as "false" is not an English Boolean parser. Compare normalized input to an accepted word yourself.

Builtins: Math and randomness

min()

`min(a, b, ...)`

Find the smallest of separate numeric arguments.

Returns: Number/value selected by numeric comparison.

```
min(8, 3, 11)
```

This is not an array reducer: loop through an array when its length is dynamic.

max()

`max(a, b, ...)`

Find the largest of separate numeric arguments.

Returns: Number/value selected by numeric comparison.

```
max(8, 3, 11)
```

abs()

`abs(value)`

Remove a number’s sign.

Returns: Nonnegative numeric value.

```
abs(-12)
```

round()

`round(value)`

Round to the nearest whole value.

Returns: Whole numeric value.

```
round(2.6)
```

floor()

`floor(value)`

Round down toward negative infinity.

Returns: Whole numeric value.

```
floor(-2.2)
```

ceil()

`ceil(value)`

Round up toward positive infinity.

Returns: Whole numeric value.

```
ceil(2.2)
```

clamp()

`clamp(value, minimum, maximum)`

Keep a numeric value inside a limit.

Returns: Number between the supplied limits.

```
clamp(120, 0, 100)
```

lerp()

`lerp(start, finish, weight)`

Interpolate two numbers. A weight of 0 chooses start; 1 chooses finish.

Returns: Number.

```
lerp(10, 20, 0.25)
```

The helper is numeric, not a vector or color interpolation API. Clamp weight yourself if extrapolation is unwanted.

distance()

`distance(a, b)`

Measure two Vector2 values, two Vector3 values, or the absolute difference of two numbers.

Returns: Number; 0 for unsupported pairs.

```
distance(vector3(0, 0, 0), vector3(3, 0, 4))
```

Do not pass character references and expect positions to be discovered automatically.

random_int() and aliases

`random_int(minimum, maximum)`

Choose a whole number, including both endpoints.

Returns: Integer.

Aliases: `random_integer()`

```
random_int(1, 6)
```

random_number() and aliases

`random_number(minimum = 0, maximum = 1)`

Choose a random floating-point value in the supplied range.

Returns: Number.

Aliases: `random_float()`

```
random_number(0.25, 0.75)
```

With one argument, it is the minimum and the maximum remains 1. Use two arguments for clarity.

randomize()

`randomize(seed_value?)`

Reseed the shared random generator. Omit the seed for normal varied play; pass a seed for reproducible tests.

Returns: Null.

```
randomize(1234)
```

The generator is shared with other random work. Do not reseed on every roll.

choose()

`choose(values, fallback = null)`

Pick one entry from an array.

Returns: Selected value; fallback for an empty/non-array value.

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

shuffle()

`shuffle(values)`

Create a shuffled copy of an array.

Returns: New array; empty array for unsupported input.

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

The supplied array is not shuffled in place.

Builtins: Text and JSON

lower()

`lower(text)`

Convert text to lowercase.

Returns: Text.

```
lower("OPEN")
```

upper()

`upper(text)`

Convert text to uppercase.

Returns: Text.

```
upper("archive")
```

trim()

`trim(text)`

Remove whitespace from the beginning and end.

Returns: Text.

```
trim("  key  ")
```

contains()

`contains(text, search, case_sensitive = true)`

Check whether text contains a substring.

Returns: Boolean.

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

This builtin is a text search. For array membership, compare entries in a loop.

starts_with()

`starts_with(text, prefix, case_sensitive = true)`

Check the beginning of text.

Returns: Boolean.

```
starts_with("open archive", "open ")
```

ends_with()

`ends_with(text, suffix, case_sensitive = true)`

Check the end of text.

Returns: Boolean.

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

replace()

`replace(text, search, replacement)`

Replace matching occurrences of text.

Returns: New text.

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

Matching is case-sensitive.

split() and aliases

`split(text, delimiter = " ", allow_empty = true)`

Break text into an array at a delimiter.

Returns: Array of text pieces.

Aliases: `string_split()`

```
split("key, note,watch", ",", false)
```

Empty entries and whitespace around entries are separate matters; trim the pieces when required.

join()

`join(values, delimiter = "")`

Join array entries, converting each to text.

Returns: Text.

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

string_length()

`string_length(value)`

Count text characters; this implementation also accepts array/dictionary sizes.

Returns: Integer.

```
string_length("archive")
```

Use len() when the intention is a general container count.

format()

`format(template, values?)`

Replace {key} from a dictionary, {0} positions from an array, or {value} for a single value.

Returns: Text.

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

There are no printf precision directives or embedded PCA expressions in the braces.

regex_search()

`regex_search(pattern, text, all_matches = false)`

Search using a regular expression. Escape backslashes for the PCA string before the regex receives it.

Returns: A match dictionary or null; an array of matches when all_matches is true.

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

A match has text, start, end and groups. groups[0] is the full match. Invalid patterns report a diagnostic and return null/empty. End positions are exclusive.

to_json()

`to_json(value, indent = "")`

Serialize ordinary data to JSON text.

Returns: Text.

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

Use arrays, dictionaries, numbers, text, Booleans and null for portable interchange. Vectors, colors and record references are not a general lossless format.

from_json()

`from_json(text, fallback = null)`

Parse JSON text.

Returns: Parsed value; fallback on invalid JSON.

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

Valid JSON null stays null. This differs from the read_json_file helper’s fallback behavior.

Builtins: Validity and gameplay queries

is_valid()

`is_valid(value)`

Check a record’s existence, an object’s validity or whether an ordinary value is non-null.

Returns: Boolean.

```
is_valid(0)
```

False and 0 are valid values. This is not a truthiness test.

type_of()

`type_of(value)`

Return the PCA-facing type name of a value.

Returns: Text such as integer, number, text, array, dictionary, vector3, color, record or null.

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

exists()

`exists(value)`

Test whether a record reference exists, or an ordinary value is non-null.

Returns: Boolean.

```
exists(objects.archive_door)
```

For a text value it does not interpret that text as a record ID; use an actual reference or a specific lookup.

has_inventory() and aliases

`has_inventory(item, character = "player", count = 1)`

Check inventory possession through the game’s inventory route.

Returns: Boolean.

Aliases: `player_has()`

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

The item comes first. Character-owned and shared inventory modes affect the result.

inventory_count()

`inventory_count(item, character = "player")`

Count copies of an inventory item for the selected owner/route.

Returns: Integer.

```
inventory_count("key", "player")
```

current_scene()

`current_scene()`

Get the currently active room ID.

Returns: Text ID.

```
current_scene()
```

active_player()

`active_player()`

Get the active player character ID.

Returns: Text ID.

```
active_player()
```

has_visited()

`has_visited(scene, character?)`

Check recorded visit history.

Returns: Boolean.

```
has_visited("lab", "ada")
```

Omitting the character follows the runtime’s default player route.

first_visit()

`first_visit(scene?)`

Ask whether this is a first visit for a room, using the current event’s room when omitted.

Returns: Boolean.

```
first_visit("lab")
```

Use in the appropriate entry context. It is a visit-history query, not a persistent once-only wrapper for any arbitrary function.

animation_ended()

`animation_ended(character)`

Check the actor’s current animation completion state.

Returns: Boolean.

```
animation_ended("player")
```

A looping clip does not naturally end. Use with wait until only when completion can occur.

quest_state()

`quest_state(quest)`

Read the runtime state of a quest.

Returns: State text, such as inactive, active, completed or failed.

```
quest_state("signal_quest")
```

objective_state()

`objective_state(quest, objective)`

Read the state of an objective within a quest.

Returns: State text.

```
objective_state("signal_quest", "find_key")
```

if_player_was_here()

`if_player_was_here(scene)`

Compatibility query for player visit history.

Returns: Boolean.

```
if_player_was_here("lab")
```

if_player_was_never_here()

`if_player_was_never_here(scene)`

Compatibility query for the absence of player visit history.

Returns: Boolean.

```
if_player_was_never_here("lab")
```

if_variable()

`if_variable(name, expected = true)`

Compare a runtime game variable to a value.

Returns: Boolean.

```
if_variable("powered", true)
```

This reads game variables, not a same-named local let. A direct PCA if expression is clearer for locals.

if_inventory_item_was_used()

`if_inventory_item_was_used(item)`

Check the item supplied in the current interaction context.

Returns: Boolean.

```
if_inventory_item_was_used("key")
```

Possession and use are separate questions. A direct Run may have no item context.

if_character_moving()

`if_character_moving(character = "player")`

Query whether the selected character is moving.

Returns: Boolean.

```
if_character_moving("player")
```

Builtins: Interfaces and project settings

gui_property() and aliases

`get_gui_property(gui, property, fallback = null)`

Read a whole authored GUI property.

Returns: Property value or fallback.

Aliases: `get_gui_property()`

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

set_gui_property()

`set_gui_property(gui, property, value)`

Change a supported property on a whole authored GUI.

Returns: Boolean indicating that the route accepted the target/property update.

```
set_gui_property("hud", "opacity", 0.8)
```

Use the GUI property table for exact keys. This changes the game interface, not the editor theme.

gui_element_property() and aliases

`get_gui_element_property(gui, element, property, fallback = null)`

Read an authored element property. For a loaded control, available live control properties such as text are read first.

Returns: Property value or fallback.

Aliases: `get_gui_element_property()`

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

set_gui_element_property()

`set_gui_element_property(gui, element, property, value)`

Change an element property through the runtime manager.

Returns: Boolean; false when the element route is not found.

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

A change can rebuild a control. Do not assume every widget’s in-progress input/selection survives an unrelated appearance change.

get_project_property() and aliases

`get_project_property(path, fallback = null)`

Read a supported project-data path.

Returns: Value or fallback.

Aliases: `project_property()`

```
get_project_property("meta.name", "Untitled")
```

set_project_property()

`set_project_property(path, value)`

Change a supported runtime project-data path.

Returns: Boolean.

```
set_project_property("settings.musicVolume", 0.5)
```

Use the picker/key reference. Arbitrarily creating a key does not add engine behavior, and live application depends on the setting.

get_setting() and aliases

`get_setting(path, fallback = null)`

Read below the project settings dictionary. A leading settings prefix is normalized by the helper.

Returns: Value or fallback.

Aliases: `setting()`

```
get_setting("musicVolume", 0.7)
```

set_setting()

`set_setting(path, value)`

Change a setting through the runtime project-property bridge.

Returns: Boolean.

```
set_setting("musicVolume", 0.5)
```

For runtime features with a dedicated action, prefer it when immediate reconfiguration matters.

Builtins: Scripts, events and execution

action()

`action(action_type, properties = {})`

Run an exact shared gameplay action ID with an options dictionary.

Returns: The action's result; many actions have no useful return value.

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

Use a supported catalog action. The 206 named wrappers below are easier to discover. An ok:false result becomes a PCA runtime failure.

call_script()

`call_script(script, function = "run", arguments = [], context = {})`

Call a script through the shared language router and wait for its return.

Returns: The called function's returned value.

```
call_script("utility_pca", "add_points", [10, 5])
```

Arguments are positional. Context is an additional dictionary; the caller context is inherited and extended by this route.

include_script()

`include_script(script, function?, arguments?, context?)`

Resolve an existing script ID; with further arguments, call it through the shared router.

Returns: One argument: ID text. More arguments: called result.

```
include_script("utility_pca")
```

It does not paste source, import local function names or execute a module on a one-argument call. A missing script reports a PCA error.

get_included_variable()

`get_included_variable(script, name, fallback = null)`

Read the router’s per-script shared-variable dictionary.

Returns: Value or fallback.

```
get_included_variable("utility_pca", "last_total", 0)
```

This store is separate from declarations in that script. It is session/cache state, not automatic save-game state.

set_included_variable()

`set_included_variable(script, name, value)`

Write to the router’s per-script shared-variable dictionary.

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

```
set_included_variable("utility_pca", "last_total", 15)
```

run_visual_graph()

`run_visual_graph(graph, context = {})`

Execute graph data through the visual runtime.

Returns: The visual graph’s returned value, where provided.

```
run_visual_graph({"nodes": [], "connections": []})
```

The first argument is a graph dictionary, not a visual-script ID. Prefer run visual script / the script router for a saved record.

emit_event()

`emit_event(name, data = {})`

Broadcast an exact script event name with additional context.

Returns: Null.

```
emit_event("signal_powered", {"source": "lever"})
```

Use when event signal_powered to listen. A non-dictionary data value is stored under value. Guard against recursive broadcasts.

end_visual_script()

`end_visual_script(end_parent = false)`

Request termination of a running visual graph, optionally its calling graph.

Returns: Runtime result / null.

```
end_visual_script(false)
```

This is not PCA’s return statement and does not break a normal action array.

end()

`end()`

Compatibility helper recognized by the VM.

Returns: Null.

```
end()
```

It is not a quit command. In ordinary source, end is the block terminator; use return to leave a function and end_game for the gameplay end route. Do not use end() as a source-level termination statement.

end_game()

`end_game(message = "", ask_first = false)`

Request the runtime’s end-game route.

Returns: Runtime Boolean result.

```
end_game("Thanks for playing.", true)
```

This affects the player’s game. Test it in a disposable play session.

quit_to_menu()

`quit_to_menu(message = "", ask_first = false)`

Request a return to the game’s menu route.

Returns: Runtime Boolean result.

```
quit_to_menu("Return to the menu?", true)
```

Builtins: Diagnostics

log()

`log(message)`

Write a message to runtime diagnostics.

Returns: Null.

```
log("Archive unlocked.")
```

warn()

`warn(message)`

Report a warning in diagnostics.

Returns: Null.

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

script_error()

`script_error(message)`

Report an error-level diagnostic.

Returns: Null.

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

The helper logs; it does not throw a catchable PCA exception or end the current function.

pca_assert()

`pca_assert(condition, message = "Assertion failed.")`

Report when a diagnostic condition is false.

Returns: The condition as a Boolean.

```
pca_assert(5 > 0, "Score must be positive.")
```

The assert statement is the failing/stop form. This helper reports and returns a value.

breakpoint()

`breakpoint()`

Diagnostic compatibility helper.

Returns: Null.

```
breakpoint()
```

For an actual debugger pause use the standalone breakpoint statement or the editor gutter while debugging is enabled.

Builtins: Calendar and files

now_unix()

`now_unix()`

Read the system clock in whole Unix seconds.

Returns: Integer seconds since the Unix epoch.

```
now_unix()
```

The system clock can be adjusted; do not confuse it with a monotonic frame timer.

date_time_now()

`date_time_now(utc = false)`

Read the current calendar date/time as a dictionary.

Returns: Dictionary including year, month, day, weekday, hour, minute, second and dst.

```
date_time_now(true)
```

date_time_string()

`date_time_string(utc = false, use_space = true)`

Format the current system date/time.

Returns: Text.

```
date_time_string(false, true)
```

unix_to_date_time()

`unix_to_date_time(timestamp, reserved?)`

Convert Unix seconds to a calendar dictionary.

Returns: Dictionary.

```
unix_to_date_time(0)
```

The validator accepts a second argument, but this VM forwards only the timestamp to the conversion. Do not rely on that argument for a timezone change.

read_text_file()

`read_text_file(path, fallback = "")`

Read the complete contents of a text file.

Returns: Text; fallback if the file is missing or cannot be opened.

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

write_text_file()

`write_text_file(path, text)`

Write the complete text, creating parent directories where needed.

Returns: Boolean success.

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

Only user:// writes are accepted. Existing content is overwritten, not appended.

read_json_file()

`read_json_file(path, fallback = null)`

Read a text file and parse its JSON.

Returns: Parsed data or fallback.

```
read_json_file("user://preferences.json", {"music": 0.7})
```

In this helper, JSON null also selects the fallback. Use from_json(read_text_file(...)) when valid null must be distinguished from failure.

write_json_file()

`write_json_file(path, value, indent = "\t")`

Serialize JSON and overwrite a user-data file.

Returns: Boolean success.

```
write_json_file("user://preferences.json", {"music": 0.5}, "  ")
```

This is a custom data file, not a replacement for a full game save.

file_exists()

`file_exists(path)`

Check whether a file exists at an accessible path.

Returns: Boolean.

```
file_exists("user://preferences.json")
```

directory_exists()

`directory_exists(path)`

Check whether a directory exists.

Returns: Boolean.

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

Builtins: Ranges

range()

`range(stop) or range(start, stop, step = 1)`

Build integers starting at start (default 0) and ending before stop.

Returns: Array of integers.

```
range(1, 6)
```

Negative steps count down. Step 0 fails. The builder has a 100,000-item safety limit.

range_inclusive()

`range_inclusive(start, finish, step = 1)`

Build a range that can include the finish value when reached by the step.

Returns: Array of integers.

```
range_inclusive(1, 5)
```

An incompatible direction gives no values. A step that skips the endpoint does not invent an extra final value.

Gameplay functions: the complete shared catalog

These 206 functions are generated from the shared gameplay action catalog in this build. Every entry gives the PCA function name, exact action ID, purpose, full options schema, starting values, timing and a practical use. These are the normal shared wrappers available to PCA Script; legacy/native-only action spellings are not invented as extra snake_case functions.

```
camera_shake({"duration": 0.5, "power": 0.25})
action("CameraShake", {"duration": 0.5, "power": 0.25})
```

Each named wrapper accepts zero or one options dictionary. Omitted fields use the catalog defaults shown. The call patterns show a few important fields, not an instruction to run every action with placeholder targets. Values such as your_asset need replacing with a real ID.

The runtime result belongs to that action: it may be null, Boolean, a query result or structured data. Do not assume every call returns success=true. A structured ok:false failure becomes a PCA runtime error; use try/catch where recovery makes sense.

Units: most durations are seconds; normalized volumes, opacity and effect amounts usually run from 0 to 1. A missing target is not inferred from whichever object is selected in the editor. Use the exact camelCase option keys shown.

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.

set_variable()

Set Variable | Action ID: `SetVariable`

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 accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
set_variable({"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.

create_variable()

Create Variable | Action ID: `CreateVariable`

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 accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
create_variable({"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.

change_variable()

Change Variable | Action ID: `ChangeVariable`

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

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
change_variable({"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.

add_score()

Add Score | Action ID: `AddScore`

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

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

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
add_score({"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.

unlock_hint()

Unlock Hint Topic / Answer | Action ID: `UnlockHint`

Unlock a numbered answer within an existing hint topic.

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

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
unlock_hint({"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.

run_visual_script()

Run Visual Script | Action ID: `RunVisualScript`

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 accepted input	Default
Script `scriptId`	Choose a visual or supported text script.	Blank

Timing. Waits for the selected script call.

```
run_visual_script({"scriptId": "utility_pca"})
```

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.

call_script_function()

Call Script Function | Action ID: `CallScriptFunction`

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

Option key / field	Meaning and accepted 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.

```
let options = {}
options["functionKey"] = "pcaScripts/rewards_pca/award"
options["parameters"] = {"points": 5, "message": "Well done."}
options["resultVariable"] = ""
call_script_function(options)
```

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.

add_inventory()

Add Inventory | Action ID: `AddInventory`

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

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

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
add_inventory({"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.

remove_inventory()

Remove Inventory | Action ID: `RemoveInventory`

Remove copies of an item through the current inventory route.

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

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
remove_inventory({"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.

change_inventory_quantity()

Change Inventory Quantity | Action ID: `ChangeInventoryQuantity`

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

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
change_inventory_quantity({"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.

transfer_inventory()

Change Inventory from Player to Player | Action ID: `TransferInventory`

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

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
transfer_inventory({"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.

execute_preposition()

Execute Two-Object Command | Action ID: `ExecutePreposition`

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

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
execute_preposition({"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.

cancel_preposition()

Cancel Two-Object Command | Action ID: `CancelPreposition`

Cancel the currently assembled two-object interaction.

No options in the standard shared schema.

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
cancel_preposition({})
```

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.

show_message()

Show Message | Action ID: ShowMessage

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

Option key / field	Meaning and accepted 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.

```
show_message({"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 values: default, none or custom. Custom uses typewriterSoundAssetId. Speed is characters per second.

say()

Say | Action ID: 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 accepted 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.

```
say({"speakerId": "player", "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 values: default, none or custom. Custom uses typewriterSoundAssetId. Speed is characters per second.

set_voice_speaker()

Set Voice Speaker | Action ID: `SetVoiceSpeaker`

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

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

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
set_voice_speaker({"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.

set_lip_sync_enabled()

Enable Lip Sync | Action ID: `SetLipSyncEnabled`

Enable or disable lip sync for a character at runtime.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
set_lip_sync_enabled({"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.

play_voice()

Play Voice | Action ID: `PlayVoice`

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

Option key / field	Meaning and accepted 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.

```
play_voice({"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.

pause_voice()

Pause Voice | Action ID: `PauseVoice`

Pause the current voice and its mouth timing.

No options in the standard shared schema.

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
pause_voice({})
```

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.

resume_voice()

Resume Voice | Action ID: ResumeVoice

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

No options in the standard shared schema.

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
resume_voice({})
```

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.

seek_voice()

Seek Voice | Action ID: SeekVoice

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

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

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
seek_voice({"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.

stop_voice()

Stop Voice | Action ID: `StopVoice`

Stop the current voice and reset its mouth presentation.

No options in the standard shared schema.

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
stop_voice({})
```

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.

start_dialogue()

Start Dialogue | Action ID: `StartDialogue`

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

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

Timing. Waits for the dialogue session.

```
start_dialogue({"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.

end_dialogue()

End Dialogue | Action ID: `EndDialogue`

End the active dialogue session.

No options in the standard shared schema.

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
end_dialogue({})
```

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.

add_dialogue_line()

Add Dialogue Line | Action ID: `AddDialogueLine`

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 accepted input	Default
Dialogue `dialogueId`	Choose a Dialogue Designer ID.	Blank
New Node ID `nodeId`	Text.	Blank
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

Option key / field	Meaning and accepted input	Default
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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
add_dialogue_line({"dialogueId": "", "nodeId": "", "speakerId": "player"})
```

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 values: default, none or custom. Custom uses typewriterSoundAssetId. Speed is characters per second.

add_dialogue_choice_node()

Add Dialogue Choice Node | Action ID: `AddDialogueChoiceNode`

Create a choice node in an existing dialogue.

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

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
add_dialogue_choice_node({"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.

remove_dialogue_node()

Remove Dialogue Node | Action ID: `RemoveDialogueNode`

Remove a node from the runtime dialogue graph.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
remove_dialogue_node({"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.

add_dialogue_choice()

Add Dialogue Choice | Action ID: `AddDialogueChoice`

Add or update an option inside a choice node.

Option key / field	Meaning and accepted 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

Option key / field	Meaning and accepted input	Default
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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
add_dialogue_choice({"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 values: default, none or custom. Custom uses typewriterSoundAssetId. Speed is characters per second.

remove_dialogue_choice()

Remove Dialogue Choice | Action ID: `RemoveDialogueChoice`

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

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
remove_dialogue_choice({"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.

set_dialogue_choice_enabled()

Enable / Disable Dialogue Choice | Action ID: `SetDialogueChoiceEnabled`

Enable or disable an existing choice without removing its record.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
set_dialogue_choice_enabled({"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.

reset_dialogue_history()

Reset Dialogue History | Action ID: `ResetDialogueHistory`

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

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
reset_dialogue_history({"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.

character_walk_to()

Character Walk To | Action ID: `CharacterWalkTo`

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

Option key / field	Meaning and accepted 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.

```
character_walk_to({"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.

clear_character_waypoints()

Character Waypoints: Clear Queue | Action ID: `ClearCharacterWaypoints`

Empty a character's queued destinations.

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

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
clear_character_waypoints({"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.

add_character_waypoint()

Character Waypoints: Add Waypoint | Action ID: `AddCharacterWaypoint`

Append one destination to a character's waypoint queue.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
add_character_waypoint({"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.

character_walk_waypoints()

Character Waypoints: Walk Queue | Action ID: `CharacterWalkWaypoints`

Run the queued waypoint route for a character.

Option key / field	Meaning and accepted 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.

```
character_walk_waypoints({"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.

character_walk_path()

Character Walk Through Waypoints | Action ID: `CharacterWalkPath`

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

Option key / field	Meaning and accepted 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.

```
character_walk_path({"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.

character_teleport()

Character Instant Teleport To | Action ID: `CharacterTeleport`

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

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
character_teleport({"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.

move_npc_to_room()

Move NPC to a Different Room | Action ID: `MoveNPCToRoom`

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

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
move_npc_to_room({"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.

follow_character()

Follow Character | Action ID: `FollowCharacter`

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

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
follow_character({"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.

stop_follow_character()

Stop Follow Character | Action ID: `StopFollowCharacter`

Stop the chosen character's follow behavior.

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

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
stop_follow_character({"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.

face_character()

Face Character | Action ID: `FaceCharacter`

Turn one character toward another.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
face_character({"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.

face_object()

Face Object | Action ID: `FaceObject`

Turn a character toward a scene object.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
face_object({"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.

face_coordinates()

Face Coordinates | Action ID: `FaceCoordinates`

Turn a character toward an explicit world position.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
face_coordinates({"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.

hide_player_character()

Hide Player Character | Action ID: `HidePlayerCharacter`

Hide or reveal the active player's visual representation.

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

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
hide_player_character({"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.

set_player_input_enabled()

Enable / Disable Player Input | Action ID: `SetPlayerInputEnabled`

Enable or disable player control input.

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

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
set_player_input_enabled({"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.

set_character_action_enabled()

Enable / Disable Character Action | Action ID: `SetCharacterActionEnabled`

Enable or disable a named interaction response on a character.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
set_character_action_enabled({"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.

change_character_property()

Change Character Property | Action ID: `ChangeCharacterProperty`

Change a selected character property in runtime data.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
change_character_property({"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.

add_character_to_scene()

Add Character To Scene | Action ID: `AddCharacterToScene`

Create a scene instance from an existing Character Designer definition.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
add_character_to_scene({"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 | Action ID: `Jump`

Ask the selected character to jump through the adventure extension.

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

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
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.

play_animation()

Play Animation | Action ID: `PlayAnimation`

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

Option key / field	Meaning and accepted 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.

```
play_animation({"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.

stop_animation()

Stop Animation | Action ID: `StopAnimation`

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

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

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
stop_animation({"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.

pause_animation()

Animation: Pause | Action ID: `PauseAnimation`

Pause the target entity's animation playback.

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

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
pause_animation({"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.

resume_animation()

Animation: Resume | Action ID: `ResumeAnimation`

Resume the target entity's paused animation playback.

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

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
resume_animation({"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.

set_object_action_enabled()

Enable / Disable Object Action | Action ID: `SetObjectActionEnabled`

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

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
set_object_action_enabled({"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.

set_object_enabled()

Enable / Disable Object | Action ID: `SetObjectEnabled`

Set an object's enabled state.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
set_object_enabled({"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.

set_object_visible()

Hide / Unhide Object | Action ID: `SetObjectVisible`

Show or hide an object.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
set_object_visible({"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.

set_trigger_enabled()

Enable / Disable Trigger | Action ID: `SetTriggerEnabled`

Enable or disable a trigger's runtime event behavior.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
set_trigger_enabled({"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.

spawn_object()

Spawn Scene Object | Action ID: `SpawnObject`

Spawn or restore an already authored scene object definition.

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

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
spawn_object({"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.

despawn_object()

Despawn Scene Object | Action ID: `Despawn0bject`

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

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

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
despawn_object({"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.

change_object_property()

Change Object Property | Action ID: `ChangeObjectProperty`

Assign a property on an object's runtime record.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
change_object_property({"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.

change_scene_description()

Change Scene Description | Action ID: `ChangeSceneDescription`

Change the description stored for a scene.

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

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
change_scene_description({"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.

change_object_description()

Change Object Description | Action ID: `ChangeObjectDescription`

Change an object's descriptive text.

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

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
change_object_description({"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.

change_inventory_description()

Change Inventory Description | Action ID: `ChangeInventoryDescription`

Change an inventory item's description.

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

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
change_inventory_description({"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.

change_character_description()

Change Character Description | Action ID: `ChangeCharacterDescription`

Change a character's description.

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

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
change_character_description({"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.

change_region_description()

Change Region Description | Action ID: `ChangeRegionDescription`

Change the description of a named region.

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

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
change_region_description({"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.

add_asset_to_scene()

Add Asset to Scene | Action ID: `AddAssetToScene`

Create a runtime scene entity using an imported asset.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
add_asset_to_scene({"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.

create_primitive()

Create Primitive | Action ID: `CreatePrimitive`

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

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
create_primitive({"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.

create_procedural()

Create Procedural | Action ID: `CreateProcedural`

Create a runtime procedural entity from the requested generator type.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
create_procedural({"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.

activate_camera()

Activate / Blend to Camera | Action ID: `ActivateCamera`

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

Option key / field	Meaning and accepted 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.

```
activate_camera({"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.

set_camera_behavior()

Set Advanced Camera Behavior | Action ID: `SetCameraBehavior`

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

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

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
set_camera_behavior({"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.

set_camera_targets()

Set Camera Targets | Action ID: `SetCameraTargets`

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

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
set_camera_targets({"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.

set_camera_priority()

Set Camera Priority | Action ID: `SetCameraPriority`

Change the priority used by the advanced-camera system.

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

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
set_camera_priority({"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.

change_camera()

Change Camera | Action ID: `ChangeCamera`

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

Option key / field	Meaning and accepted input	Default
Room `sceneId`	Choose a scene ID.	Blank
Camera View `cameraId`	Choose a camera in the selected scene.	Blank

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
change_camera({"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.

camera_orbit()

Camera Orbit | Action ID: `CameraOrbit`

Start an orbit around an entity with distance, height, direction, speed and repetitions.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
camera_orbit({"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.

stop_camera_orbit()

Stop Camera Orbit | Action ID: `StopCameraOrbit`

Stop the current extension-driven orbit.

No options in the standard shared schema.

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
stop_camera_orbit({})
```

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.

camera_shake()

Camera Shake | Action ID: `CameraShake`

Shake the 3D camera for the chosen duration and direction, waiting for the shake routine.

Option key / field	Meaning and accepted 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.

```
camera_shake({"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.

shake_screen()

Shake Screen | Action ID: ShakeScreen

Start the extension's screen-shake effect with horizontal, vertical or combined movement.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
shake_screen({"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.

stop_screen_shake()

Stop Screen Shake | Action ID: StopScreenShake

Stop the current extension-driven screen shake.

No options in the standard shared schema.

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
stop_screen_shake({})
```

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.

fade_screen()

Screen Fade | Action ID: FadeScreen

Fade a color over the screen, or fade that cover away, over a duration.

Option key / field	Meaning and accepted 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.

```
fade_screen({"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.

screen_transition()

Screen Transition | Action ID: `ScreenTransition`

Play a screen transition effect such as dissolve, wipe, iris, pixelation or glitch.

Option key / field	Meaning and accepted 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.

```
screen_transition({"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.

set_retro_style()

Set Retro Rendering Style | Action ID: `SetRetroStyle`

Choose the retro screen treatment: none, PSX, N64 or horror fog, with strength and pixel size.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
set_retro_style({"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.

zoom_blur()

Zoom Blur | Action ID: `ZoomBlur`

Play a radial zoom blur around a normalized screen center, optionally including GUI and waiting for completion.

Option key / field	Meaning and accepted 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.

```
zoom_blur({"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.

stop_zoom_blur()

Stop Zoom Blur | Action ID: `StopZoomBlur`

Cancel the active zoom-blur treatment.

No options in the standard shared schema.

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
stop_zoom_blur({})
```

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.

change_scene()

Change Scene | Action ID: ChangeScene

Move the active player into another scene using a spawn and optional destination position/camera/transition settings.

Option key / field	Meaning and accepted 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.

```
change_scene({"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.

PCA dictionaries can supply sceneId directly. Some GUI/shared forms omit the destination scene field; a PCA script using change_scene is a useful bridge from those forms.

switch_character()

Switch Character | Action ID: `SwitchCharacter`

Change the currently controlled playable character.

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

Timing. Awaits the character-switch routine.

```
switch_character({"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.

is_current_room()

Is Current Room | Action ID: `IsCurrentRoom`

Test the current room through the adventure extension and return the result.

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

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
is_current_room({"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.

set_room_dimension()

Set Room Dimension | Action ID: `SetRoomDimension`

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 accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
set_room_dimension({"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.

set2d_camera()

Set 2D Camera | Action ID: `Set2DCamera`

Set the 2D camera's position, zoom and rotation for a scene.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
set2d_camera({"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.

set2d_layer_visible()

Show / Hide 2D Layer | Action ID: `Set2DLayerVisible`

Show or hide a named 2D layer.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
set2d_layer_visible({"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.

set2d_entity_depth()

Set 2D Entity Depth | Action ID: `Set2DEntityDepth`

Change an entity's sorting depth in the 2D/hybrid scene.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
set2d_entity_depth({"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.

set2d_parallax()

Set 2D Layer Parallax | Action ID: `Set2DParallax`

Change a named 2D layer's parallax ratio.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
set2d_parallax({"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.

set_room_camera_bounds()

Change Room Camera Bounds | Action ID: `SetRoomCameraBounds`

Set whether camera bounds are enabled and edit the room's X/Z limits.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
set_room_camera_bounds({"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.

set_interaction_approach_enabled()

Enable / Disable Automatic Interaction Approach | Action ID: `SetInteractionApproachEnabled`

Set the global runtime approach-to-target behavior, including distance and whether to face the target.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
set_interaction_approach_enabled({"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.

set_object_interaction_approach()

Set Object Interaction Approach | Action ID: `SetObjectInteractionApproach`

Override approach behavior for one object, or inherit the global behavior.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
let options = {}
options["sceneId"] = "lab"
options["entityId"] = "archive_door"
options["mode"] = "inherit"
set_object_interaction_approach(options)
```

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.

set_scene_brightness()

Set Room Brightness | Action ID: `SetSceneBrightness`

Change a scene's brightness multiplier.

Option key / field	Meaning and accepted input	Default
Scene `sceneId`	Choose a scene ID.	Blank
Brightness (0 = Dark, 1 = Normal) `brightness`	Number; decimals allowed.	1.0

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
set_scene_brightness({"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.

set_scene_tint()

Set Room Tint | Action ID: `SetSceneTint`

Apply a color tint to a scene with a chosen strength.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
set_scene_tint({"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.

set_flashlight_mode()

Enable / Disable Flashlight Mode | Action ID: `SetFlashlightMode`

Enable or disable the flashlight-style visibility mask and adjust its strength, radius and shape.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
set_flashlight_mode({"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.

set_snow_accumulation()

Set Snow Accumulation | Action ID: `SetSnowAccumulation`

Set or animate accumulated snow on the scene, with duration/pace and optional waiting.

Option key / field	Meaning and accepted 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.

```
set_snow_accumulation({"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.

melt_snow()

Melt Accumulated Snow | Action ID: `MeltSnow`

Reduce the scene's accumulated snow over time.

Option key / field	Meaning and accepted 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.

```
melt_snow({"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.

toggle_motion()

Toggle Motion | Action ID: `ToggleMotion`

Invert an object's authored motion-enabled state, optionally mirroring that state into a variable.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
toggle_motion({"sceneId": "lab", "entityId": "archive_door", "variable": "powered"})
```

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.

set_motion()

Set Motion | Action ID: `SetMotion`

Explicitly enable or disable authored object motion.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
set_motion({"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.

toggle_glow()

Toggle Glow | Action ID: `ToggleGlow`

Invert an object's authored glow state and optionally record it in a variable.

Option key / field	Meaning and accepted input	Default
Entity `entityId`	Choose a scene entity ID.	Blank
Variable `variable`	Text.	Blank

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
toggle_glow({"entityId": "archive_door", "variable": "powered"})
```

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.

set_glow()

Set Glow | Action ID: `SetGlow`

Set an object's authored glow state explicitly.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
set_glow({"entityId": "archive_door", "enabled": true, "variable": "powered"})
```

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.

toggle_rgb_string()

Toggle Rgb String | Action ID: `ToggleRGBString`

Toggle an authored RGB light string, optionally writing its state to a variable.

Option key / field	Meaning and accepted input	Default
Entity `entityId`	Choose a scene entity ID.	Blank
Variable `variable`	Text.	Blank

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
toggle_rgb_string({"entityId": "archive_door", "variable": "powered"})
```

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.

set_attach_point()

Set Attach Point | Action ID: `SetAttachPoint`

Create or update a named attach point with position, rotation, coordinate space, bone and contact radius.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
set_attach_point({"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.

remove_attach_point()

Remove Attach Point | Action ID: `RemoveAttachPoint`

Remove a named attach point from an object or character.

Option key / field	Meaning and accepted input	Default
Object / Character `entityId`	Choose a scene entity ID.	Blank
Attach Point `point`	Name of the source attach point.	Blank

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
remove_attach_point({"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.

get_attach_point()

Get Attach Point | Action ID: `GetAttachPoint`

Read an attach point's information and optionally store the result in a game variable.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
get_attach_point({"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.

attach_points()

Attach Points | Action ID: `AttachPoints`

Attach one entity's point to another entity's point, optionally aligning rotation and suspending movement.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
attach_points({"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.

align_attach_points()

Align Attach Points | Action ID: `AlignAttachPoints`

Align two attach points once without establishing the same continuing attachment relationship.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
align_attach_points({"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.

attach_points_touching()

Attach Points Touching | Action ID: `AttachPointsTouching`

Check whether two points are touching within their radii and tolerance, and optionally store the result.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
attach_points_touching({"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.

get_attach_point_contacts()

Get Attach Point Contacts | Action ID: `GetAttachPointContacts`

Get contact information for the requested point/target filters.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
let options = {}
options["entityId"] = "archive_door"
options["point"] = ""
options["targetId"] = "archive_door"
get_attach_point_contacts(options)
```

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.

watch_attach_point_contacts()

Watch Attach Point Contacts | Action ID: `WatchAttachPointContacts`

Start a named contact watch, optionally using swept detection between frames.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
watch_attach_point_contacts({"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.

stop_attach_point_watch()

Stop Attach Point Watch | Action ID: `StopAttachPointWatch`

Stop a previously started contact watch by its watch ID.

Option key / field	Meaning and accepted input	Default
Watch ID `watchId`	Stable name for this continuing contact watch.	Blank

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
stop_attach_point_watch({"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.

attach_object()

Attach Object | Action ID: `AttachObject`

Attach an object to a target object's attach point.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
attach_object({"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.

detach_object()

Detach Object | Action ID: `DetachObject`

Detach an object that was attached through the attachment system.

Option key / field	Meaning and accepted input	Default
Object `entityId`	Choose a scene entity ID.	Blank

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
detach_object({"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.

attach_particle_effect()

Attach Particle Effect | Action ID: `AttachParticleEffect`

Attach an existing particle-effect entity to a target point with an optional offset.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
let options = {}
options["entityId"] = "archive_door"
options["targetId"] = "archive_door"
options["targetPoint"] = ""
attach_particle_effect(options)
```

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.

detach_particle_effect()

Detach Particle Effect | Action ID: `DetachParticleEffect`

Detach a particle-effect entity from its target.

Option key / field	Meaning and accepted input	Default
Particle Effect `entityId`	Choose a scene entity ID.	Blank

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
detach_particle_effect({"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.

show_gui()

Show Gui | Action ID: `ShowGUI`

Show an authored interface.

Option key / field	Meaning and accepted input	Default
GUI `guiId`	Choose an authored GUI ID.	Blank

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
show_gui({"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.

hide_gui()

Hide Gui | Action ID: `HideGUI`

Hide an authored interface.

Option key / field	Meaning and accepted input	Default
GUI `guiId`	Choose an authored GUI ID.	Blank

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
hide_gui({"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.

enable_gui()

Enable Gui | Action ID: `EnableGUI`

Enable an authored interface through the runtime GUI manager.

Option key / field	Meaning and accepted input	Default
GUI `guiId`	Choose an authored GUI ID.	Blank

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
enable_gui({"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.

disable_gui()

Disable Gui | Action ID: `DisableGUI`

Disable an authored interface through the runtime GUI manager.

Option key / field	Meaning and accepted input	Default
GUI `guiId`	Choose an authored GUI ID.	Blank

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
disable_gui({"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.

show_gui_element()

Show Gui Element | Action ID: `ShowGUIElement`

Show one element inside an authored GUI.

Option key / field	Meaning and accepted input	Default
GUI `guiId`	Choose an authored GUI ID.	Blank
Element / Group `elementId`	Choose an element inside the selected GUI.	Blank

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
show_gui_element({"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.

hide_gui_element()

Hide Gui Element | Action ID: `HideGUIElement`

Hide one element inside an authored GUI.

Option key / field	Meaning and accepted input	Default
GUI `guiId`	Choose an authored GUI ID.	Blank
Element / Group `elementId`	Choose an element inside the selected GUI.	Blank

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
hide_gui_element({"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.

enable_gui_element()

Enable Gui Element | Action ID: `EnableGUIElement`

Enable one element for interaction.

Option key / field	Meaning and accepted input	Default
GUI `guiId`	Choose an authored GUI ID.	Blank
Element / Group `elementId`	Choose an element inside the selected GUI.	Blank

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
enable_gui_element({"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.

disable_gui_element()

Disable Gui Element | Action ID: `DisableGUIElement`

Disable one element for interaction.

Option key / field	Meaning and accepted input	Default
GUI `guiId`	Choose an authored GUI ID.	Blank
Element / Group `elementId`	Choose an element inside the selected GUI.	Blank

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
disable_gui_element({"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.

change_gui_property()

Change Gui Property | Action ID: `ChangeGUIProperty`

Change a supported property on an authored GUI.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
change_gui_property({"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.

change_gui_element_property()

Change Gui Element Property | Action ID: `ChangeGUIElementProperty`

Change a supported property on a GUI element.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
change_gui_element_property({"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.

list_box_add_item()

List Box: Add Item | Action ID: `ListBoxAddItem`

Add a manual list-box item with an ID, text, optional icon, disabled state and insertion index.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
list_box_add_item({"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.

list_box_delete_item()

List Box: Delete Item | Action ID: `ListBoxDeleteItem`

Delete a list-box item by its item ID.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
list_box_delete_item({"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.

list_box_modify_item()

List Box: Modify Item | Action ID: `ListBoxModifyItem`

Change an existing list item's text, ID, icon, disabled or selected property.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
list_box_modify_item({"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.

list_box_clear_items()

List Box: Clear Items | Action ID: `ListBoxClearItems`

Clear the manual items in a list box.

Option key / field	Meaning and accepted input	Default
GUI `guiId`	Choose an authored GUI ID.	Blank
List Box `elementId`	Choose an element inside the selected GUI.	Blank

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
list_box_clear_items({"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.

carousel_add_image()

Carousel: Add Image | Action ID: `CarouselAddImage`

Add an imported image asset to an image carousel.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
carousel_add_image({"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.

carousel_remove_image()

Carousel: Remove Image | Action ID: `CarouselRemoveImage`

Remove a selected image asset from a carousel's authored/runtime list.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
carousel_remove_image({"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.

carousel_clear_images()

Carousel: Clear Images | Action ID: `CarouselClearImages`

Clear a carousel's image list.

Option key / field	Meaning and accepted input	Default
GUI `guiId`	Choose an authored GUI ID.	Blank
Image Carousel `elementId`	Choose an element inside the selected GUI.	Blank

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
carousel_clear_images({"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.

carousel_previous()

Carousel: Previous Image | Action ID: `CarouselPrevious`

Move a carousel to its previous image.

Option key / field	Meaning and accepted input	Default
GUI `guiId`	Choose an authored GUI ID.	Blank
Carousel `elementId`	Choose an element inside the selected GUI.	Blank

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
carousel_previous({"guiId": "hud", "elementId": "topline"})
```

In use. Wire a left-arrow button to the chosen carousel. Wrapping, animation and layout remain properties of that carousel.

carousel_next()

Carousel: Next Image | Action ID: `CarouselNext`

Move a carousel to its next image.

Option key / field	Meaning and accepted input	Default
GUI `guiId`	Choose an authored GUI ID.	Blank
Carousel `elementId`	Choose an element inside the selected GUI.	Blank

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
carousel_next({"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.

save_game_widget_command()

Save Game Widget: Command | Action ID: SaveGameWidgetCommand

Control a save widget's navigation, selection or selected/explicit-slot operations.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
save_game_widget_command({"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.

center_gui_model()

Center GUI Model | Action ID: `CenterGUIModel`

Fit and center the model displayed inside a GUI 3D model viewer.

Option key / field	Meaning and accepted input	Default
GUI `guiId`	Choose an authored GUI ID.	Blank
3D Model `elementId`	Choose an element inside the selected GUI.	Blank

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
center_gui_model({"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.

play_gui_model_animation()

Play GUI Model Animation | Action ID: `PlayGUIModelAnimation`

Play a named animation in a GUI model viewer, with loop and speed controls.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
play_gui_model_animation({"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.

set_runtime_controls_enabled()

Set Runtime Controls Enabled | Action ID: `SetRuntimeControlsEnabled`

Enable or disable the runtime control-profile system.

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

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
set_runtime_controls_enabled({"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.

set_movement_profile()

Set Movement Profile | Action ID: `SetMovementProfile`

Choose legacy, point-and-click, direct, first-person or drag-to-move movement.

Option key / field	Meaning and accepted input	Default
Movement Profile `profile`	Choose: legacy, point-and-click, direct, first-person, drag-to-move.	"point-and-click"

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
set_movement_profile({"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.

set_interaction_mode()

Set Interaction Mode | Action ID: `SetInteractionMode`

Choose the runtime hotspot interaction style, including context, verb, direct, cycling cursor and verb-coin variants.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
set_interaction_mode({"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.

set_auto_input_switching()

Automatic Input Switching | Action ID: `SetAutoInputSwitching`

Enable or disable automatic input-device switching.

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

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
set_auto_input_switching({"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.

set_input_device()

Set Input Device | Action ID: `SetInputDevice`

Select auto, mouse-keyboard, gamepad or touch input mode.

Option key / field	Meaning and accepted input	Default
Input Device `device`	Choose: auto, mouse-keyboard, gamepad, touch.	"auto"

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
set_input_device({"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.

set_proximity_interaction_enabled()

Proximity Interaction | Action ID: `SetProximityInteractionEnabled`

Enable or disable nearest/proximity-based interaction.

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

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
set_proximity_interaction_enabled({"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.

set_control_setting()

Set Control Setting | Action ID: `SetControlSetting`

Change one supported runtime control setting with a value of the matching type.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
set_control_setting({"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.

get_control_state()

Get Control State | Action ID: `GetControlState`

Read the current control-profile state and store it in a game variable.

Option key / field	Meaning and accepted input	Default
Result Variable `resultVariable`	Name of the game variable that receives the returned value. Custom values are accepted.	Blank

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
get_control_state({"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.

open_interaction_coin()

Open Interaction Coin | Action ID: `OpenInteractionCoin`

Open the interaction coin for a target entity when that feature is configured.

Option key / field	Meaning and accepted input	Default
Object / Character `entityId`	Choose a scene entity ID. May be blank.	Blank

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
open_interaction_coin({"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.

close_interaction_coin()

Close Interaction Coin | Action ID: `CloseInteractionCoin`

Close the currently open interaction coin.

No options in the standard shared schema.

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
close_interaction_coin({})
```

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.

cycle_interaction_verb()

Cycle Verb | Action ID: `CycleInteractionVerb`

Move forward or backward through the runtime verb cycle.

Option key / field	Meaning and accepted input	Default
Direction `direction`	Whole number. Range: -1 to 1.	1

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
cycle_interaction_verb({"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.

interact_nearest()

Interact with Nearest Hotspot | Action ID: `InteractNearest`

Perform a verb interaction with the nearest eligible hotspot under the proximity system.

Option key / field	Meaning and accepted input	Default
Verb `verbId`	Choose a Verb Designer ID. May be blank.	Blank

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
interact_nearest({"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.

set_mouse_clicks_enabled()

Enable Mouse Clicks | Action ID: `SetMouseClicksEnabled`

Enable or disable the extension's mouse-click interaction switch.

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

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
set_mouse_clicks_enabled({"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.

set_mouse_pointing_enabled()

Enable Mouse Pointing | Action ID: `SetMousePointingEnabled`

Enable or disable mouse pointing/hover behavior through the extension.

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

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
set_mouse_pointing_enabled({"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.

set_mouse_pointer_visible()

Show Mouse Pointer | Action ID: `SetMousePointerVisible`

Show or hide the mouse pointer.

Option key / field	Meaning and accepted input	Default
Visible `visible`	Explicit visible/hidden state.	true

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
set_mouse_pointer_visible({"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.

change_verb()

Change Current Verb Command | Action ID: `ChangeVerb`

Select a verb from the project's Verb Designer.

Option key / field	Meaning and accepted input	Default
New Verb `verbId`	Choose a Verb Designer ID.	Blank

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
change_verb({"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.

set_verb_usage_enabled()

Enable / Disable Verb Command Usage | Action ID: `SetVerbUsageEnabled`

Enable or disable verb usage at runtime.

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

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
set_verb_usage_enabled({"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.

change_game_cursor()

Change Game Cursor | Action ID: `ChangeGameCursor`

Change the runtime cursor to an authored cursor definition.

Option key / field	Meaning and accepted input	Default
Cursor `cursorId`	Choose a Cursor Designer ID.	Blank

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
change_game_cursor({"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.

play_sound()

Play Sound | Action ID: `PlaySound`

Start an imported audio asset as a sound effect, optionally on a named channel.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
play_sound({"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.

stop_sound()

Stop Sound | Action ID: `StopSound`

Stop effects selected by channel.

Option key / field	Meaning and accepted input	Default
Channel / ID `channel`	Sound-effect channel name. Reuse it to pause, stop or fade that sound group.	Blank

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
stop_sound({"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.

pause_sound()

Sound: Pause | Action ID: `PauseSound`

Pause effects selected by channel.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
pause_sound({"channel": "puzzle"})
```

In use. Pause a repeating machine sound during an inspection that freezes its motion. Pausing is reversible; stopping ends playback instead.

resume_sound()

Sound: Resume | Action ID: ResumeSound

Resume paused effects selected by channel.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
resume_sound({"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.

fade_sound()

Sound: Fade | Action ID: FadeSound

Fade a sound channel to a normalized volume over time, waiting for the fade routine.

Option key / field	Meaning and accepted 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.

```
fade_sound({"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.

play_music()

Play Music | Action ID: `PlayMusic`

Start the chosen audio asset as music.

Option key / field	Meaning and accepted input	Default
Music Asset `assetId`	Choose an imported audio asset.	Blank
Skippable `skippable`	On/off.	true

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
play_music({"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.

stop_music()

Stop Music | Action ID: `StopMusic`

Stop the current music track.

No options in the standard shared schema.

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
stop_music({})
```

In use. Stop the room theme before a silent reveal. It is music-specific and leaves separate effects and voice alone.

pause_music()

Music: Pause | Action ID: `PauseMusic`

Pause the current music track.

No options in the standard shared schema.

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
pause_music({})
```

In use. Pause music while an in-game device temporarily takes over the soundtrack. The track position is kept for Resume Music.

resume_music()

Music: Resume | Action ID: `ResumeMusic`

Resume the paused music track.

No options in the standard shared schema.

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
resume_music({})
```

In use. Use it after the device view closes. If another action replaced the track, the old track is not automatically restored.

fade_music()

Music: Fade | Action ID: FadeMusic

Fade the current music to a normalized volume, waiting for the fade routine.

Option key / field	Meaning and accepted 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.

```
fade_music({"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.

seek_music()

Skip Ahead in Soundtrack | Action ID: SeekMusic

Move playing music to minutes plus seconds.

Option key / field	Meaning and accepted input	Default
Minute `minutes`	Whole minutes; added to Seconds to form a playback position.	0
Second `seconds`	Number; decimals allowed.	0

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
seek_music({"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.

stop_startup_splash_audio()

Stop Startup Splash Audio | Action ID: `StopStartupSplashAudio`

Stop audio still playing from startup splash presentation.

No options in the standard shared schema.

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
stop_startup_splash_audio({})
```

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.

play_video()

Play Video | Action ID: `PlayVideo`

Start an imported video with full-screen, skippable and muted options.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
play_video({"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.

stop_video()

Stop Video | Action ID: `StopVideo`

Stop the current video playback.

No options in the standard shared schema.

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
stop_video({})
```

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.

seek_video()

Seek Video | Action ID: `SeekVideo`

Seek the current video to minutes plus seconds.

Option key / field	Meaning and accepted input	Default
Minute `minutes`	Whole minutes; added to Seconds to form a playback position.	0
Second `seconds`	Number; decimals allowed.	0

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
seek_video({"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.

start_fmv()

Start FMV | Action ID: `StartFMV`

Start the extension's FMV playback route with an end, point or interrupt continuation mode and optional script.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
start_fmv({"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.

stop_fmv()

Stop FMV | Action ID: `StopFMV`

Stop the current FMV playback route.

No options in the standard shared schema.

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
stop_fmv({})
```

In use. Use it when leaving an FMV sequence entirely. Decide what room, interface or script takes over after cancellation.

pause_fmv()

Pause FMV | Action ID: PauseFMV

Pause the current FMV route.

No options in the standard shared schema.

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
pause_fmv({})
```

In use. Pause the clip while a decision is visible. This does not pause every other game system automatically.

resume_fmv()

Resume FMV | Action ID: ResumeFMV

Resume FMV with the chosen continuation mode, endpoint and script.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
resume_fmv({"mode": "end", "endTime": 0, "scriptId": "utility_pca"})
```

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.

seek_fmv()

Seek FMV | Action ID: SeekFMV

Move FMV to a time and set its continuation behavior.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
seek_fmv({"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.

start_gameplay_recording()

Start Gameplay Recording | Action ID: `StartGameplayRecording`

Begin the extension's gameplay-recording route using a writable JSON path.

Option key / field	Meaning and accepted input	Default
Recording Path `path`	Writable recording-data path, such as user://walkthrough.json.	"user://walkthrough.json"

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
start_gameplay_recording({"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.

stop_gameplay_recording()

Stop Gameplay Recording | Action ID: `StopGameplayRecording`

Stop the active gameplay recording.

No options in the standard shared schema.

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
stop_gameplay_recording({})
```

In use. End the test run cleanly before starting another recording. It does not stop a separate operating-system video capture.

play_gameplay_recording()

Play Gameplay Recording | Action ID: `PlayGameplayRecording`

Play a saved gameplay recording at the chosen speed.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
play_gameplay_recording({"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.

stop_gameplay_playback()

Stop Gameplay Playback | Action ID: `StopGameplayPlayback`

Stop playback of recorded gameplay data.

No options in the standard shared schema.

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
stop_gameplay_playback({})
```

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.

start_quest()

Start Quest | Action ID: `StartQuest`

Start a quest and optionally select its initial stage.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
start_quest({"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.

set_quest_stage()

Quest: Set Stage | Action ID: `SetQuestStage`

Move a quest to a chosen stage.

Option key / field	Meaning and accepted input	Default
Quest `questId`	Choose a Quest Designer ID.	Blank
Stage `stageId`	Choose a stage in that quest.	Blank

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
set_quest_stage({"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.

complete_objective()

Quest: Complete Objective | Action ID: `CompleteObjective`

Complete a named objective, optionally selecting a stage first.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
complete_objective({"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.

fail_objective()

Quest: Fail Objective | Action ID: `FailObjective`

Mark an objective as failed, optionally selecting its stage first.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
fail_objective({"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.

reset_objective()

Quest: Reset Objective | Action ID: `ResetObjective`

Reset one objective to its unfinished state, optionally selecting its stage first.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
reset_objective({"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.

complete_quest()

Quest: Complete Quest | Action ID: `CompleteQuest`

Mark the quest completed.

Option key / field	Meaning and accepted input	Default
Quest `questId`	Choose a Quest Designer ID.	Blank

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
complete_quest({"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.

fail_quest()

Quest: Fail Quest | Action ID: `FailQuest`

Mark the quest failed.

Option key / field	Meaning and accepted input	Default
Quest `questId`	Choose a Quest Designer ID.	Blank

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
fail_quest({"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.

reset_quest()

Quest: Reset Quest | Action ID: `ResetQuest`

Reset the quest state through the quest manager.

Option key / field	Meaning and accepted input	Default
Quest `questId`	Choose a Quest Designer ID.	Blank

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
reset_quest({"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.

set_quest_hidden()

Quest: Hide / Unhide | Action ID: `SetQuestHidden`

Set a quest's hidden state.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
set_quest_hidden({"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.

set_quest_tracked()

Quest: Track / Untrack | Action ID: `SetQuestTracked`

Set whether a quest is tracked.

Option key / field	Meaning and accepted input	Default
Quest `questId`	Choose a Quest Designer ID.	Blank
Tracked `tracked`	On/off.	true

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
set_quest_tracked({"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.

change_quest_property()

Quest: Change Property | Action ID: `ChangeQuestProperty`

Change a supported quest property such as title, description, category or flags.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
let options = {}
options["questId"] = ""
options["property"] = "description"
options["value"] = "The signal has been restored."
change_quest_property(options)
```

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.

change_quest_stage_property()

Quest: Change Stage Property | Action ID: `ChangeQuestStageProperty`

Change a supported property's value on one quest stage.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
change_quest_stage_property({"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.

change_quest_objective_property()

Quest: Change Objective Property | Action ID: `ChangeQuestObjectiveProperty`

Change a supported property such as text, optional or hidden on one objective.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
let options = {}
options["questId"] = ""
options["stageId"] = ""
options["objectiveId"] = ""
options["property"] = "text"
change_quest_objective_property(options)
```

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.

save_game()

Save Game | Action ID: SaveGame

Save through the basic runtime route. With no structured slot supplied, this route uses slot 0.

No options in the standard shared schema.

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
save_game({})
```

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.

load_game()

Load Game | Action ID: LoadGame

Load through the basic runtime route, defaulting to slot 0.

No options in the standard shared schema.

Timing. Awaits loading.

```
load_game({})
```

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.

save_game_slot()

Save Game Slot | Action ID: SaveGameSlot

Save the current game into a chosen slot with an optional name.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
save_game_slot({"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.

load_game_slot()

Load Game Slot | Action ID: LoadGameSlot

Load the chosen save slot.

Option key / field	Meaning and accepted input	Default
Slot slot	Explicit save slot number. Selected-slot widget commands use widget selection instead.	0

Timing. Awaits loading.

```
load_game_slot({"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.

set_save_slot_image()

Set Save Slot Image | Action ID: `SetSaveSlotImage`

Set the image override used for save-slot presentation through the extension.

Option key / field	Meaning and accepted input	Default
Save Slot Image `assetId`	Choose an imported image.	Blank

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
set_save_slot_image({"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.

set_game_language()

Set Game Language | Action ID: `SetGameLanguage`

Set the game text language, optionally persisting the selection.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
set_game_language({"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.

change_voice_language()

Change Voice Language | Action ID: ChangeVoiceLanguage

Set the runtime voice language.

Option key / field	Meaning and accepted input	Default
Language language	Language code used by the project, such as en.	Blank

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
change_voice_language({"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.

change_subtitle_language()

Change Subtitle Language | Action ID: ChangeSubtitleLanguage

Set the runtime subtitle language.

Option key / field	Meaning and accepted input	Default
Language language	Language code used by the project, such as en.	Blank

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
change_subtitle_language({"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.

change_project_property()

Change Project Property | Action ID: `ChangeProjectProperty`

Assign a supported project property through a property path.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
change_project_property({"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.

change_game_setting()

Change Game Setting | Action ID: `ChangeGameSetting`

Assign a supported runtime game setting.

Option key / field	Meaning and accepted 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. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
change_game_setting({"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.

start_cutscene()

Cutscene: Start | Action ID: `StartCutscene`

Start a Cutscene Timeline record and wait for its session.

Option key / field	Meaning and accepted input	Default
Cutscene `cutsceneId`	Choose a Cutscene Timeline ID.	Blank

Timing. Waits for the cutscene session.

```
start_cutscene({"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.

end_cutscene()

Cutscene: End | Action ID: `EndCutscene`

End the active cutscene, optionally restoring the camera.

Option key / field	Meaning and accepted input	Default
Restore Previous Camera `restoreCamera`	Restore the cutscene's previous camera when ending.	true

Timing. The call waits for its runtime routine to return; no additional completion wait is promised by this schema.

```
end_cutscene({"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.

Designer access: namespaces and property rules

This section lists all 19 direct namespaces, all property paths supplied by the PCA Script property catalog for those record kinds, and the complete record-method API. The catalog is supplemented by the GUI/project/setting keys exposed by shared property pickers.

Namespace	Aliases
characters	character
objects	object, entity, entities
triggers	trigger
rooms	room, scene, scenes
inventory	item, items
guis	gui
dialogues	dialogue
quests	quest
objectives	objective
assets	asset, sound, sounds, music
cameras	camera
spawns	spawn
cursors	cursor
verbs	verb
fonts	font
scripts	script
lua_scripts	lua_script, lua
basic_scripts	basic_script, basic
visual_scripts	visual_script

A row’s path is appended to the record reference. For example, a path of transform.position under objects.archive_door becomes objects.archive_door.transform.position. Root paths marked Container expose nested data, not a callable method.

R = read-only; W = writable through the record bridge. Live = catalogued for immediate runtime application; Definition = stored data/configuration without that guarantee. Raw field = use the nested-effects workaround after the namespace tables, rather than direct dotted PCA access.

Designer: Characters

These are shared character definitions. A room actor can have a separate character dictionary under an object instance. Model/state changes on the shared definition are not a guarantee of an immediate rebuild of an already loaded actor.

Reference shape: `characters.elias.property_path`.

The example ID is illustrative. Choose an existing record in your own project; direct camera, spawn and objective lookups must be unambiguous.

Methods on Characters

say(text, portrait_id, voice_asset_id, voice_key)

Speak dialogue text as this character.

Accepts 1 to 4 arguments.

walk_to(destination)

Walk this character to an object, character, spawn, Vector2, or Vector3.

Accepts 1 to 1 arguments.

change_room(room, spawn = null)

Change to a room, optionally at a spawn point.

Accepts 1 to 2 arguments.

> This method routes a ChangeScene action for the active game. It is not an NPC-transfer method tied to the receiver. Use Move NPC to Room for an NPC that should move between rooms without changing the active room.

play_animation(animation)

Play an animation state or clip on this character.

Accepts 1 to 1 arguments.

Property 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.

Property path	Type / access	Purpose and limits
`description`	text W / Definition	Character description.
`assetId`	record W / Definition	Model asset used by this character. Reference 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.
`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.

Property path	Type / access	Purpose and limits
`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. Reference 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.
`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.

Property path	Type / access	Purpose and limits
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. Reference kind: guis.
character.personalInventoryGuiId	record W / Definition	Inventory GUI used by this playable character. Reference kind: guis.
character.personalJournalGuiId	record W / Definition	Journal GUI used by this playable character. Reference kind: guis.
character.personalDialogueGuiId	record W / Definition	Dialogue GUI used by this playable character. Reference 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.

Property path	Type / access	Purpose and limits
`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.
`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.

Property path	Type / access	Purpose and limits
`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. Reference kind: cursors.
`pointerFeedback.leftClickCursorId`	record W / Definition	Cursor used for left click. Reference kind: cursors.
`pointerFeedback.rightClickCursorId`	record W / Definition	Cursor used for right click. Reference kind: cursors.
`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.
`character.follow.enabled`	boolean W / Definition	Whether persistent following is enabled.

Property path	Type / access	Purpose and limits
`character.follow.targetId`	record W / Definition	Character to follow. Reference 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

Objects include placed models, primitives, characters, effects and other scene entity kinds. A property shown here may only be useful on the matching kind. The nested character entries apply to character actors, not every door or light.

Reference shape: `objects.archive_door.property_path`.

The example ID is illustrative. Choose an existing record in your own project; direct camera, spawn and objective lookups must be unambiguous.

Methods on Objects

play_animation(animation)

Play an animation state or clip on this object.

Accepts 1 to 1 arguments.

Property 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. Reference 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.

Property path	Type / access	Purpose and limits
`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.
`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.

Property path	Type / access	Purpose and limits
`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.
`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.

Property path	Type / access	Purpose and limits
`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.
`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. Reference kind: cursors.
`pointerFeedback.leftClickCursorId`	record W / Definition	Cursor used for left click. Reference kind: cursors.
`pointerFeedback.rightClickCursorId`	record W / Definition	Cursor used for right click. Reference 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.

Property path	Type / access	Purpose and limits
`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.
`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. Reference kind: assets.

Property path	Type / access	Purpose and limits
`glow.variable`	text W / Definition	Variable used to control glow.
`glow.startOn`	boolean W / Definition	Initial variable-controlled glow state.
`shader.preset`	text Raw field / use action	Use heat-haze for hot-air refraction. Explicit layers take priority over this legacy single preset. Read the parent dictionary, then index the field. For entity writes, use the raw property action.
`shader.layers`	array Raw field / use action	Ordered effects. Heat Haze is heat-haze; each layer has preset and intensity. Local heat surfaces use heat alone. Read the parent dictionary, then index the field. For entity writes, use the raw property action.
`shader.intensity`	number Raw field / use action	Strength of a single-preset effect. Zero removes heat distortion; layer intensity overrides this value. Read the parent dictionary, then index the field. For entity writes, use the raw property action.
`shader.distortionAmount`	number Raw field / use action	Heat refraction distance in screen UV units. Start at 0.02; zero removes distortion. Read the parent dictionary, then index the field. For entity writes, use the raw property action.
`shader.distortionSpeed`	number Raw field / use action	Rising heat speed. Start at 1; zero freezes movement. Read the parent dictionary, then index the field. For entity writes, use the raw property action.
`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.

Property path	Type / access	Purpose and limits
`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 Raw field / use action	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. Reference kind: assets. Read the parent dictionary, then index the field. For entity writes, use the raw property action.
`particle.enabled`	boolean Raw field / use action	Enable lightning playback. Enabling after disabling restarts the flash. Read the parent dictionary, then index the field. For entity writes, use the raw property action.
`particle.oneShot`	boolean Raw field / use action	Play one flash instead of repeating. Read the parent dictionary, then index the field. For entity writes, use the raw property action.
`particle.ribbon`	boolean Raw field / use action	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. Read the parent dictionary, then index the field. For entity writes, use the raw property action.
`particle.color`	text Raw field / use action	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. Read the parent dictionary, then index the field. For entity writes, use the raw property action.
`particle.emissionColor`	text Raw field / use action	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. Read the parent dictionary, then index the field. For entity writes, use the raw property action.
`particle.opacity`	number Raw field / use action	Particle opacity from 0 to 1. Read the parent dictionary, then index the field. For entity writes, use the raw property action.
`particle.direction`	vector3 Raw field / use action	Bolt direction relative to the emitter. Read the parent dictionary, then index the field. For entity writes, use the raw property action.

Property path	Type / access	Purpose and limits
particle.boltWidth	number Raw field / use action	Lightning thickness in scene units. The bright core is one third of this width. Read the parent dictionary, then index the field. For entity writes, use the raw property action.
particle.boltLength	number Raw field / use action	Distance from the emitter to the bolt tip in scene units, along Direction. Transform scale also affects the result. Read the parent dictionary, then index the field. For entity writes, use the raw property action.
particle.flashDuration	number Raw field / use action	Seconds each flash stays visible. One Shot plays one flash; disable and enable Emit In Game to replay it. Read the parent dictionary, then index the field. For entity writes, use the raw property action.
particle.flashInterval	number Raw field / use action	Dark time in seconds between flashes. Zero produces a continuous bolt. Each new flash gets a new jagged shape. Read the parent dictionary, then index the field. For entity writes, use the raw property action.
particle.boltJaggedness	number Raw field / use action	Sideways displacement in scene units. Zero makes a straight beam. Start and end remain fixed. Read the parent dictionary, then index the field. For entity writes, use the raw property action.
particle.boltSegments	number Raw field / use action	Number of lightning segments, from 2 to 128. Fewer segments reduce rendering work. Read the parent dictionary, then index the field. For entity writes, use the raw property action.
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.
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.

Property path	Type / access	Purpose and limits
`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.
`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. Reference kind: guis.

Property path	Type / access	Purpose and limits
`character.personalInventoryGuiId`	record W / Definition	Inventory GUI used by this playable character. Reference kind: guis.
`character.personalJournalGuiId`	record W / Definition	Journal GUI used by this playable character. Reference kind: guis.
`character.personalDialogueGuiId`	record W / Definition	Dialogue GUI used by this playable character. Reference 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. Reference 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

Triggers are a filtered view of scene entities. Event volume and blocking/collision dimensions have different jobs. A trigger does not gain a new collision shape just because a custom key has the same name as a physics setting.

Reference shape: `triggers.welcome_trigger.property_path`.

The example ID is illustrative. Choose an existing record in your own project; direct camera, spawn and objective lookups must be unambiguous.

Methods on Triggers

play_animation(animation)

Play an animation state or clip on this object.

Accepts 1 to 1 arguments.

Property 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. Reference 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.

Property path	Type / access	Purpose and limits
`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.
`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.

Property path	Type / access	Purpose and limits
`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.
`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.

Property path	Type / access	Purpose and limits
`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.
`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. Reference kind: cursors.
`pointerFeedback.leftClickCursorId`	record W / Definition	Cursor used for left click. Reference kind: cursors.
`pointerFeedback.rightClickCursorId`	record W / Definition	Cursor used for right click. Reference 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.

Property path	Type / access	Purpose and limits
`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.
`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. Reference kind: assets.
`glow.variable`	text W / Definition	Variable used to control glow.

Property path	Type / access	Purpose and limits
`glow.startOn`	boolean W / Definition	Initial variable-controlled glow state.
`shader.preset`	text Raw field / use action	Use heat-haze for hot-air refraction. Explicit layers take priority over this legacy single preset. Read the parent dictionary, then index the field. For entity writes, use the raw property action.
`shader.layers`	array Raw field / use action	Ordered effects. Heat Haze is heat-haze; each layer has preset and intensity. Local heat surfaces use heat alone. Read the parent dictionary, then index the field. For entity writes, use the raw property action.
`shader.intensity`	number Raw field / use action	Strength of a single-preset effect. Zero removes heat distortion; layer intensity overrides this value. Read the parent dictionary, then index the field. For entity writes, use the raw property action.
`shader.distortionAmount`	number Raw field / use action	Heat refraction distance in screen UV units. Start at 0.02; zero removes distortion. Read the parent dictionary, then index the field. For entity writes, use the raw property action.
`shader.distortionSpeed`	number Raw field / use action	Rising heat speed. Start at 1; zero freezes movement. Read the parent dictionary, then index the field. For entity writes, use the raw property action.
`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.

Property path	Type / access	Purpose and limits
motion.variable	text W / Definition	Variable used to control motion.
motion.startOn	boolean W / Definition	Initial variable-controlled motion state.
particle.imageAssetId	record Raw field / use action	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. Reference kind: assets. Read the parent dictionary, then index the field. For entity writes, use the raw property action.
particle.enabled	boolean Raw field / use action	Enable lightning playback. Enabling after disabling restarts the flash. Read the parent dictionary, then index the field. For entity writes, use the raw property action.
particle.oneShot	boolean Raw field / use action	Play one flash instead of repeating. Read the parent dictionary, then index the field. For entity writes, use the raw property action.
particle.ribbon	boolean Raw field / use action	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. Read the parent dictionary, then index the field. For entity writes, use the raw property action.
particle.color	text Raw field / use action	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. Read the parent dictionary, then index the field. For entity writes, use the raw property action.
particle.emissionColor	text Raw field / use action	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. Read the parent dictionary, then index the field. For entity writes, use the raw property action.
particle.opacity	number Raw field / use action	Particle opacity from 0 to 1. Read the parent dictionary, then index the field. For entity writes, use the raw property action.
particle.direction	vector3 Raw field / use action	Bolt direction relative to the emitter. Read the parent dictionary, then index the field. For entity writes, use the raw property action.
particle.boltWidth	number Raw field / use action	Lightning thickness in scene units. The bright core is one third of this width. Read the parent dictionary, then index the field. For entity writes, use the raw property action.

Property path	Type / access	Purpose and limits
`particle.boltLength`	number Raw field / use action	Distance from the emitter to the bolt tip in scene units, along Direction. Transform scale also affects the result. Read the parent dictionary, then index the field. For entity writes, use the raw property action.
`particle.flashDuration`	number Raw field / use action	Seconds each flash stays visible. One Shot plays one flash; disable and enable Emit In Game to replay it. Read the parent dictionary, then index the field. For entity writes, use the raw property action.
`particle.flashInterval`	number Raw field / use action	Dark time in seconds between flashes. Zero produces a continuous bolt. Each new flash gets a new jagged shape. Read the parent dictionary, then index the field. For entity writes, use the raw property action.
`particle.boltJaggedness`	number Raw field / use action	Sideways displacement in scene units. Zero makes a straight beam. Start and end remain fixed. Read the parent dictionary, then index the field. For entity writes, use the raw property action.
`particle.boltSegments`	number Raw field / use action	Number of lightning segments, from 2 to 128. Fewer segments reduce rendering work. Read the parent dictionary, then index the field. For entity writes, use the raw property action.

Designer: Rooms

Room data includes cameras, navigation, rendering, stage settings and nested arrays. Changes to definition data can require a room reload. Prefer dedicated camera, transition and shader actions when you need a controlled immediate effect.

Reference shape: `rooms.lab.property_path`.

The example ID is illustrative. Choose an existing record in your own project; direct camera, spawn and objective lookups must be unambiguous.

Property 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. Aliases: title.
`description`	text W / Definition	Room description.
`folder`	text W / Definition	Editor organization folder.
`defaultCameraId`	record W / Definition	Camera selected when the room loads. Reference 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.

Property path	Type / access	Purpose and limits
navigation	dictionary W / Definition / Container	Walkable areas and navigation-agent settings.
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.

Property path	Type / access	Purpose and limits
`environment.volumetricFogEnabled`	boolean W / Definition	Enable volumetric fog in Forward+.
`environment.volumetricFogDensity`	number W / Definition	Volumetric fog density from 0 to 1.
`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.

Property path	Type / access	Purpose and limits
`viewport.width`	integer W / Definition	Room viewport width.
`viewport.height`	integer W / Definition	Room viewport height.
`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. Reference 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.

Property path	Type / access	Purpose and limits
`navigation.agentRadius`	number W / Definition	Default navigation-agent radius.
`navigation.type`	text W / Definition	Plane or polygon navigation area type.
`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 Raw field / use action	Use heat-haze for hot-air refraction. Explicit layers take priority over this legacy single preset. Read the parent dictionary, then index the field. For entity writes, use the raw property action.
`environment.postProcess.layers`	array Raw field / use action	Ordered effects. Heat Haze is heat-haze; each layer has preset and intensity. Local heat surfaces use heat alone. Read the parent dictionary, then index the field. For entity writes, use the raw property action.
`environment.postProcess.intensity`	number Raw field / use action	Strength of a single-preset effect. Zero removes heat distortion; layer intensity overrides this value. Read the parent dictionary, then index the field. For entity writes, use the raw property action.
`environment.postProcess.distortionAmount`	number Raw field / use action	Heat refraction distance in screen UV units. Start at 0.02; zero removes distortion. Read the parent dictionary, then index the field. For entity writes, use the raw property action.
`environment.postProcess.distortionSpeed`	number Raw field / use action	Rising heat speed. Start at 1; zero freezes movement. Read the parent dictionary, then index the field. For entity writes, use the raw property action.

Designer: Inventory

An inventory definition describes an item. It does not itself say how many the player owns. Use inventory_count, has_inventory and the inventory actions to change bags.

Reference shape: `inventory.key.property_path`.

The example ID is illustrative. Choose an existing record in your own project; direct camera, spawn and objective lookups must be unambiguous.

Property 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. Reference 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.

Property path	Type / access	Purpose and limits
`sortOrder`	integer W / Definition	Inventory sorting order.
`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. Reference kind: scenes.
`acquireSceneRemoval.entityId`	record W / Definition	Object removed from the room. Reference kind: entities.

Designer: GUIs

Use GUI helpers for whole-window and element updates. guis.hud.elements is an array, not a direct elements namespace. A helper targets the loaded control and can rebuild it; direct nested array mutation is not a reliable live notification route.

Reference shape: `guis.hud.property_path`.

The example ID is illustrative. Choose an existing record in your own project; direct camera, spawn and objective lookups must be unambiguous.

Property 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.
`clickActivationAtMousePosition`	boolean W / Definition	Place click-activated GUI at the mouse.
`clickActivationStopsPlayer`	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.

Property path	Type / access	Purpose and limits
openTransition.type	text W / Definition	Cut, fade, wipe, dissolve, or another authored transition.
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.

Property path	Type / access	Purpose and limits
`windowFrame.opacity`	number W / Definition	Window opacity.
`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. Reference 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: Dialogues

Dialogue node data is nested. Use the dialogue-line/choice actions to make runtime changes where offered, and keep IDs stable. The dialogue graph in the editor is not a general PCA graph-editing API.

Reference shape: `dialogues.elias_intro.property_path`.

The example ID is illustrative. Choose an existing record in your own project; direct camera, spawn and objective lookups must be unambiguous.

Property 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 and active runtime progress are related but separate. Use start_quest, set_quest_stage and completion actions to drive progress, then quest_state/objective_state to query it.

Reference shape: `quests.signal_quest.property_path`.

The example ID is illustrative. Choose an existing record in your own project; direct camera, spawn and objective lookups must be unambiguous.

Property 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 are nested inside quest stages. IDs need to be unambiguous when referenced directly. Actions that take quest, stage and objective IDs make the ownership explicit.

Reference shape: `objectives.find_key.property_path`.

The example ID is illustrative. Choose an existing record in your own project; direct camera, spawn and objective lookups must be unambiguous.

Property 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

These are imported asset records, not arbitrary operating-system files. Changing path/import metadata does not automatically reimport an asset or rebuild loaded resources. Import through the Asset Importer.

Reference shape: `assets.chime.property_path`.

The example ID is illustrative. Choose an existing record in your own project; direct camera, spawn and objective lookups must be unambiguous.

Property 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 are nested in rooms. Give camera IDs project-wide unique names when direct lookup would otherwise be ambiguous. Activate/blend through dedicated actions; use property writes for supported configuration.

Reference shape: `cameras.wide.property_path`.

The example ID is illustrative. Choose an existing record in your own project; direct camera, spawn and objective lookups must be unambiguous.

Property 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.

Property 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

A spawn is an arrival marker inside a room. A room change can select it by ID. For an unambiguous destination, supply both room and spawn through the change-scene route.

Reference shape: `spawns.start.property_path`.

The example ID is illustrative. Choose an existing record in your own project; direct camera, spawn and objective lookups must be unambiguous.

Property 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 choose pointer assets and hotspot offsets. Selecting a cursor at runtime is a separate action. Input-mode logic may later choose another pointer state.

Reference shape: `cursors.walk.property_path`.

The example ID is illustrative. Choose an existing record in your own project; direct camera, spawn and objective lookups must be unambiguous.

Property 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. Reference 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

A verb definition supplies identity and presentation. Selecting a verb does not execute an interaction or create missing object responses.

Reference shape: `verbs.look.property_path`.

The example ID is illustrative. Choose an existing record in your own project; direct camera, spawn and objective lookups must be unambiguous.

Property 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. Reference 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 records describe imported/registered font data. A change to the definition may require controls to rebuild. It is not a system-font installation API.

Reference shape: `fonts.body.property_path`.

The example ID is illustrative. Choose an existing record in your own project; direct camera, spawn and objective lookups must be unambiguous.

Property 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. Reference 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 Scripts

scripts means PCA Script records in this language. Native GDScript records do not use this direct namespace; call them by ID through call_script.

Reference shape: `scripts.utility_pca.property_path`.

The example ID is illustrative. Choose an existing record in your own project; direct camera, spawn and objective lookups must be unambiguous.

Property 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 Scripts

Read/editing source metadata is different from invoking a function. Use the shared router to execute Lua, and explicitly preserve module/event conventions in Lua source.

Reference shape: `lua_scripts.math_lua.property_path`.

The example ID is illustrative. Choose an existing record in your own project; direct camera, spawn and objective lookups must be unambiguous.

Property 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 Scripts

BASIC has its own module-local persistence and syntax. Direct record access here does not make those local values visible as PCA properties.

Reference shape: `basic_scripts.math_basic.property_path`.

The example ID is illustrative. Choose an existing record in your own project; direct camera, spawn and objective lookups must be unambiguous.

Property 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 Scripts

A visual script is saved graph data. Use its ID through the visual/script execution route. Exposed functions need a valid entry node and matching definition metadata.

Reference shape: `visual_scripts.open_door.property_path`.

The example ID is illustrative. Choose an existing record in your own project; direct camera, spawn and objective lookups must be unambiguous.

Property 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.

Nested effects: a raw-path workaround

A small group of catalog entries under shader and particle are supplied as relative field names in 0.86.3.4. Direct dotted PCA access can normalize away that group. The tables show their actual data location and label them Raw field so they are not confused with ordinary direct properties.

For reads, fetch the parent dictionary into a local variable and index the field. For a live scene entity write, Change Object Property accepts the full raw property string and bypasses that dotted-name normalization.

```
function run()
    let settings = objects.archive_door.shader
    let amount = settings["distortionAmount"]
    let options = {}
    options["entityId"] = "archive_door"
    options["property"] = "shader.distortionAmount"
    options["value"] = 0.02
    change_object_property(options)
    return amount
end
```

Use an object that actually has the relevant effect. The same pattern works for particle.opacity and the other marked entity fields. For room overlay shader data, read the parent dictionary and update the whole supported container or configure it through the relevant designer/action; a room definition change may still need a rebuild.

> The workaround changes the access route, not renderer support. It does not make an unconfigured door into a particle emitter, and a material color or metal value does not automatically produce a character mirror reflection.

Custom properties and nested containers

Use the editor’s custom-property definitions to add per-record data such as a lock code, inspection count or faction name. Access it below custom, for example objects.archive_door.custom.locked. This keeps your extension data separate from engine fields.

```
function run()
    let locked = objects.archive_door.custom.locked
    if locked
        log("The custom lock flag is set.")
    end
end
```

Declare a custom property before using it. A missing ordinary record property can be a runtime error; it is not a reliable automatic initializer. For saved story progress, prefer a runtime game variable unless you have tested that record/custom state’s save behavior.

Property path matching is case-insensitive where supported. Do not freely add/remove underscores, hyphens or spaces in a property key: aliases are explicit. Use the canonical path in the table. Dynamic dictionary keys and player-entered text are data, not normalized record names.

> Array edits need a deliberate write-back. For a designer-owned array, take a working value, change that value, and assign the whole array back to its parent property, or use a dedicated action. Whether loaded resources rebuild still depends on that parent property.

Shared character-property action keys

These are the exact selectable keys exposed by the shared property picker in this build. Use the matching helper/action with the correct value type. A selected key is not a guarantee that every widget/entity kind implements it or updates without rebuilding.

Key / path	How to use it
`visible`	Boolean true/false. Visibility, interaction enablement and feature configuration are separate states.
`position`	Vector2 for GUI layout; match the actual target property. For 3D object transforms use Vector3 and the record path.
`rotation_degrees`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
`scale`	Vector2 for GUI layout; match the actual target property. For 3D object transforms use Vector3 and the record path.
`walk_speed`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
`character.displayName`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
`character.enabled`	Boolean true/false. Visibility, interaction enablement and feature configuration are separate states.
`character.playable`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
`character.switchEnabled`	Boolean true/false. Visibility, interaction enablement and feature configuration are separate states.

Game setting keys

These are the exact selectable keys exposed by the shared property picker in this build. Use the matching helper/action with the correct value type. A selected key is not a guarantee that every widget/entity kind implements it or updates without rebuilding.

Key / path	How to use it
`renderQuality`	Setting below settings. Read with get_setting(key, fallback), write with set_setting(key, value). Preserve the existing type and use dedicated actions for immediate runtime reconfiguration.
`globalSceneShaders`	Setting below settings. Read with get_setting(key, fallback), write with set_setting(key, value). Preserve the existing type and use dedicated actions for immediate runtime reconfiguration.
`globalSceneShaders.layers`	Container/array data. Prefer a dedicated action for a live update; direct nested changes need whole-container write-back.
`globalSceneShaders.preset`	Setting below settings. Read with get_setting(key, fallback), write with set_setting(key, value). Preserve the existing type and use dedicated actions for immediate runtime reconfiguration.
`globalSceneShaders.intensity`	Setting below settings. Read with get_setting(key, fallback), write with set_setting(key, value). Preserve the existing type and use dedicated actions for immediate runtime reconfiguration.
`globalSceneShaders.psxPixelSize`	Setting below settings. Read with get_setting(key, fallback), write with set_setting(key, value). Preserve the existing type and use dedicated actions for immediate runtime reconfiguration.
`globalSceneShaders.psxColorBits`	A color value / color text in the form expected by this property.
`globalSceneShaders.psxDither`	Setting below settings. Read with get_setting(key, fallback), write with set_setting(key, value). Preserve the existing type and use dedicated actions for immediate runtime reconfiguration.
`globalSceneShaders.psxJitter`	Setting below settings. Read with get_setting(key, fallback), write with set_setting(key, value). Preserve the existing type and use dedicated actions for immediate runtime reconfiguration.
`globalSceneShaders.n64PixelSize`	Setting below settings. Read with get_setting(key, fallback), write with set_setting(key, value). Preserve the existing type and use dedicated actions for immediate runtime reconfiguration.
`globalSceneShaders.n64FilterStrength`	Setting below settings. Read with get_setting(key, fallback), write with set_setting(key, value). Preserve the existing type and use dedicated actions for immediate runtime reconfiguration.

Key / path	How to use it
`globalSceneShaders.horrorFogAmount`	Setting below settings. Read with get_setting(key, fallback), write with set_setting(key, value). Preserve the existing type and use dedicated actions for immediate runtime reconfiguration.
`globalSceneShaders.horrorFogStart`	Setting below settings. Read with get_setting(key, fallback), write with set_setting(key, value). Preserve the existing type and use dedicated actions for immediate runtime reconfiguration.
`globalSceneShaders.horrorFogEnd`	Setting below settings. Read with get_setting(key, fallback), write with set_setting(key, value). Preserve the existing type and use dedicated actions for immediate runtime reconfiguration.
`globalSceneShaders.horrorFogColor`	A color value / color text in the form expected by this property.
`globalSceneShaders.horrorFogGrain`	Setting below settings. Read with get_setting(key, fallback), write with set_setting(key, value). Preserve the existing type and use dedicated actions for immediate runtime reconfiguration.
`globalSceneShaders.horrorFogVignette`	Setting below settings. Read with get_setting(key, fallback), write with set_setting(key, value). Preserve the existing type and use dedicated actions for immediate runtime reconfiguration.
`enhancedRendering.quality`	Setting below settings. Read with get_setting(key, fallback), write with set_setting(key, value). Preserve the existing type and use dedicated actions for immediate runtime reconfiguration.
`enhancedRendering.giEnabled`	Boolean true/false. Visibility, interaction enablement and feature configuration are separate states.
`enhancedRendering.ssrEnabled`	Boolean true/false. Visibility, interaction enablement and feature configuration are separate states.
`enhancedRendering.ssaoEnabled`	Boolean true/false. Visibility, interaction enablement and feature configuration are separate states.
`enhancedRendering.reflectionProbeEnabled`	Boolean true/false. Visibility, interaction enablement and feature configuration are separate states.
`enhancedRendering.giEnergy`	Setting below settings. Read with get_setting(key, fallback), write with set_setting(key, value). Preserve the existing type and use dedicated actions for immediate runtime reconfiguration.
`enhancedRendering.giCellSize`	Setting below settings. Read with get_setting(key, fallback), write with set_setting(key, value). Preserve the existing type and use dedicated actions for immediate runtime reconfiguration.
`enhancedRendering.giReadSky`	Setting below settings. Read with get_setting(key, fallback), write with set_setting(key, value). Preserve the existing type and use dedicated actions for immediate runtime reconfiguration.

Key / path	How to use it
enhancedRendering.giOcclusion	Setting below settings. Read with get_setting(key, fallback), write with set_setting(key, value). Preserve the existing type and use dedicated actions for immediate runtime reconfiguration.
enhancedRendering.aoIntensity	Setting below settings. Read with get_setting(key, fallback), write with set_setting(key, value). Preserve the existing type and use dedicated actions for immediate runtime reconfiguration.
enhancedRendering.reflectionProbeSize	Setting below settings. Read with get_setting(key, fallback), write with set_setting(key, value). Preserve the existing type and use dedicated actions for immediate runtime reconfiguration.
cinematic.enabled	Boolean true/false. Visibility, interaction enablement and feature configuration are separate states.
cinematic.exposure	Setting below settings. Read with get_setting(key, fallback), write with set_setting(key, value). Preserve the existing type and use dedicated actions for immediate runtime reconfiguration.
cinematic.contrast	Setting below settings. Read with get_setting(key, fallback), write with set_setting(key, value). Preserve the existing type and use dedicated actions for immediate runtime reconfiguration.
cinematic.saturation	Setting below settings. Read with get_setting(key, fallback), write with set_setting(key, value). Preserve the existing type and use dedicated actions for immediate runtime reconfiguration.
cinematic.warmth	Setting below settings. Read with get_setting(key, fallback), write with set_setting(key, value). Preserve the existing type and use dedicated actions for immediate runtime reconfiguration.
cinematic.vignette	Setting below settings. Read with get_setting(key, fallback), write with set_setting(key, value). Preserve the existing type and use dedicated actions for immediate runtime reconfiguration.
cinematic.grain	Setting below settings. Read with get_setting(key, fallback), write with set_setting(key, value). Preserve the existing type and use dedicated actions for immediate runtime reconfiguration.
cinematic.letterbox	Setting below settings. Read with get_setting(key, fallback), write with set_setting(key, value). Preserve the existing type and use dedicated actions for immediate runtime reconfiguration.
subtitles	Setting below settings. Read with get_setting(key, fallback), write with set_setting(key, value). Preserve the existing type and use dedicated actions for immediate runtime reconfiguration.
voiceLanguage	Setting below settings. Read with get_setting(key, fallback), write with set_setting(key, value). Preserve the existing type and use dedicated actions for immediate runtime reconfiguration.

Key / path	How to use it
`subtitleLanguage`	Setting below settings. Read with get_setting(key, fallback), write with set_setting(key, value). Preserve the existing type and use dedicated actions for immediate runtime reconfiguration.
`masterVolume`	Normalized number, normally 0 to 1. Check the runtime result and use a dedicated audio action for controlled playback fades.
`musicVolume`	Normalized number, normally 0 to 1. Check the runtime result and use a dedicated audio action for controlled playback fades.
`soundVolume`	Normalized number, normally 0 to 1. Check the runtime result and use a dedicated audio action for controlled playback fades.
`uiScale`	Setting below settings. Read with get_setting(key, fallback), write with set_setting(key, value). Preserve the existing type and use dedicated actions for immediate runtime reconfiguration.
`fontScale`	Setting below settings. Read with get_setting(key, fallback), write with set_setting(key, value). Preserve the existing type and use dedicated actions for immediate runtime reconfiguration.
`highContrast`	Setting below settings. Read with get_setting(key, fallback), write with set_setting(key, value). Preserve the existing type and use dedicated actions for immediate runtime reconfiguration.
`fullscreen`	Setting below settings. Read with get_setting(key, fallback), write with set_setting(key, value). Preserve the existing type and use dedicated actions for immediate runtime reconfiguration.
`showMainMenuOnStart`	Setting below settings. Read with get_setting(key, fallback), write with set_setting(key, value). Preserve the existing type and use dedicated actions for immediate runtime reconfiguration.
`customCursor`	Setting below settings. Read with get_setting(key, fallback), write with set_setting(key, value). Preserve the existing type and use dedicated actions for immediate runtime reconfiguration.
`cursorScale`	Setting below settings. Read with get_setting(key, fallback), write with set_setting(key, value). Preserve the existing type and use dedicated actions for immediate runtime reconfiguration.

GUI element property keys

These are the exact selectable keys exposed by the shared property picker in this build. Use the matching helper/action with the correct value type. A selected key is not a guarantee that every widget/entity kind implements it or updates without rebuilding.

Key / path	How to use it
`visible`	Boolean true/false. Visibility, interaction enablement and feature configuration are separate states.
`position`	Vector2 for GUI layout; match the actual target property. For 3D object transforms use Vector3 and the record path.
`size`	Vector2 for GUI layout; match the actual target property. For 3D object transforms use Vector3 and the record path.
`scale`	Vector2 for GUI layout; match the actual target property. For 3D object transforms use Vector3 and the record path.
`rotation_degrees`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
`modulate`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
`mouse_filter`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
`z_index`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
`windowBorderRadius`	Numeric radius. For corner/part overrides, use the intended part’s key and inheritance value; do not assign one shared value to every part unless that is intended.
`radiusTopLeft`	Numeric radius. For corner/part overrides, use the intended part’s key and inheritance value; do not assign one shared value to every part unless that is intended.
`radiusTopRight`	Numeric radius. For corner/part overrides, use the intended part’s key and inheritance value; do not assign one shared value to every part unless that is intended.
`radiusBottomRight`	Numeric radius. For corner/part overrides, use the intended part’s key and inheritance value; do not assign one shared value to every part unless that is intended.

Key / path	How to use it
radiusBottomLeft	Numeric radius. For corner/part overrides, use the intended part’s key and inheritance value; do not assign one shared value to every part unless that is intended.
windowBackgroundEnabled	Boolean true/false. Visibility, interaction enablement and feature configuration are separate states.
windowBackground	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
windowBorderEnabled	Boolean true/false. Visibility, interaction enablement and feature configuration are separate states.
windowBorderWidth	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
windowBorderColor	A color value / color text in the form expected by this property.
backgroundImageAssetId	Asset ID text. Import the required media and select the ID; a path to a random local file is not an asset reference.
hoverImageAssetId	Asset ID text. Import the required media and select the ID; a path to a random local file is not an asset reference.
pressedImageAssetId	Asset ID text. Import the required media and select the ID; a path to a random local file is not an asset reference.
backgroundFit	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
gridBackgroundAssetId	Asset ID text. Import the required media and select the ID; a path to a random local file is not an asset reference.
gridHoverImageAssetId	Asset ID text. Import the required media and select the ID; a path to a random local file is not an asset reference.
gridPressedImageAssetId	Asset ID text. Import the required media and select the ID; a path to a random local file is not an asset reference.
gridBackgroundFit	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
slotBackgroundImageAssetId	Asset ID text. Import the required media and select the ID; a path to a random local file is not an asset reference.
slotHoverImageAssetId	Asset ID text. Import the required media and select the ID; a path to a random local file is not an asset reference.

Key / path	How to use it
`slotPressedImageAssetId`	Asset ID text. Import the required media and select the ID; a path to a random local file is not an asset reference.
`slotBackgroundFit`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
`saveScreenshotBackgroundImageAssetId`	Asset ID text. Import the required media and select the ID; a path to a random local file is not an asset reference.
`saveScreenshotHoverImageAssetId`	Asset ID text. Import the required media and select the ID; a path to a random local file is not an asset reference.
`saveScreenshotPressedImageAssetId`	Asset ID text. Import the required media and select the ID; a path to a random local file is not an asset reference.
`saveScreenshotFit`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
`gridRadius`	Numeric radius. For corner/part overrides, use the intended part’s key and inheritance value; do not assign one shared value to every part unless that is intended.
`gridRadiusTopLeft`	Numeric radius. For corner/part overrides, use the intended part’s key and inheritance value; do not assign one shared value to every part unless that is intended.
`gridRadiusTopRight`	Numeric radius. For corner/part overrides, use the intended part’s key and inheritance value; do not assign one shared value to every part unless that is intended.
`gridRadiusBottomRight`	Numeric radius. For corner/part overrides, use the intended part’s key and inheritance value; do not assign one shared value to every part unless that is intended.
`gridRadiusBottomLeft`	Numeric radius. For corner/part overrides, use the intended part’s key and inheritance value; do not assign one shared value to every part unless that is intended.
`slotRadius`	Numeric radius. For corner/part overrides, use the intended part’s key and inheritance value; do not assign one shared value to every part unless that is intended.
`slotRadiusTopLeft`	Numeric radius. For corner/part overrides, use the intended part’s key and inheritance value; do not assign one shared value to every part unless that is intended.
`slotRadiusTopRight`	Numeric radius. For corner/part overrides, use the intended part’s key and inheritance value; do not assign one shared value to every part unless that is intended.

Key / path	How to use it
`slotRadiusBottomRight`	Numeric radius. For corner/part overrides, use the intended part’s key and inheritance value; do not assign one shared value to every part unless that is intended.
`slotRadiusBottomLeft`	Numeric radius. For corner/part overrides, use the intended part’s key and inheritance value; do not assign one shared value to every part unless that is intended.
`saveScreenshotRadius`	Numeric radius. For corner/part overrides, use the intended part’s key and inheritance value; do not assign one shared value to every part unless that is intended.
`saveScreenshotRadiusTopLeft`	Numeric radius. For corner/part overrides, use the intended part’s key and inheritance value; do not assign one shared value to every part unless that is intended.
`saveScreenshotRadiusTopRight`	Numeric radius. For corner/part overrides, use the intended part’s key and inheritance value; do not assign one shared value to every part unless that is intended.
`saveScreenshotRadiusBottomRight`	Numeric radius. For corner/part overrides, use the intended part’s key and inheritance value; do not assign one shared value to every part unless that is intended.
`saveScreenshotRadiusBottomLeft`	Numeric radius. For corner/part overrides, use the intended part’s key and inheritance value; do not assign one shared value to every part unless that is intended.
`radius`	Numeric radius. For corner/part overrides, use the intended part’s key and inheritance value; do not assign one shared value to every part unless that is intended.
`sliderThumbAssetId`	Asset ID text. Import the required media and select the ID; a path to a random local file is not an asset reference.
`sliderThumbHoverAssetId`	Asset ID text. Import the required media and select the ID; a path to a random local file is not an asset reference.
`sliderThumbSize`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
`modelInitialView`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
`modelStartDistance`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.

Key / path	How to use it
`modelRotationX`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
`modelRotationY`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
`modelRotationZ`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
`modelPanX`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
`modelPanY`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
`modelFov`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
`modelAnimated`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
`modelAnimation`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
`modelAnimationLoop`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
`modelAnimationSpeed`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
`modelInteractive`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
`modelMouseRotate`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
`modelMousePan`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.

Key / path	How to use it
`modelMouseResize`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.

Images have normal/hover/pressed variants. Pressed is a temporary pointer/button state, not a permanent selected state. Widget part radii/images are independent keys; choosing a whole-element radius does not replace a deliberate per-part override.

Whole GUI property keys

These are the exact selectable keys exposed by the shared property picker in this build. Use the matching helper/action with the correct value type. A selected key is not a guarantee that every widget/entity kind implements it or updates without rebuilding.

Key / path	How to use it
`visible`	Boolean true/false. Visibility, interaction enablement and feature configuration are separate states.
`position`	Vector2 for GUI layout; match the actual target property. For 3D object transforms use Vector3 and the record path.
`size`	Vector2 for GUI layout; match the actual target property. For 3D object transforms use Vector3 and the record path.
`scale`	Vector2 for GUI layout; match the actual target property. For 3D object transforms use Vector3 and the record path.
`rotation_degrees`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
`modulate`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
`mouse_filter`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
`z_index`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
`windowBorderRadius`	Numeric radius. For corner/part overrides, use the intended part’s key and inheritance value; do not assign one shared value to every part unless that is intended.
`radiusTopLeft`	Numeric radius. For corner/part overrides, use the intended part’s key and inheritance value; do not assign one shared value to every part unless that is intended.
`radiusTopRight`	Numeric radius. For corner/part overrides, use the intended part’s key and inheritance value; do not assign one shared value to every part unless that is intended.
`radiusBottomRight`	Numeric radius. For corner/part overrides, use the intended part’s key and inheritance value; do not assign one shared value to every part unless that is intended.

Key / path	How to use it
`radiusBottomLeft`	Numeric radius. For corner/part overrides, use the intended part’s key and inheritance value; do not assign one shared value to every part unless that is intended.
`windowBackgroundEnabled`	Boolean true/false. Visibility, interaction enablement and feature configuration are separate states.
`windowBackground`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
`windowBorderEnabled`	Boolean true/false. Visibility, interaction enablement and feature configuration are separate states.
`windowBorderWidth`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
`windowBorderColor`	A color value / color text in the form expected by this property.
`backgroundImageAssetId`	Asset ID text. Import the required media and select the ID; a path to a random local file is not an asset reference.
`hoverImageAssetId`	Asset ID text. Import the required media and select the ID; a path to a random local file is not an asset reference.
`pressedImageAssetId`	Asset ID text. Import the required media and select the ID; a path to a random local file is not an asset reference.
`backgroundFit`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.

Shared object-property action keys

These are the exact selectable keys exposed by the shared property picker in this build. Use the matching helper/action with the correct value type. A selected key is not a guarantee that every widget/entity kind implements it or updates without rebuilding.

Key / path	How to use it
`visible`	Boolean true/false. Visibility, interaction enablement and feature configuration are separate states.
`position`	Vector2 for GUI layout; match the actual target property. For 3D object transforms use Vector3 and the record path.
`rotation_degrees`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
`scale`	Vector2 for GUI layout; match the actual target property. For 3D object transforms use Vector3 and the record path.
`enabled`	Boolean true/false. Visibility, interaction enablement and feature configuration are separate states.
`transform.position`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
`transform.rotation`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
`transform.scale`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.

Project property paths

These are the exact selectable keys exposed by the shared property picker in this build. Use the matching helper/action with the correct value type. A selected key is not a guarantee that every widget/entity kind implements it or updates without rebuilding.

Key / path	How to use it
`meta.title`	Project-data path. Read/write with get_project_property and set_project_property. This edits runtime data, not the editor’s saved source file.
`meta.version`	Project-data path. Read/write with get_project_property and set_project_property. This edits runtime data, not the editor’s saved source file.
`meta.startSceneId`	Project-data path. Read/write with get_project_property and set_project_property. This edits runtime data, not the editor’s saved source file.
`meta.inventoryMode`	Project-data path. Read/write with get_project_property and set_project_property. This edits runtime data, not the editor’s saved source file.
`meta.interactionMode`	Project-data path. Read/write with get_project_property and set_project_property. This edits runtime data, not the editor’s saved source file.
`settings.controlProfiles.movementProfile`	Project-data path. Read/write with get_project_property and set_project_property. This edits runtime data, not the editor’s saved source file.
`settings.controlProfiles.autoSwitchInput`	Project-data path. Read/write with get_project_property and set_project_property. This edits runtime data, not the editor’s saved source file.
`settings.controlProfiles.inputDevice`	Project-data path. Read/write with get_project_property and set_project_property. This edits runtime data, not the editor’s saved source file.
`settings.controlProfiles.autoSwitchMovement`	Project-data path. Read/write with get_project_property and set_project_property. This edits runtime data, not the editor’s saved source file.
`settings.controlProfiles.proximityInteraction`	Project-data path. Read/write with get_project_property and set_project_property. This edits runtime data, not the editor’s saved source file.
`settings.controlProfiles.coinEnabled`	Boolean true/false. Visibility, interaction enablement and feature configuration are separate states.

Key / path	How to use it
`settings.globalSceneShaders`	Project-data path. Read/write with get_project_property and set_project_property. This edits runtime data, not the editor’s saved source file.
`settings.globalSceneShaders.layers`	Container/array data. Prefer a dedicated action for a live update; direct nested changes need whole-container write-back.
`settings.globalSceneShaders.preset`	Project-data path. Read/write with get_project_property and set_project_property. This edits runtime data, not the editor’s saved source file.
`settings.globalSceneShaders.intensity`	Project-data path. Read/write with get_project_property and set_project_property. This edits runtime data, not the editor’s saved source file.
`settings.globalSceneShaders.psxPixelSize`	Project-data path. Read/write with get_project_property and set_project_property. This edits runtime data, not the editor’s saved source file.
`settings.globalSceneShaders.psxColorBits`	A color value / color text in the form expected by this property.
`settings.globalSceneShaders.psxDither`	Project-data path. Read/write with get_project_property and set_project_property. This edits runtime data, not the editor’s saved source file.
`settings.globalSceneShaders.psxJitter`	Project-data path. Read/write with get_project_property and set_project_property. This edits runtime data, not the editor’s saved source file.
`settings.globalSceneShaders.n64PixelSize`	Project-data path. Read/write with get_project_property and set_project_property. This edits runtime data, not the editor’s saved source file.
`settings.globalSceneShaders.n64FilterStrength`	Project-data path. Read/write with get_project_property and set_project_property. This edits runtime data, not the editor’s saved source file.
`settings.globalSceneShaders.horrorFogAmount`	Project-data path. Read/write with get_project_property and set_project_property. This edits runtime data, not the editor’s saved source file.
`settings.globalSceneShaders.horrorFogStart`	Project-data path. Read/write with get_project_property and set_project_property. This edits runtime data, not the editor’s saved source file.
`settings.globalSceneShaders.horrorFogEnd`	Project-data path. Read/write with get_project_property and set_project_property. This edits runtime data, not the editor’s saved source file.
`settings.globalSceneShaders.horrorFogColor`	A color value / color text in the form expected by this property.

Key / path	How to use it
`settings.globalSceneShaders.horrorFogGrain`	Project-data path. Read/write with get_project_property and set_project_property. This edits runtime data, not the editor’s saved source file.
`settings.globalSceneShaders.horrorFogVignette`	Project-data path. Read/write with get_project_property and set_project_property. This edits runtime data, not the editor’s saved source file.

Quest-objective action keys

These are the exact selectable keys exposed by the shared property picker in this build. Use the matching helper/action with the correct value type. A selected key is not a guarantee that every widget/entity kind implements it or updates without rebuilding.

Key / path	How to use it
`text`	Text. GUI display text can use that GUI's supported wildcard substitution; normal PCA string construction uses format/concatenation.
`textKey`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
`optional`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
`hidden`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.

Quest-property action keys

These are the exact selectable keys exposed by the shared property picker in this build. Use the matching helper/action with the correct value type. A selected key is not a guarantee that every widget/entity kind implements it or updates without rebuilding.

Key / path	How to use it
`title`	Text. GUI display text can use that GUI's supported wildcard substitution; normal PCA string construction uses format/concatenation.
`titleKey`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
`description`	Text. GUI display text can use that GUI's supported wildcard substitution; normal PCA string construction uses format/concatenation.
`descriptionKey`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
`category`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
`hidden`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
`repeatable`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
`autoTrack`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.

Quest-stage action keys

These are the exact selectable keys exposed by the shared property picker in this build. Use the matching helper/action with the correct value type. A selected key is not a guarantee that every widget/entity kind implements it or updates without rebuilding.

Key / path	How to use it
`title`	Text. GUI display text can use that GUI's supported wildcard substitution; normal PCA string construction uses format/concatenation.
`titleKey`	Use with the matching property action/helper. Read its current value to confirm the type, then assign an appropriate value. Type-specific widgets can expose extra behavior.
`description`	Text. GUI display text can use that GUI's supported wildcard substitution; normal PCA string construction uses format/concatenation.

Events: phrases, shared hooks and payloads

The following 18 entries map the shared runtime hooks to PCA source headers. IDs in the readable headers are examples: replace lab, welcome_trigger, elias_intro and media IDs with your own. Each handler ends with end.

PCA when handlers are discovered in enabled source records. They are not the same as merely naming a function after a native handler. Exact-name form, when event Name, matches the event name the runtime dispatches.

> Event context is supplied by the emitter. Check optional keys and inspect event when debugging. return is only legal inside a function, not directly inside when. Put a handler’s work in a function when you need an early return, or guard it with if.

OnGameStart

```
when game starts
    log(to_json(event))
end
```

The runtime reaches its game-start hook. It is a game-session event, not a once-per-room event.

Common context keys	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

```
when room lab loads
    log(to_json(event))
end
```

A room has been constructed and its local entry actions have run.

Common context keys	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

```
when room lab leaves
    log(to_json(event))
end
```

The scene-loading route is leaving one scene for another.

Common context keys	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

```
when player changes
    log(to_json(event))
end
```

The controlled playable character changes.

Common context keys	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

```
when event OnCharacterControl
    log(to_json(event))
end
```

A character becomes the controlled character, including initial control at startup.

Common context keys	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

```
when event OnItemFound
    log(to_json(event))
end
```

The active-inventory add route adds an item that was previously absent from that bag.

Common context keys	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

```
when video intro_movie ends
    log(to_json(event))
end
```

The runtime receives the video-finished callback.

Common context keys	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

```
when music theme ends
    log(to_json(event))
end
```

The music-finished callback reaches the runtime.

Common context keys	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

```
when dialogue elias_intro starts
    log(to_json(event))
end
```

A valid dialogue session begins, before its graph is processed.

Common context keys	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

```
when dialogue elias_intro ends
    log(to_json(event))
end
```

A dialogue session ends, including interrupted endings.

Common context keys	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

```
when inventory changes
    log(to_json(event))
end
```

An inventory mutation route reports an add, remove or quantity change.

Common context keys	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

```
when save loads
    log(to_json(event))
end
```

A save has successfully restored game state.

Common context keys	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

```
when event OnCustomEvent
    log(to_json(event))
end
```

A script or Emit Event graph node explicitly emits this event name.

Common context keys	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.

For your own exact name, emit_event("signal_powered", {...}) pairs with when event signal_powered. To use the standard OnCustomEvent hook, emit that exact hook name and put your discriminator in the data dictionary.

OnGlobalReady

```
when global ready
    log(to_json(event))
end
```

A persistent/global script is initialized through its language runtime's ready route.

Common context keys	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

```
when player interacts
    log(to_json(event))
end
```

A scene-entity verb interaction reaches its execution point.

Common context keys	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

```
when player enters trigger welcome_trigger
    log(to_json(event))
end
```

The active player enters an eligible trigger/interactive-region overlap.

Common context keys	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

```
when player leaves trigger welcome_trigger
    log(to_json(event))
end
```

The active player leaves an eligible trigger/interactive-region overlap.

Common context keys	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

```
when event OnAttachPointContact
    log(to_json(event))
end
```

A named attach-point watch reports entering or leaving point contact.

Common context keys	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.

Event filters and caller context

PCA header	Filter
`when player enters lab`	Room-load hook filtered to lab; also available as when room lab loads.
`when player talks to elias`	Interaction hook, target character elias and talk verb.
`when player interacts using look`	Interaction hook filtered to look.
`when player uses key on archive_door`	Interaction hook filtered by item, target and use verb.
`when event signal_powered`	Exact custom event name; inspect the body’s event data for finer filters.

An interaction event observes a hook; it does not automatically replace the object’s authored interaction response. Do not put a reward both in the response and an unguarded broad listener. Trigger enter/exit reports overlap, not every possible scene collision.

A GUI script caller can provide guiId, elementId and the GUI event name. Parser calls can provide command, cleanedCommand and customCommand. A function invoked manually may have none of these keys. Use a small diagnostic test to see the actual payload before depending on it.

Readable adventure commands

These are the command patterns offered by the editor’s Insert/completion catalog. They complement the shared action functions. Words such as item, character_id and room_id are placeholders for actual project records, not reserved variables you should declare.

The same command can accept an expression where its grammar expects a value. Use normal quoted strings for speech and IDs from completion for records. A start prefix is available on supported non-session action phrases to begin them without waiting; it is not a prefix for every arbitrary function call.

Job	Source pattern
say text	`say ""`
character says text	`player says ""`
give item to character	`give item to player`
remove item from character	`remove item from player`
set property to value	`set property to value`
set snow accumulation	`set snow accumulation to 1 over 30 seconds`
melt accumulated snow	`melt accumulated snow over 5 seconds`
enable automatic interaction approach	`enable interaction approach at distance 0.35`
set object interaction approach override	`set interaction approach for object_id to disabled`
add value to variable	`add 1 to variable`
subtract value from variable	`subtract 1 from variable`
play sound	`play sound sound_id`

Readable commands, continued

Job	Source pattern
stop sound	`stop sound sound_channel`
pause sound	`pause sound sound_channel`
resume sound	`resume sound sound_channel`
fade sound	`fade sound sound_channel to 0 over 1 second`
play music	`play music music_id`
stop music	`stop music`
pause music	`pause music`
resume music	`resume music`
fade music	`fade music to 0 over 1 second`
stop character animation	`stop player animation`
pause character animation	`pause player animation`
resume character animation	`resume player animation`
character walks to target	`player walks to target`
character faces target	`player faces target`
change room	`change room to room_id`
show gui	`show gui gui_id`
hide gui	`hide gui gui_id`
enable gui	`enable gui gui_id`

Readable commands, continued

Job	Source pattern
disable gui	`disable gui gui_id`
start dialogue	`start dialogue dialogue_id`
end dialogue	`end dialogue`
start cutscene	`start cutscene cutscene_id`
end cutscene	`end cutscene`
start quest	`start quest quest_id`
set quest stage	`set quest quest_id stage to stage_id`
complete quest	`complete quest quest_id`
fail quest	`fail quest quest_id`
reset quest	`reset quest quest_id`
complete quest objective	`complete objective objective_id in quest quest_id`
fail quest objective	`fail objective objective_id in quest quest_id`
save game	`save game`
save named game slot	`save game slot 1 as "Save name"`
load game slot	`load game slot 1`
switch active player	`switch player to character_id`
shake camera	`shake camera for 0.5 seconds with power 0.25`
fade screen out	`fade screen out to "#000000" over 0.5 seconds`

Readable commands, continued

Job	Source pattern
fade screen in	`fade screen in from "#000000" over 0.5 seconds`
screen transition	`transition screen using dissolve over 0.8 seconds`
change cursor	`change cursor to cursor_id`
spawn scene object	`spawn object object_id`
despawn scene object	`despawn object object_id`
enable scene object	`enable object object_id`
disable scene object	`disable object object_id`
show scene object	`show object object_id`
hide scene object	`hide object object_id`
emit custom event	`emit event alarm_triggered with {"source": player}`
run visual script	`run visual script script_id`
call script function	`call script script_id function run`
wait seconds	`wait 1 second`
log value	`log ""`
warn value	`warn ""`
assert condition	`assert condition`

Paths, following and record methods

The parser also supports the following readable movement patterns. A marker can be a valid object/spawn reference or a suitable vector position. Use the catalog action functions when you need every optional movement setting.

Pattern	Meaning
`player walks to archive_door`	Navigate to an object/character target.
`move elias to vector3(2, 0, -1)`	Move/teleport to a position; not a navigated walk.
`player walks through marker1, marker2`	Walk a specified path of targets.
`add waypoint marker1 to player`	Queue a waypoint.
`clear waypoints for player`	Clear that actor’s queued waypoints.
`player walks waypoints`	Walk the queued route.
`elias follows player`	Start character following.
`elias stops following`	Stop character following.
`wait until elias animation ends`	Wait for a finite animation to complete.

The direct method API consists of say, walk_to, change_room and play_animation on character references; objects/triggers expose play_animation. This is the complete direct record-method list for this PCA build, not a subset of a larger Godot Node API.

```
function run()
    characters.elias.say("Follow me.")
    characters.elias.walk_to(objects.archive_door)
    characters.elias.play_animation("nod")
end
```

change_room routes the active game’s ChangeScene action. To move an NPC to another room while play stays where it is, use move_npc_to_room. It is unsafe to infer NPC-transfer behavior just from calling the method on an NPC reference.

Language quick reference

Construct	Form / rule
Language header	`language 1.0`
Comments	`# comment` or `// comment`
Text	Single or double quotes; backslash escapes; one source line.
Declarations	`let name: type = value`, `global name = value`, `const NAME = value`
Assignment	`set name to value`, `=`, `+=`, `-=`, `*=`, `/=`
Decision	`if test`, `else if test`, `else`, `end`
Loops	`repeat n times`, `while test`, `for each value, index in array`; close with `end`.
Loop control	`break`, `continue`
Function	`function name(a: type, b = default): type` … `end`
Return	`return value` or bare `return`
Data record	`record Name` with field:type lines, then `end`.
Timing	`wait n seconds`, `wait until condition`, `every n seconds` … `end`.
Concurrency	`parallel` immediate branch instructions … `end`.
Recovery	`try` … `catch problem` … `end`
Event	`when ...` … `end`
Debugging	`log value`, `warn value`, `assert condition`, `breakpoint`
Cross-script	`call_script(id, function, [arguments], context)`; readable call script / run visual script forms.

Optional type names: number, integer, text, boolean, array, dictionary, vector2, vector3, color, record and null, plus a declared data-record type. There are no typed generic array declarations such as Array[String] in PCA source.

Common mistakes and their fixes

Symptom	Likely cause / correction
Unknown function	Use an exact builtin or catalog wrapper. Native-only GiveInventory/HideObject are not automatic PCA give_inventory/hide_object wrappers; use the readable phrase, add_inventory or set_object_visible as appropriate.
Nothing happens after a property write	A definition changed while the loaded object/control still uses built resources. Use the live helper/action or reload/rebuild at a deliberate time.
A drop-down shows the record but the call fails	The action needs the ID and correct kind, not a display label. Duplicate camera/spawn IDs can make a direct reference ambiguous.
Use Definitions appears to do nothing	Add a function and its parameters first. The button accepts metadata; it does not infer source functions.
The text box returns its original text	Read the loaded element’s text through get_gui_element_property in a running game. Designer preview is not runtime event execution.
The timer runs faster every visit	A second periodic job was started. Add a guard and a deliberate stop/restart lifecycle.
The line says true, but the test is wrong	The value may be text "true" instead of Boolean true, or a local may shadow the game variable.
A wait never finishes	The flag is never set, the clip loops, or the expected event is not emitted. Add a bounded wait and log the actual condition.
The reward happens twice	Both a response and a listener do the work, or a completion flag is set after a long wait. Set one-time state early.
It works until load or room re-entry	Transient state/visual changes were not rebuilt from saved game variables.
An array edit does not update a designer object	Write back the whole container or use a dedicated live action.
A file seems unchanged	Write helpers overwrite only user:// files and return a Boolean. Check the path, return value and read-back.

Boundaries and compatibility notes

PCA Script 1.0 is a deliberately small authored language. Use an explicit helper in GDScript, Lua or BASIC when their supported capabilities better fit the job, then pass normal data through the shared router.

Not a PCA 1.0 facility	Use instead
C-style headers, textual imports or macros	Script records and call_script/include_script.
Terminal input / prompt() / input()	GUI text boxes, variable-bound controls or the text parser.
C for-loops, do-while, switch, ternary, ++ or **	for each with range, while, else if, explicit arithmetic.
General array/string object methods	The listed builtins and loops, or a helper function.
Arbitrary native Node methods, networking or OS access	An intentional native helper, written for the target platform.
General file rename/delete/append API	Read-modify-write for supported user-data files; native helper where needed.
Automatic permanent editor document editing	The corresponding designer and project save workflow.
C# / Python authored script routes in this build	The implemented PCA/GDScript/Lua/BASIC/visual routes.

Compatibility details worth keeping nearby: every runs its body immediately before the first wait; include_script with one argument only resolves an ID; included variables are a separate session store; read_json_file treats parsed null as fallback; unix_to_date_time ignores its accepted second argument; end() is not a game termination command.

The distinction between a diagnostic helper and a statement also matters: script_error logs; pca_assert reports and returns a Boolean; assert can fail a frame; the standalone breakpoint statement is the pause form.

Reference basis and version coverage

The language descriptions were checked against the installed 0.86.3.4 lexer, parser, validator, compiler and virtual machine, along with the script property catalog, shared action catalog/runner, script router and authored GUI runtime.

The compiler and runtime take precedence where a shorter help page is incomplete. Examples in this book use the implemented function spellings and options rather than assuming that a legacy native action name creates an equivalent PCA wrapper.

Coverage includes all 103 recognized builtin spellings, all 206 normal shared action wrappers, all 19 direct namespace kinds and their catalogued property paths, the shared property picker keys, the full direct record-method API and all 18 shared event hooks. Existing custom/raw dictionary fields can vary with your project and asset types.

For a later release, use the editor’s completion/signature hints and validation to check changes. A new catalog action can appear in the language without needing a new grammar rule. Editor-only operations still need their own editor workflow.

> No engine code or user project was changed to produce this guide. The screenshots show the native editor and its runtime; the PDF examples remain text you can select, search and copy.

Alphabetical function and event index

Click an entry to jump to its reference. The PDF’s outline/bookmarks provide another route through chapters, function groups and designer namespaces.

Entry	Kind	Entry	Kind
abs	Builtin	action	Builtin
activate_camera	Gameplay	ActivateCamera	Action ID
active_player	Builtin	add_asset_to_scene	Gameplay
add_character_to_scene	Gameplay	add_character_waypoint	Gameplay
add_dialogue_choice	Gameplay	add_dialogue_choice_node	Gameplay
add_dialogue_line	Gameplay	add_inventory	Gameplay
add_score	Gameplay	AddAssetToScene	Action ID
AddCharacterToScene	Action ID	AddCharacterWaypoint	Action ID
AddDialogueChoice	Action ID	AddDialogueChoiceNode	Action ID
AddDialogueLine	Action ID	AddInventory	Action ID
AddScore	Action ID	align_attach_points	Gameplay
AlignAttachPoints	Action ID	animation_ended	Builtin
assets	Namespace	attach_object	Gameplay
attach_particle_effect	Gameplay	attach_points	Gameplay
attach_points_touching	Gameplay	AttachObject	Action ID
AttachParticleEffect	Action ID	AttachPoints	Action ID
AttachPointsTouching	Action ID	basic_scripts	Namespace
bool	Builtin	boolean	Builtin
breakpoint	Builtin	call_script	Builtin
call_script_function	Gameplay	CallScriptFunction	Action ID
camera_orbit	Gameplay	camera_shake	Gameplay
CameraOrbit	Action ID	cameras	Namespace
CameraShake	Action ID	cancel_preposition	Gameplay
CancelPreposition	Action ID	carousel_add_image	Gameplay
carousel_clear_images	Gameplay	carousel_next	Gameplay
carousel_previous	Gameplay	carousel_remove_image	Gameplay
CarouselAddImage	Action ID	CarouselClearImages	Action ID
CarouselNext	Action ID	CarouselPrevious	Action ID
CarouselRemoveImage	Action ID	ceil	Builtin
center_gui_model	Gameplay	CenterGUIModel	Action ID
change_camera	Gameplay	change_character_description	Gameplay
change_character_property	Gameplay	change_game_cursor	Gameplay
change_game_setting	Gameplay	change_gui_element_property	Gameplay
change_gui_property	Gameplay	change_inventory_description	Gameplay

Entry	Kind	Entry	Kind
change_inventory_quantity	Gameplay	change_object_description	Gameplay
change_object_property	Gameplay	change_project_property	Gameplay
change_quest_objective_property	Gameplay	change_quest_property	Gameplay
change_quest_stage_property	Gameplay	change_region_description	Gameplay
change_scene	Gameplay	change_scene_description	Gameplay
change_subtitle_language	Gameplay	change_variable	Gameplay
change_verb	Gameplay	change_voice_language	Gameplay
ChangeCamera	Action ID	ChangeCharacterDescription	Action ID
ChangeCharacterProperty	Action ID	ChangeGameCursor	Action ID
ChangeGameSetting	Action ID	ChangeGUIElementProperty	Action ID
ChangeGUIProperty	Action ID	ChangeInventoryDescription	Action ID
ChangeInventoryQuantity	Action ID	ChangeObjectDescription	Action ID
ChangeObjectProperty	Action ID	ChangeProjectProperty	Action ID
ChangeQuestObjectiveProperty	Action ID	ChangeQuestProperty	Action ID
ChangeQuestStageProperty	Action ID	ChangeRegionDescription	Action ID
ChangeScene	Action ID	ChangeSceneDescription	Action ID
ChangeSubtitleLanguage	Action ID	ChangeVariable	Action ID
ChangeVerb	Action ID	ChangeVoiceLanguage	Action ID
character_teleport	Gameplay	character_walk_path	Gameplay
character_walk_to	Gameplay	character_walk_waypoints	Gameplay
characters	Namespace	characters.change_room	Method
characters.play_animation	Method	characters.say	Method
characters.walk_to	Method	CharacterTeleport	Action ID
CharacterWalkPath	Action ID	CharacterWalkTo	Action ID
CharacterWalkWaypoints	Action ID	choose	Builtin
clamp	Builtin	clear_character_waypoints	Gameplay
ClearCharacterWaypoints	Action ID	close_interaction_coin	Gameplay
CloseInteractionCoin	Action ID	color	Builtin
complete_objective	Gameplay	complete_quest	Gameplay
CompleteObjective	Action ID	CompleteQuest	Action ID
contains	Builtin	create_primitive	Gameplay
create_procedural	Gameplay	create_variable	Gameplay
CreatePrimitive	Action ID	CreateProcedural	Action ID
CreateVariable	Action ID	current_scene	Builtin
cursors	Namespace	cycle_interaction_verb	Gameplay
CycleInteractionVerb	Action ID	date_time_now	Builtin
date_time_string	Builtin	despawn_object	Gameplay
DespawnObject	Action ID	detach_object	Gameplay
detach_particle_effect	Gameplay	DetachObject	Action ID

Entry	Kind	Entry	Kind
DetachParticleEffect	Action ID	dialogues	Namespace
directory_exists	Builtin	disable_gui	Gameplay
disable_gui_element	Gameplay	DisableGUI	Action ID
DisableGUIElement	Action ID	distance	Builtin
emit_event	Builtin	enable_gui	Gameplay
enable_gui_element	Gameplay	EnableGUI	Action ID
EnableGUIElement	Action ID	end	Builtin
end_cutscene	Gameplay	end_dialogue	Gameplay
end_game	Builtin	end_visual_script	Builtin
EndCutscene	Action ID	EndDialogue	Action ID
ends_with	Builtin	execute_preposition	Gameplay
ExecutePreposition	Action ID	exists	Builtin
face_character	Gameplay	face_coordinates	Gameplay
face_object	Gameplay	FaceCharacter	Action ID
FaceCoordinates	Action ID	FaceObject	Action ID
fade_music	Gameplay	fade_screen	Gameplay
fade_sound	Gameplay	FadeMusic	Action ID
FadeScreen	Action ID	FadeSound	Action ID
fail_objective	Gameplay	fail_quest	Gameplay
FailObjective	Action ID	FailQuest	Action ID
file_exists	Builtin	first_visit	Builtin
float	Builtin	floor	Builtin
follow_character	Gameplay	FollowCharacter	Action ID
fonts	Namespace	format	Builtin
from_json	Builtin	get_attach_point	Gameplay
get_attach_point_contacts	Gameplay	get_control_state	Gameplay
get_gui_element_property	Builtin	get_gui_property	Builtin
get_included_variable	Builtin	get_project_property	Builtin
get_setting	Builtin	GetAttachPoint	Action ID
GetAttachPointContacts	Action ID	GetControlState	Action ID
gui_element_property	Builtin	gui_property	Builtin
guis	Namespace	has_inventory	Builtin
has_visited	Builtin	hide_gui	Gameplay
hide_gui_element	Gameplay	hide_player_character	Gameplay
HideGUI	Action ID	HideGUIElement	Action ID
HidePlayerCharacter	Action ID	if_character_moving	Builtin
if_inventory_item_was_used	Builtin	if_player_was_here	Builtin
if_player_was_never_here	Builtin	if_variable	Builtin
include_script	Builtin	int	Builtin

Entry	Kind	Entry	Kind
integer	Builtin	interact_nearest	Gameplay
InteractNearest	Action ID	inventory	Namespace
inventory_count	Builtin	is_current_room	Gameplay
is_valid	Builtin	IsCurrentRoom	Action ID
join	Builtin	jump	Gameplay
Jump	Action ID	len	Builtin
length	Builtin	lerp	Builtin
list_box_add_item	Gameplay	list_box_clear_items	Gameplay
list_box_delete_item	Gameplay	list_box_modify_item	Gameplay
ListBoxAddItem	Action ID	ListBoxClearItems	Action ID
ListBoxDeleteItem	Action ID	ListBoxModifyItem	Action ID
load_game	Gameplay	load_game_slot	Gameplay
LoadGame	Action ID	LoadGameSlot	Action ID
log	Builtin	lower	Builtin
lua_scripts	Namespace	max	Builtin
melt_snow	Gameplay	MeltSnow	Action ID
min	Builtin	move_npc_to_room	Gameplay
MoveNPCToRoom	Action ID	now_unix	Builtin
number	Builtin	objective_state	Builtin
objectives	Namespace	objects	Namespace
objects.play_animation	Method	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	open_interaction_coin	Gameplay
OpenInteractionCoin	Action ID	pause_animation	Gameplay
pause_fmv	Gameplay	pause_music	Gameplay
pause_sound	Gameplay	pause_voice	Gameplay
PauseAnimation	Action ID	PauseFMV	Action ID
PauseMusic	Action ID	PauseSound	Action ID
PauseVoice	Action ID	pca_assert	Builtin
play_animation	Gameplay	play_gameplay_recording	Gameplay
play_gui_model_animation	Gameplay	play_music	Gameplay
play_sound	Gameplay	play_video	Gameplay

Entry	Kind	Entry	Kind
play_voice	Gameplay	PlayAnimation	Action ID
player_has	Builtin	PlayGameplayRecording	Action ID
PlayGUIModelAnimation	Action ID	PlayMusic	Action ID
PlaySound	Action ID	PlayVideo	Action ID
PlayVoice	Action ID	project_property	Builtin
quest_state	Builtin	quests	Namespace
quit_to_menu	Builtin	random_float	Builtin
random_int	Builtin	random_integer	Builtin
random_number	Builtin	randomize	Builtin
range	Builtin	range_inclusive	Builtin
read_json_file	Builtin	read_text_file	Builtin
regex_search	Builtin	remove_attach_point	Gameplay
remove_dialogue_choice	Gameplay	remove_dialogue_node	Gameplay
remove_inventory	Gameplay	RemoveAttachPoint	Action ID
RemoveDialogueChoice	Action ID	RemoveDialogueNode	Action ID
RemoveInventory	Action ID	replace	Builtin
reset_dialogue_history	Gameplay	reset_objective	Gameplay
reset_quest	Gameplay	ResetDialogueHistory	Action ID
ResetObjective	Action ID	ResetQuest	Action ID
resume_animation	Gameplay	resume_fmv	Gameplay
resume_music	Gameplay	resume_sound	Gameplay
resume_voice	Gameplay	ResumeAnimation	Action ID
ResumeFMV	Action ID	ResumeMusic	Action ID
ResumeSound	Action ID	ResumeVoice	Action ID
rooms	Namespace	round	Builtin
run_visual_graph	Builtin	run_visual_script	Gameplay
RunVisualScript	Action ID	save_game	Gameplay
save_game_slot	Gameplay	save_game_widget_command	Gameplay
SaveGame	Action ID	SaveGameSlot	Action ID
SaveGameWidgetCommand	Action ID	say	Gameplay
Say	Action ID	screen_transition	Gameplay
ScreenTransition	Action ID	script_error	Builtin
scripts	Namespace	seek_fmv	Gameplay
seek_music	Gameplay	seek_video	Gameplay
seek_voice	Gameplay	SeekFMV	Action ID
SeekMusic	Action ID	SeekVideo	Action ID
SeekVoice	Action ID	set2d_camera	Gameplay
set2d_entity_depth	Gameplay	set2d_layer_visible	Gameplay
set2d_parallax	Gameplay	Set2DCamera	Action ID

Entry	Kind	Entry	Kind
Set2DEntityDepth	Action ID	Set2DLayerVisible	Action ID
Set2DParallax	Action ID	set_attach_point	Gameplay
set_auto_input_switching	Gameplay	set_camera_behavior	Gameplay
set_camera_priority	Gameplay	set_camera_targets	Gameplay
set_character_action_enabled	Gameplay	set_control_setting	Gameplay
set_dialogue_choice_enabled	Gameplay	set_flashlight_mode	Gameplay
set_game_language	Gameplay	set_glow	Gameplay
set_gui_element_property	Builtin	set_gui_property	Builtin
set_included_variable	Builtin	set_input_device	Gameplay
set_interaction_approach_enabled	Gameplay	set_interaction_mode	Gameplay
set_lip_sync_enabled	Gameplay	set_motion	Gameplay
set_mouse_clicks_enabled	Gameplay	set_mouse_pointer_visible	Gameplay
set_mouse_pointing_enabled	Gameplay	set_movement_profile	Gameplay
set_object_action_enabled	Gameplay	set_object_enabled	Gameplay
set_object_interaction_approach	Gameplay	set_object_visible	Gameplay
set_player_input_enabled	Gameplay	set_project_property	Builtin
set_proximity_interaction_enabled	Gameplay	set_quest_hidden	Gameplay
set_quest_stage	Gameplay	set_quest_tracked	Gameplay
set_retro_style	Gameplay	set_room_camera_bounds	Gameplay
set_room_dimension	Gameplay	set_runtime_controls_enabled	Gameplay
set_save_slot_image	Gameplay	set_scene_brightness	Gameplay
set_scene_tint	Gameplay	set_setting	Builtin
set_snow_accumulation	Gameplay	set_trigger_enabled	Gameplay
set_variable	Gameplay	set_verb_usage_enabled	Gameplay
set_voice_speaker	Gameplay	SetAttachPoint	Action ID
SetAutoInputSwitching	Action ID	SetCameraBehavior	Action ID
SetCameraPriority	Action ID	SetCameraTargets	Action ID
SetCharacterActionEnabled	Action ID	SetControlSetting	Action ID
SetDialogueChoiceEnabled	Action ID	SetFlashlightMode	Action ID
SetGameLanguage	Action ID	SetGlow	Action ID
SetInputDevice	Action ID	SetInteractionApproachEnabled	Action ID
SetInteractionMode	Action ID	SetLipSyncEnabled	Action ID
SetMotion	Action ID	SetMouseClicksEnabled	Action ID
SetMousePointerVisible	Action ID	SetMousePointingEnabled	Action ID
SetMovementProfile	Action ID	SetObjectActionEnabled	Action ID
SetObjectEnabled	Action ID	SetObjectInteractionApproach	Action ID
SetObjectVisible	Action ID	SetPlayerInputEnabled	Action ID
SetProximityInteractionEnabled	Action ID	SetQuestHidden	Action ID
SetQuestStage	Action ID	SetQuestTracked	Action ID

Entry	Kind	Entry	Kind
SetRetroStyle	Action ID	SetRoomCameraBounds	Action ID
SetRoomDimension	Action ID	SetRuntimeControlsEnabled	Action ID
SetSaveSlotImage	Action ID	SetSceneBrightness	Action ID
SetSceneTint	Action ID	SetSnowAccumulation	Action ID
setting	Builtin	SetTriggerEnabled	Action ID
SetVariable	Action ID	SetVerbUsageEnabled	Action ID
SetVoiceSpeaker	Action ID	shake_screen	Gameplay
ShakeScreen	Action ID	show_gui	Gameplay
show_gui_element	Gameplay	show_message	Gameplay
ShowGUI	Action ID	ShowGUIElement	Action ID
ShowMessage	Action ID	shuffle	Builtin
spawn_object	Gameplay	SpawnObject	Action ID
spawns	Namespace	split	Builtin
start_cutscene	Gameplay	start_dialogue	Gameplay
start_fmv	Gameplay	start_gameplay_recording	Gameplay
start_quest	Gameplay	StartCutscene	Action ID
StartDialogue	Action ID	StartFMV	Action ID
StartGameplayRecording	Action ID	StartQuest	Action ID
starts_with	Builtin	stop_animation	Gameplay
stop_attach_point_watch	Gameplay	stop_camera_orbit	Gameplay
stop_fmv	Gameplay	stop_follow_character	Gameplay
stop_gameplay_playback	Gameplay	stop_gameplay_recording	Gameplay
stop_music	Gameplay	stop_screen_shake	Gameplay
stop_sound	Gameplay	stop_startup_splash_audio	Gameplay
stop_video	Gameplay	stop_voice	Gameplay
stop_zoom_blur	Gameplay	StopAnimation	Action ID
StopAttachPointWatch	Action ID	StopCameraOrbit	Action ID
StopFMV	Action ID	StopFollowCharacter	Action ID
StopGameplayPlayback	Action ID	StopGameplayRecording	Action ID
StopMusic	Action ID	StopScreenShake	Action ID
StopSound	Action ID	StopStartupSplashAudio	Action ID
StopVideo	Action ID	StopVoice	Action ID
StopZoomBlur	Action ID	str	Builtin
string	Builtin	string_length	Builtin
string_split	Builtin	switch_character	Gameplay
SwitchCharacter	Action ID	text	Builtin
to_json	Builtin	toggle_glow	Gameplay
toggle_motion	Gameplay	toggle_rgb_string	Gameplay
ToggleGlow	Action ID	ToggleMotion	Action ID

Entry	Kind	Entry	Kind
ToggleRGBString	Action ID	transfer_inventory	Gameplay
TransferInventory	Action ID	triggers	Namespace
triggers.play_animation	Method	trim	Builtin
type_of	Builtin	unix_to_date_time	Builtin
unlock_hint	Gameplay	UnlockHint	Action ID
upper	Builtin	vector2	Builtin
vector3	Builtin	verbs	Namespace
visual_scripts	Namespace	warn	Builtin
watch_attach_point_contacts	Gameplay	WatchAttachPointContacts	Action ID
write_json_file	Builtin	write_text_file	Builtin
zoom_blur	Gameplay	ZoomBlur	Action ID