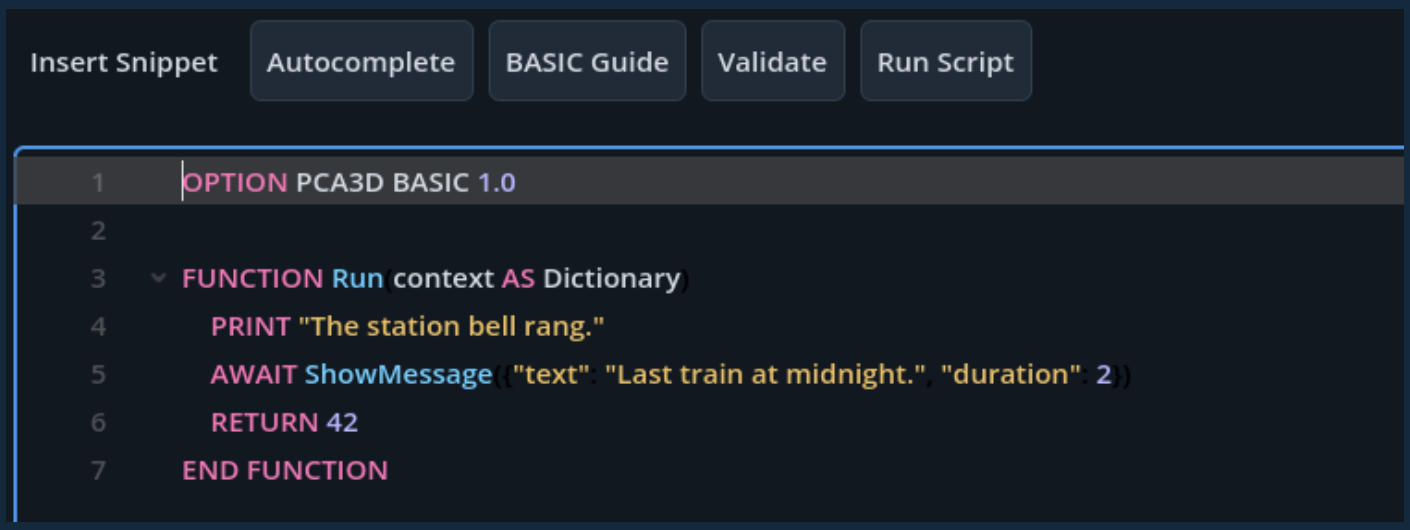
PCA 3D ADVENTURE STUDIO

BASIC scripting

A programming guide for adventure authors



What happens when a player clicks?

Follow the answer from a button, through a BASIC function, and back to the game. Read each example right on the page.

CONTENTS

Find a topic

New to BASIC? Read pages 3–5 first. After that, go straight to the question that brought you here. Chapter titles and PDF bookmarks jump to the same places.

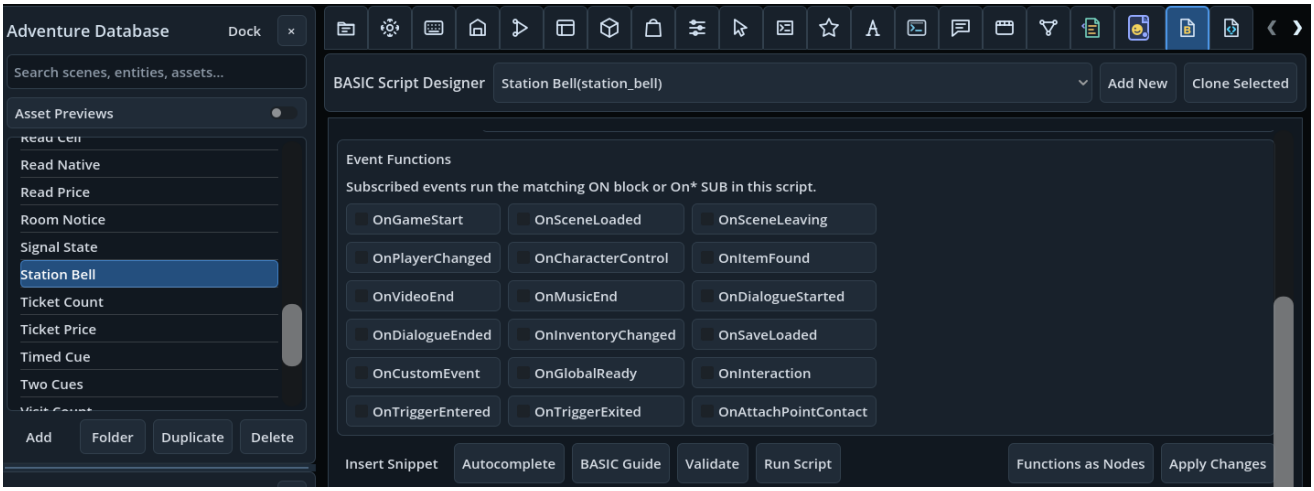
Find the editor and run your first script	3
Variables, types, operators and decisions	7
Loops, lists and dictionaries	12
Text, math and functions	15
Connect a GUI and read player input	18
Waits and parallel actions	20
Text files and JSON settings	22
Game events and custom messages	24
Functions as Nodes and native Godot calls	26
Inventory, story actions and save slots	28
Debugging and platform differences	31
Event names and common helpers	33
Look up all 206 gameplay actions	35

> Most dark boxes show a whole script record. When the text calls a line a fragment, place it inside an existing function. Read the expected result below the listing to see what it does.

GETTING ORIENTED

Open the BASIC Script Designer

A script record gives a piece of code a name in your game. A button or a room event can use that name to start it.



The Adventure Database is on the left. The BASIC Script Designer has its own record picker and Add New button.

01 Open your game in PCA3D. Select a record under **BASIC Scripts** in the Adventure Database, or switch to the **BASIC Script** workspace. On a narrow screen, scroll the workspace icons.

02 Choose **Add New** to create a record. For the first listing, use **station_bell** for Script ID and **Station Bell** for Name. Use a unique ID across all script languages. The display name can contain spaces.



The ID is what other records call. The Name is the label you see in the editor.

Leave the event boxes clear for a script you plan to run by hand. Keep **enabled** checked in the Inspector. Enter BASIC in the large code box below the event controls.

FIRST SCRIPT

Write and run your first function

What should happen when someone examines a station bell? Let the author see a log entry and let the player see a line of text.

```
OPTION PCA3D BASIC 1.0

FUNCTION Run(context AS Dictionary)
    PRINT "The station bell rang."
    AWAIT ShowMessage({"text": "Last train at midnight.", "duration": 2})
    RETURN 42
END FUNCTION
```

Run is the entry point. PCA3D starts there when you run the record. **context** holds information from the caller. The **OPTION** line identifies the dialect. You can leave it out in the shorter examples.

PRINT	Writes “The station bell rang.” to the output. Use it to check what the code is doing.
ShowMessage	Displays the quoted line to the player. The dictionary in braces supplies the text and its duration in seconds.
RETURN 42	Sends 42 to the caller and leaves the function. A manual run does not automatically display the returned number.

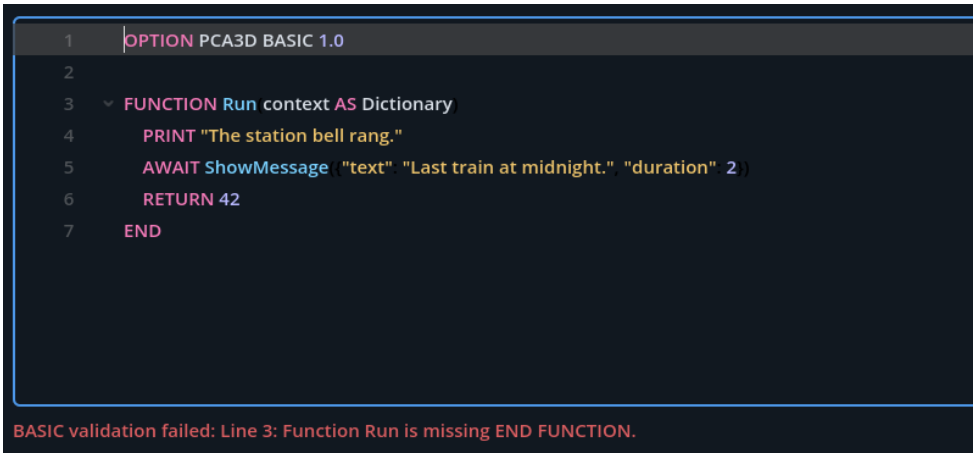
01 Choose **Validate** on the BASIC toolbar. When the status below the code says **BASIC validation passed.**, choose **Apply Changes** and then **Save Project**.

02 Choose **Run Script**. Watch the game preview for “Last train at midnight.” Change just those words, apply, and run again to check your edit.

EDITOR TOOLS

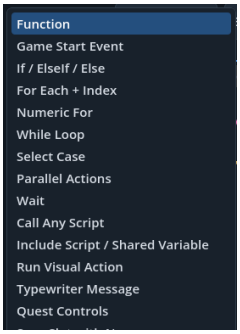
Read a validation message

When a complete-looking script refuses to run, the status below the code is a better starting point than changing several lines at once.



Here the function starts correctly, but its last line says END. BASIC reports that Run is missing END FUNCTION.

Read the name in the message. Find the matching opening line and follow the block down to its closing line. In this case, replace **END** with **END FUNCTION** and choose Validate again.



Insert Snippet includes templates for functions, loops, events, parallel actions and cross-script calls.

Insert Snippet inserts text at the caret. Place a new function outside an existing function. Place the body of a loop inside a function. **Autocomplete** can help with command names and record references. **BASIC Guide** opens the documentation supplied with that build.

> Validate checks the source. Apply Changes updates the record. Save Project writes your game to disk. The main toolbar also has a project-wide Validate button.

THE BASIC LANGUAGE

Write text, quotes and comments

A line of story text often contains punctuation BASIC also uses. A sign that reads “Closed” is a good example.

```
FUNCTION Run()
    REM A notice painted on the wall.
    DIM notice = "The sign reads ""Closed""."
    ' Keep the punctuation when adding the second line.
    RETURN notice & "\n" & "Come back at dawn."
END FUNCTION
```

The outer double quotes mark the start and end of a string. Two double quotes inside that string produce one visible quotation mark. An apostrophe starts a comment outside quoted text. **REM** at the start of a line also marks an author’s note.

Read the result

```
The sign reads "Closed".
Come back at dawn.
```

The backslash and n inside the string produce a newline. The ampersand joins the pieces. Keep straight double quotes in code, even if your word processor prefers curly ones.

> Keywords and BASIC identifiers are case-insensitive. TRUE and true mean the same thing. Text inside quotes keeps its spelling and capitalization. Copy project IDs and named action fields exactly as shown by the editor.

THE BASIC LANGUAGE

Name a value and give it a type

A ticket counter needs a starting quantity, a sale, and a number of seats left. DIM gives a changing value a name. CONST holds the capacity steady.

```
FUNCTION Run()
    CONST capacity AS Integer = 8
    DIM sold AS Integer = 3
    sold += 2
    DIM spare = capacity - sold
    RETURN spare
END FUNCTION
```

> **What you should see** The function returns **3**. Five tickets were sold, leaving three seats. The += on the fourth line adds two to the existing value.

Integer / Number	Use Integer to count items and Number for fractional values such as seconds. Long is an Integer alias. Single, Double and Decimal map to Number.
String / Boolean	String or Text holds words, including an empty string. Boolean holds TRUE or FALSE.
Array / Dictionary	An Array holds entries in order. A Dictionary stores values under keys such as "owner".
Variant / NOTHING	A value can be left untyped. NOTHING represents an absent value. An empty string is a piece of text with zero characters.

The AS annotation is optional. Use it when the expected kind of value helps someone understand the function or gives the validator a useful check.

THE BASIC LANGUAGE

Decide how long data should live

A counter used during one calculation, a counter private to a record, and a flag saved with the game each need a different home.

```
DIM visits AS Integer = 0

FUNCTION Run()
    visits += 1
    RETURN visits
END FUNCTION
```

What you should see Two calls to the same unchanged record returned **1** and **2**. The DIM sits above FUNCTION, so both calls see the same private counter.

```
GLOBAL signal_seen AS Boolean = FALSE

FUNCTION Run()
    signal_seen = TRUE
    RETURN signal_seen
END FUNCTION
```

GLOBAL uses the game’s shared, saveable variables. In the editor runtime, saving with signal_seen true, clearing it, and loading the slot restored true. Use a distinct name for every flag that means something different.

Inside a function	DIM belongs to that call. It starts again next time.
At the top of the record	DIM or top-level STATIC retains private state until the module/cache is reset. Editing or reloading can reset it.
Across records and saves	GLOBAL or SHARED gives other records and the save system access. Local STATIC inside a function is unsupported.

THE BASIC LANGUAGE

Read operators in a line of BASIC

The same equals sign can assign or compare. Read the whole line. In coins = 3 it sets a value. In IF coins = 3 THEN it asks a question.

Arithmetic	+ adds, − subtracts (type the plain minus key), * multiplies, / divides, and MOD gives the remainder. For 17 MOD 6 the answer is 5.
Assignment	Use =, +=, -=, *= and /= to update a variable. Use CONST if later reassignment should produce a validation error.
Comparison	Use = and <> for equal and unequal. Use <, <=, > and >= for ordered comparisons.
Logic	AND needs both tests to succeed. OR needs either one. NOT reverses a test. ANDALSO and ORELSE are short-circuit aliases.
Join text	Use & between pieces of text. Parentheses make the order clear when you mix arithmetic with concatenation.

Predict the result

```
DIM coins = 3
DIM has_pass = FALSE
DIM admitted = coins >= 2 OR has_pass
PRINT "Admitted: " & admitted
```

This is a fragment for Run. The left half of OR is true because three is at least two. Admitted is therefore true. Set coins to 1 and has_pass to FALSE to reach the other result.

THE BASIC LANGUAGE

Choose a response with IF

Evaluate the most specific condition first. Once a condition succeeds, BASIC runs that branch and skips the rest.

```
FUNCTION Run()
    DIM coins = 3
    IF coins < 2 THEN
        RETURN "Too few coins."
    ELSEIF coins = 2 THEN
        RETURN "Exact fare."
    ELSE
        RETURN "Change due."
    END IF
END FUNCTION
```

coins = 1	The first comparison succeeds. The answer is “Too few coins.”
coins = 2	The first comparison fails. ELSEIF catches the exact amount.
coins = 3	Both comparisons fail. ELSE handles the remaining case and returns “Change due.”

Close a multiline IF with **END IF**. ELSEIF is one word. An ELSE is optional when the script has nothing to do for the remaining case.

THE BASIC LANGUAGE

Pick from several named cases

SELECT CASE is handy when one value determines which of several lines to use. Here, both local platforms share one reply.

```
FUNCTION Run()
    DIM track = 2
    SELECT CASE track
    CASE 1, 2
        RETURN "Local train"
    CASE 3
        RETURN "Express"
    CASE ELSE
        RETURN "No departure"
    END SELECT
END FUNCTION
```

BASIC evaluates track once. A comma in **CASE 1, 2** means either value matches. CASE ELSE catches any value the earlier choices missed.

> **What you should see** With track set to 2, the result is **Local train**. A 3 returns Express. A 9 returns No departure.

> Use IF and ELSEIF for ranges such as “fewer than five coins.” The PCA3D dialect accepts equality choices in CASE. A classic BASIC CASE range or CASE IS expression needs to be rewritten here.

THE BASIC LANGUAGE

Count with FOR and FOR EACH

Use a numeric FOR when you know the first and last numbers. Use FOR EACH when the job is to visit the entries in a collection.

```
FUNCTION Run()
    DIM total = 0
    FOR n = 5 TO 1 STEP -2
        total += n
    NEXT
    RETURN total
END FUNCTION
```

The loop visits 5, 3 and 1. Both endpoints are included when the step lands on them. total ends at **9**. STEP defaults to 1 and must be nonzero.

```
FUNCTION Run()
    DIM names = ["Ada", "Ben", "Cy"]
    DIM chosen = ""
    FOR EACH person, i IN names
        IF i = 1 THEN
            CONTINUE
        END IF
        chosen = chosen & person
    NEXT
    RETURN chosen
END FUNCTION
```

The first item has index 0. On index 1, CONTINUE skips Ben and advances to the next item. The result is **AdaCy**. For a dictionary, FOR EACH visits keys. Its optional index counts visits.

THE BASIC LANGUAGE

Repeat until something changes

A WHILE loop checks before it starts. A DO loop with a test at the bottom always gets one turn. Choose the form that matches the rule you mean.

```
FUNCTION Run()
    DIM turns = 0
    WHILE turns < 2
        turns += 1
    WEND
    DO
        turns += 1
    LOOP UNTIL turns >= 3
    RETURN turns
END FUNCTION
```

The WHILE runs at 0 and 1, stopping at 2. The DO body then adds one, and LOOP UNTIL tests the new value. The result is **3**.

WHILE … WEND	Use it when the body might need zero turns.
DO WHILE / DO UNTIL	Put the test at the top for another pre-tested form. Close with LOOP.
REPEAT … END REPEAT	Give a fixed number of repetitions. BREAK exits a loop. CONTINUE skips ahead to its next turn.

> Change the value your loop tests. A tight loop waiting for a person to click cannot process the click. Use a GUI event or WAIT UNTIL for something that can change on a later frame.

THE BASIC LANGUAGE

Keep lists and named fields together

A list answers “which item is second?” A dictionary answers “who owns the list?” BASIC can store one inside the other.

```
FUNCTION Run()
    DIM items = ["lamp", "key", "map"]
    DIM card = {"owner": "Ada", "items": items}
    RETURN card.owner & " carries the " & card.items[1]
END FUNCTION
```

> **What you should see** The function returns **Ada carries the key**. The sequence is lamp at 0, key at 1 and map at 2.

card.owner reads a named field. **card.items[1]** reads the items field and then selects its second entry. In a list of length 3, the valid indexes are 0 through 2. Use Len(items) when the list length depends on the game.

When players add their own data

Use stable field names like owner and items. Keep a displayed sentence, an internal record ID and a numeric index distinct when you pass them to another function. For a potentially empty list, check its length before reading entry 0.

THE BASIC LANGUAGE

Clean and combine player text

A player who types a direction with spaces or mixed capitalization probably means the same thing. Normalize the input before comparing it.

```
FUNCTION Run()
    DIM raw = "  WeSt  "
    DIM word = Lower(Trim(raw))
    DIM report = Replace("Dial: ?", "?", word)
    RETURN report
END FUNCTION
```

> **What you should see** The result is **Dial: west**. Trim removes whitespace at the ends, Lower changes WeSt to west, and Replace fills the question mark.

Upper / Lower	Convert the letter case for display or comparison.
Contains / StartsWith / EndsWith	Test a piece of text. An optional third Boolean chooses case sensitivity.
Split / Join	Split("red,blue", ",") produces two entries. Join(["red","blue"], " / ") produces the text red / blue.
StringLength	Count the length of a string. A zero-length answer can be given its own feedback before any other comparisons.

THE BASIC LANGUAGE

Calculate distances and pick a hint

Math helpers handle small but common gameplay jobs. A gauge can stop at its maximum, while a clue can be chosen from several lines.

```
FUNCTION Run()
    DIM charge = Clamp(89 + 25, 0, 100)
    DIM ticks = 17 MOD 6
    DIM distance = Distance(Vector3(0,0,0), Vector3(3,0,4))
    RETURN [charge, ticks, distance]
END FUNCTION
```

The results are **100, 5 and 5**. Clamp keeps charge in bounds, MOD gives the remainder, and Distance measures a 3–4–5 displacement between two 3D positions.

```
FUNCTION Run()
    DIM clues = ["The ink is fresh.", "The latch is warm."]
    DIM pick = RandomInt(0, 1)
    RETURN clues[pick]
END FUNCTION
```

RandomInt includes both ends. There are two possible indexes here, 0 and 1. For a list that changes size, use **RandomInt(0, Len(clues) - 1)** after checking the list contains at least one entry.

> Use Floor, Ceil and Round when converting a fraction to a whole step. Store a randomly chosen puzzle answer in game state if players should still get the same answer after loading.

THE BASIC LANGUAGE

Give a function a useful name

A FUNCTION calculates an answer. A SUB is a named job you can call for its effect, such as writing a line to the output.

```
FUNCTION Fare(count AS Integer, rate AS Integer = 2)
    RETURN count * rate
END FUNCTION

SUB Announce(total AS Integer)
    PRINT "Fare: " & total
END SUB

FUNCTION Run()
    DIM total = Fare(3)
    Announce(total)
    RETURN total
END FUNCTION
```

Fare receives a count and a rate. The default rate is 2. Fare(3) returns **6**, while Fare(3, 4) returns **12**. Announce prints the value it receives. Run returns the number to its own caller.

> Use RETURN value for a function result. EXIT SUB and EXIT FUNCTION leave without a result. PCA3D BASIC supports optional default parameters. Pass explicit values or return updated data when adapting a classic routine that uses BYREF.

BASIC IN YOUR GAME

Connect a button to BASIC

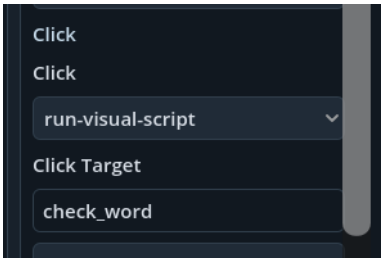
A text box holds the player’s words. A button decides when to read them. A label gives an answer the player can see.



The GUI Designer shows the three elements. The button is selected in GUI Elements.

01 In GUI Designer, select the GUI that contains your controls. The example uses GUI ID **station**, text box **entry**, button **submit** and label **reply**. The button’s visible caption is “Check direction.”

02 Select the button. In **Element Properties → Events**, find **Click**. Set its behavior to **run-visual-script** and its target to the enabled BASIC record **check_word**. That behavior accepts a BASIC script ID.



Click and Click Target in the button Inspector. The target names a script record.

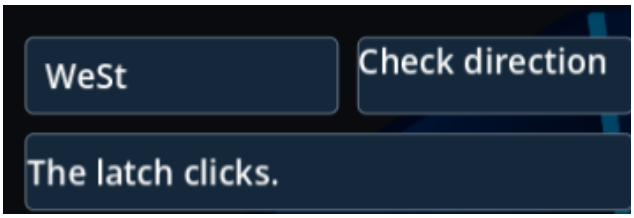
Apply the changes and save the project. Use the next page for the code that goes inside check_word. In a game preview, make sure the GUI is open or set to be visible at the start.

BASIC IN YOUR GAME

Read exactly what the player typed

Three answers are enough to tell whether the whole connection works. Try an empty box, a wrong word, and a correct word typed in mixed case.

```
FUNCTION Run()
    DIM raw = GetGUIElementProperty("station", "entry", "text", "")
    DIM answer = Lower(Trim(raw))
    DIM reply = "Read the sign again."
    IF answer = "" THEN
        reply = "Type a direction."
    ELSEIF answer = "west" THEN
        reply = "The latch clicks."
    END IF
    SetGUIElementProperty("station", "reply", "text", reply)
    RETURN reply
END FUNCTION
```



The running preview after the player entered WeSt and selected Check direction.

Empty box or spaces	Trim leaves an empty string. The label reads “Type a direction.”
east	The starting reply stays in place, so the label reads “Read the sign again.”
WeSt	Lower turns it into west. The label reads “The latch clicks.”

The last empty string in GetGUIElementProperty is the fallback. If every entry acts blank, check the GUI ID, element ID and property name. RETURN also gives the calling script the answer.

BASIC IN YOUR GAME

Pause a script without freezing the room

WAIT takes seconds. A tenth of a second is 0.1. While this script waits, the game can keep drawing the room and listening for input.

```
FUNCTION Run()
    PRINT "First chime"
    WAIT 0.1
    PRINT "Second chime"
    RETURN "done"
END FUNCTION
```

The output shows First chime, then Second chime. The script returns **done** after the second line. Use a longer wait when the pause should be noticeable to the player.

```
FUNCTION Run()
    DIM ready = TRUE
    WAIT UNTIL ready
    RETURN "Ready"
END FUNCTION
```

WAIT UNTIL asks a yes-or-no question. Here ready is already true, so the result is **Ready** immediately. In a game, use a shared flag that a button or another event can change. A local flag set to FALSE and never changed would leave the script waiting.

A deadline the game can remember

```
signal_deadline = NowUnix() + 30
remaining = Max(0, signal_deadline - NowUnix())
```

These are two separate fragments. Declare signal_deadline as a GLOBAL Integer and use DIM remaining for the local calculation. Use the first line when the clock starts. Use the second whenever you display the seconds left. NowUnix reads the device clock, so time away from the game counts too.

BASIC IN YOUR GAME

Let two cues overlap

A bell and a whistle can start on different beats while the same script waits for both to finish.

```
FUNCTION Run()
    PARALLEL
        WAIT 0.05
        PRINT "Bell"
    BRANCH
        WAIT 0.1
        PRINT "Whistle"
    END PARALLEL
    RETURN "Both finished"
END FUNCTION
```

The first branch waits 0.05 seconds, then prints Bell. The second starts alongside it, waits 0.1 seconds, then prints Whistle. After both finish, the function returns **Both finished**.

AWAIT	Marks a gameplay call that waits for completion. BASIC also waits for gameplay calls automatically. Check action options such as wait or waitForContinue when choosing when that action finishes.
BRANCH	Keeps the statements in one branch in sequence. Without BRANCH, each statement directly inside PARALLEL starts independently.
Shared data	Each branch has its own local values. An assignment there does not update the parent’s local variable. Avoid having two branches change the same global counter at once.

BASIC IN YOUR GAME

Read and write a text file

Suppose you want a journal to keep every entry. Writing a file replaces its contents, so read the old text first and add the new line to it.

```
FUNCTION Run()
    DIM path = "user://station/journal.txt"
    DIM previous = ReadTextFile(path, "")
    DIM saved = WriteTextFile(path, previous & "Bell heard.\n")
    IF NOT saved THEN
        RETURN "Write failed"
    END IF
    RETURN ReadTextFile(path, "")
END FUNCTION
```

On the first run, ReadTextFile uses the empty fallback because there is no journal yet. WriteTextFile creates the folder if needed and writes one line. Run it again and the returned text contains two lines.

user://	The game’s writable data location. PCA3D resolves it for the device. A Windows drive letter or a Linux home-directory name would tie the code to one machine.
Write failed	WriteTextFile returns a Boolean. Check it before telling the player that something was saved.
A single save slot?	This path is shared by every slot using it. Put story progress in saveable game state, or deliberately give each slot its own file path.

BASIC IN YOUR GAME

Keep settings in a JSON file

JSON stores named values in a text file. It works well for a preference like “show hints” because the name and value stay together.

```
FUNCTION Run()
    DIM path = "user://station/options.json"
    DIM prefs = ReadJSONFile(path, {"hints": TRUE})
    prefs.hints = FALSE
    IF NOT WriteJSONFile(path, prefs) THEN
        RETURN "Write failed"
    END IF
    DIM loaded = ReadJSONFile(path, {})
    RETURN loaded.hints
END FUNCTION
```

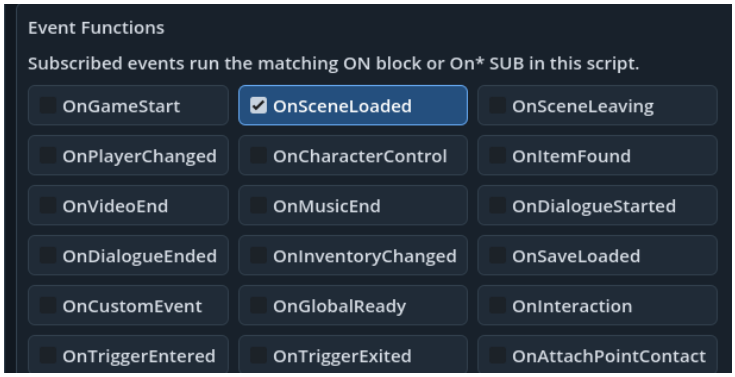
For a missing or unreadable setting, the starting dictionary supplies hints = TRUE. The script changes hints to FALSE and writes the dictionary. It then reads it back. The returned **FALSE** is the value of hints. The text **Write failed** is the separate failure result.

> A successful JSON parse can still produce the wrong kind of data. This listing expects a dictionary. If another tool or an older edition of your game can write the same file, check the structure and supply defaults for missing fields before using them.

BASIC IN YOUR GAME

Let the engine call your script

An event is the game telling you something happened. A room-arrival handler can read which room just loaded without checking the room every frame.



OnSceneLoaded is checked for the Room Notice record in the BASIC Designer.

```
SUB OnSceneLoaded(context AS Dictionary)
    PRINT "Arrived in " & context.sceneId
END SUB
```

01 Open the BASIC record, check **OnSceneLoaded** under Event Functions, enter the matching SUB, then apply and save. The editor’s upper record fields identify this handler as room_notice.

02 Start or enter a room. In the recorded preview, the room’s ID was **lab** and the output read **Arrived in lab**. The player-facing room name can be completely different.

context is a dictionary of details for this event. A scene event can supply sceneId. Other events supply different details. To inspect what actually arrived, put **PRINT context** inside the SUB and trigger the event in the game.

BASIC IN YOUR GAME

Send a message to another script

One part of a game can announce that the bell rang. A different record can decide whether it cares about that bell.

The listening record

```
ON OnCustomEvent THEN
    IF event.topic = "station_bell" THEN
        PRINT "Heard " & event.strikes & " strikes"
    END IF
END ON
```

Check **OnCustomEvent** for bell_listener and apply. In an ON block, the incoming dictionary is called **event**. The topic check lets this listener ignore other messages on the same event.

The sending record

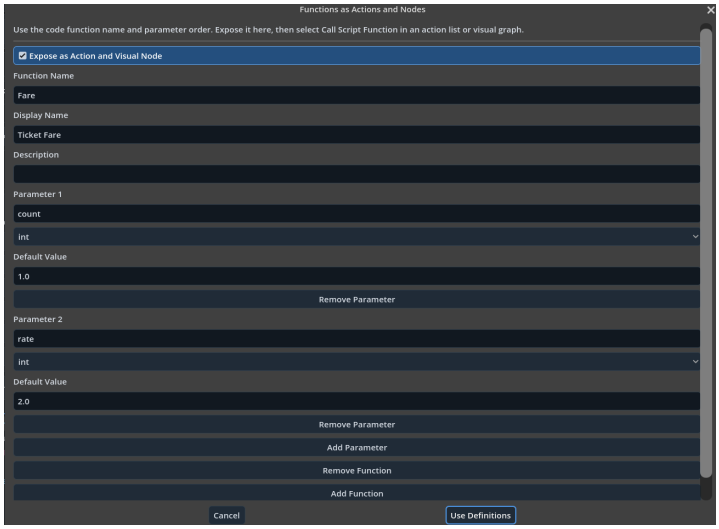
```
FUNCTION Run()
    EMIT EVENT "OnCustomEvent" WITH {"topic": "station_bell", "strikes": 2}
    RETURN TRUE
END FUNCTION
```

> **What you should see** Running bell_sender dispatched the message. The listener printed **Heard 2 strikes**. Both records use the same topic text and the same field name, strikes.

BASIC IN YOUR GAME

Use a BASIC function as a visual node

A designer can use a fare calculation in an action list without retyping its formula. The BASIC record owns the calculation. The visual action supplies the two numbers.



Functions as Nodes in the BASIC Designer. Fare is the callable name. Ticket Fare is its display label.

01 Open **ticket_price** from page 17, then **Functions as Nodes**. Enable **Expose as Action and Visual Node**. Set Function Name to **Fare** and Display Name to **Ticket Fare**.

02 Define two integer parameters in function order, **count** and **rate**. The pictured defaults are 1 and 2. Select **Use Definitions** to accept the form, then apply and save the script record.

03 Add **Call Script Function** to an action list or visual graph. Choose Ticket Fare. Supply count 3 and rate 4. Select a result variable if the next action needs the answer.

> **What you should see** The action returned **12** and stored **12** in the selected game variable. A renamed function or reordered parameter list also needs an updated node definition.

BASIC IN YOUR GAME

Call BASIC and native Godot helpers

A function can belong to another BASIC record, or to a PCA Script, Lua or GDScript record. The caller needs the record ID, function name and arguments in the right order.

```
FUNCTION Run()
    DIM total = CallScript("ticket_price", "Fare", [3, 4], {})
    RETURN total
END FUNCTION
```

The arguments are [3, 4]. The last {} is the optional context dictionary. This call to Fare returns **12**. The script record and function can have different names.

When you need a Godot API

Put this listing in a native **GDScript** record with ID **native_screen**. It asks Godot for the width of the visible game viewport.

```
func Width() -> int:
    return int(Runtime.get_viewport().get_visible_rect().size.x)
```

Put the following in a BASIC record. Let GDScript handle the engine object and return a plain number to BASIC.

```
FUNCTION Run()
    RETURN CallScript("native_screen", "Width", [], {})
END FUNCTION
```

> **What you should see** The Linux preview returned **1280**. Another window or device can have a different width. To run a visual graph, BASIC also accepts **RUN VISUAL SCRIPT "graph_id"** with a matching graph ID.

BASIC IN YOUR GAME

Read rooms and count inventory

Is the player carrying the battery, or does the battery merely exist in your database? Those are two different questions. InventoryCount tells you what the character actually has.

```
FUNCTION Run()
    DIM before = InventoryCount("cell")
    AWAIT AddInventory({"itemId": "cell", "count": 1})
    RETURN InventoryCount("cell") - before
END FUNCTION
```

Here **cell** is an existing inventory item ID. The first line counts what the player already has. After the action, the difference is **1**, whether the count started at zero or higher. Supply the optional character ID to InventoryCount when checking another character.

```
FUNCTION Run()
    RETURN rooms.lab.name
END FUNCTION
```

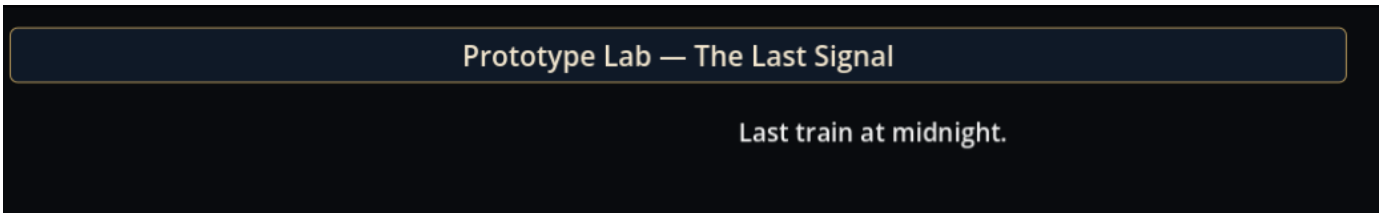
Read this as “the name field on the room whose ID is lab.” In the preview it returned **01 · Blockout Lab**. Use the actual ID from your own room record.

> Other record families include characters, objects, triggers, guis, dialogues, quests, objectives, assets, cameras and spawns. After typing the record name and a dot, use the editor’s completions to look for fields available on that record.

BASIC IN YOUR GAME

Choose the action the story needs

For a simple notice, ShowMessage is enough. When the story calls for a conversation, a camera change or a journey to another room, pick the action that owns that job.



The running game shows “Last train at midnight.” after BASIC calls ShowMessage.

Action fragments for use inside Run

```
AWAIT StartDialogue({"dialogueId": "ticket_clerk"})
AWAIT PlaySound({"assetId": "station_chime"})
AWAIT StartCutscene({"cutsceneId": "train_arrival"})
```

These are separate examples. Replace each quoted ID with an existing record of the matching kind in your game. A sound needs an imported audio asset. A dialogue and a cutscene each need their own authored record.

Change rooms	ChangeScene takes a destination sceneId. Set spawnId when the player should arrive at a particular named spawn.
Move a character	CharacterWalkTo can use characterId and targetId. Choose an existing character and a reachable target in the correct room.
Advance a quest	StartQuest takes questId and can take stageId. Use an objective action when the event only completes one part of an ongoing quest.

BASIC IN YOUR GAME

Save a flag with a game slot

A flag that remembers the signal was seen belongs to the player’s progress. Save slots are useful when two playthroughs can reach different points in the story.

At the point where your game offers Save

```
AWAIT SaveGameSlot({"slot": 3, "saveName": "Station"})
```

This fragment goes inside the function that handles saving. The slot number identifies the destination. Saving again to that slot replaces the previous snapshot. Use the slot the player selected in a real save menu.

At the point where the player chooses Load

```
AWAIT LoadGameSlot({"slot": 3})
```

The load belongs to a separate user action. Loading immediately after saving would only bring back the state the player already sees.

Before saving	The shared Boolean signal_seen from page 8 was TRUE.
After saving	The running game changed signal_seen to FALSE.
After loading slot 3	The game restored the room and signal_seen read TRUE again.

> Use a GLOBAL or shared game variable for story flags. Use the file chapters for settings that should apply across multiple saved games.

BASIC IN YOUR GAME

Track down a runtime problem

A script can pass validation and still receive a value that makes no sense for your game. If a ticket count must be zero or greater, say so in the code.

```
FUNCTION Run()
    DIM count = -1
    TRY
        ASSERT count >= 0
        RETURN "Ready"
    CATCH problem
        RETURN "Check the count"
    END TRY
END FUNCTION
```

ASSERT checks the expression. With count at -1, the check fails and control moves to CATCH. The code returns **Check the count**. With a nonnegative count it returns **Ready**. The name problem holds the failure details.

The script will not validate	Start with the reported line and the line before it. Check quotes, closing parentheses and matching block endings.
No visible result	PRINT goes to the output. For a player-facing response, change a visible GUI label or call ShowMessage.
A click seems dead	Check that the GUI is visible, the button is enabled, its Click target is the script ID and the script is enabled. Start Run with a short PRINT to see whether the click got that far.
An event never arrives	Check the subscription box, the matching handler name and whether the event actually occurred after the listener was available.

DESK REFERENCE

Use the guide across builds

The same BASIC source lives inside the PCA3D game project. The screenshots show PCA3D 0.86.3.4 on Linux. Use the control names when your theme, language or screen layout differs.

PCA3D versions	PCA3D BASIC 1.0 arrived in PCA3D 0.40.0. An earlier editor needs a BASIC-capable release to open the BASIC workspace. Functions as Nodes arrived in 0.81.0. Later actions and fields depend on your installed build.
Windows, macOS and Linux	Use the editor's named controls. File examples use user:// so the engine chooses the right data folder. Generated games include the BASIC frontend and shared runtime.
Touch and smaller screens	Tap the controls named in the screenshots and scroll the workspace or Inspector if necessary. Use a visible submit button so keyboard, touch and pointer users have a clear way to send text.
Mobile and web exports	Use the export tools offered by your edition. Test GUI focus, the on-screen keyboard and a save/load round trip on the intended device. A browser's storage lifetime also depends on the browser and session.

Coming from an older BASIC?

Use named functions in place of numbered lines and GOTO. Read player input through a GUI or game event. Return a value from a function to pass an answer back. Use a GDScript helper for native engine work. These are the entry points PCA3D provides.

DESK REFERENCE

Find an event by name

Use these names in the Event Functions list and in your BASIC handler. To see the exact fields supplied by an event in your version, print its context when the event happens.

Starting a game	OnGameStart, OnGlobalReady
Changing rooms	OnSceneLeaving, OnSceneLoaded
Player and control	OnPlayerChanged, OnCharacterControl
World interactions	OnInteraction, OnTriggerEntered, OnTriggerExited, OnAttachPointContact
Items and inventory	OnItemFound, OnInventoryChanged
Dialogue	OnDialogueStarted, OnDialogueEnded
Media playback	OnVideoEnd, OnMusicEnd
Loading a game	OnSaveLoaded
Your own messages	OnCustomEvent

Choose the event that answers the question directly. For example, to show a line when the player steps into a trigger, subscribe to OnTriggerEntered. A room-load handler would run much earlier, when the room first appears.

DESK REFERENCE

Find a helper for a common job

A helper answers a question or performs a small operation. A gameplay action changes what the game does. You can use both in the same function.

Text	Lower, Upper and Trim prepare an answer. Replace edits a piece of text. Split and Join move between strings and lists.
Numbers and positions	Clamp keeps a value in bounds. RandomInt picks a whole number in an inclusive range. Distance and Vector3 are useful for positions.
GUI	GetGUIElementProperty(gui, element, property, fallback) reads one control. SetGUIElementProperty(gui, element, property, value) updates it. SetGUIProperty can change the whole GUI.
Stored game data	ReadTextFile and ReadJSONFile accept a fallback. WriteTextFile and WriteJSONFile report whether the write succeeded. FileExists tests a path.
Story progress	InventoryCount(item, character) counts a character’s items. HasVisited(room, character) checks room history. The character argument is optional for both.
Communicate	CallScript passes an argument list to a named function. EMIT EVENT sends a dictionary to a listener. RETURN sends one result to the caller.

DESK REFERENCE

Read the action index

On the following pages, look up an action by its BASIC name. The words beside it are dictionary keys from the shared action editor in PCA3D 0.86.3.4.

```
FUNCTION Run()
    AWAIT ShowMessage({"text": "The gate is shut.", "duration": 2})
END FUNCTION
```

Find **ShowMessage**. Two of its keys are text and duration. The text key takes the words to display. The duration key takes a number of seconds. Optional settings can keep their defaults.

There are 206 actions in this build’s scripting catalog. The index lists the fields supplied by the shared editor schema. Some actions also use a separately selected target, such as the destination sceneId for ChangeScene. An empty field entry means the shared schema lists no keys for that action.

> Use the action inspector in your installed edition for choices, allowed values and defaults. Record IDs refer to records in your own project. You can search this PDF for either an action name or a field name.

ALPHABETICAL ACTION INDEX

Action names and fields

ActivateCamera → CameraShake

BASIC action	Dictionary fields
ActivateCamera	cameraId, duration, wait
AddAssetToScene	assetId, id, name, position, enabled, visible
AddCharacterToScene	characterId, id, name, position, enabled, visible, animation
AddCharacterWaypoint	characterId, sceneId, waypointId, position, ignoreNavigation
AddDialogueChoice	dialogueId, nodeId, choiceId, text, next, enabled, repeatable, voiceAssetId, voiceKey, typewriter, typewriterSpeed, typewriterSound, typewriterSoundAssetId
AddDialogueChoiceNode	dialogueId, nodeId
AddDialogueLine	dialogueId, nodeId, speakerId, text, next, returnToPrevious, voiceAssetId, voiceKey, typewriter, typewriterSpeed, typewriterSound, typewriterSoundAssetId
AddInventory	itemId, count
AddScore	amount
AlignAttachPoints	entityId, point, targetId, targetPoint, alignRotation
AttachObject	entityId, point, targetId, targetPoint
AttachParticleEffect	entityId, targetId, targetPoint, offset
AttachPoints	entityId, point, targetId, targetPoint, alignRotation, suspendMovement
AttachPointsTouching	entityId, point, targetId, targetPoint, tolerance, resultVariable
CallScriptFunction	functionKey, parameters, resultVariable
CameraOrbit	entityId, speed, distance, mode, repetitions, clockwise, height
CameraShake	direction, duration, power

ALPHABETICAL ACTION INDEX

Action names and fields

CancelPreposition → ChangeQuestObjectiveProperty

BASIC action	Dictionary fields
CancelPreposition	No fields listed
CarouselAddImage	guiId, elementId, assetId
CarouselClearImages	guiId, elementId
CarouselNext	guiId, elementId
CarouselPrevious	guiId, elementId
CarouselRemoveImage	guiId, elementId, assetId
CenterGUIModel	guiId, elementId
ChangeCamera	sceneId, cameraId
ChangeCharacterDescription	characterId, description
ChangeCharacterProperty	characterId, property, value
ChangeGameCursor	cursorId
ChangeGameSetting	setting, value
ChangeGUIElementProperty	guiId, elementId, property, value
ChangeGUIProperty	guiId, property, value
ChangeInventoryDescription	itemId, description
ChangeInventoryQuantity	characterId, itemId, amount
ChangeObjectDescription	sceneId, entityId, description
ChangeObjectProperty	sceneId, entityId, property, value
ChangeProjectProperty	property, value
ChangeQuestObjectiveProperty	questId, stageId, objectiveId, property, value

ALPHABETICAL ACTION INDEX

Action names and fields

ChangeQuestProperty → CreateVariable

BASIC action	Dictionary fields
ChangeQuestProperty	questId, property, value
ChangeQuestStageProperty	questId, stageId, property, value
ChangeRegionDescription	sceneId, regionId, description
ChangeScene	spawnId, position
ChangeSceneDescription	sceneId, description
ChangeSubtitleLanguage	language
ChangeVariable	name, operation, value
ChangeVerb	verbId
ChangeVoiceLanguage	language
CharacterTeleport	characterId, sceneId, targetId, position
CharacterWalkPath	characterId, sceneId, waypointIds, speed, ignoreNavigation
CharacterWalkTo	characterId, sceneId, targetId, position, animation, speed
CharacterWalkWaypoints	characterId, speed
ClearCharacterWaypoints	characterId
CloseInteractionCoin	No fields listed
CompleteObjective	questId, stageId, objectiveId
CompleteQuest	questId
CreatePrimitive	primitive, id, position
CreateProcedural	procedural, id, position
CreateVariable	name, variableType, value

ALPHABETICAL ACTION INDEX

Action names and fields

CycleInteractionVerb → FollowCharacter

BASIC action	Dictionary fields
CycleInteractionVerb	direction
DespawnObject	sceneId, entityId
DetachObject	entityId
DetachParticleEffect	entityId
DisableGUI	guiId
DisableGUIElement	guiId, elementId
EnableGUI	guiId
EnableGUIElement	guiId, elementId
EndCutscene	restoreCamera
EndDialogue	No fields listed
ExecutePreposition	verbId, sourceId, targetId
FaceCharacter	characterId, targetCharacterId
FaceCoordinates	characterId, position
FaceObject	characterId, sceneId, targetId
FadeMusic	volume, duration
FadeScreen	direction, color, duration
FadeSound	channel, volume, duration
FailObjective	questId, stageId, objectiveId
FailQuest	questId
FollowCharacter	characterId, targetCharacterId, distance, speed

ALPHABETICAL ACTION INDEX

Action names and fields

GetAttachPoint → PauseFMV

BASIC action	Dictionary fields
GetAttachPoint	entityId, point, resultVariable
GetAttachPointContacts	entityId, point, targetId, targetPoint, tolerance, resultVariable
GetControlState	resultVariable
HideGUI	guiId
HideGUIElement	guiId, elementId
HidePlayerCharacter	hidden
InteractNearest	verbId
IsCurrentRoom	sceneId
Jump	characterId
ListBoxAddItem	guiId, elementId, itemId, text, iconAssetId, disabled, insertIndex
ListBoxClearItems	guiId, elementId
ListBoxDeleteItem	guiId, elementId, itemId
ListBoxModifyItem	guiId, elementId, itemId, property, value
LoadGame	No fields listed
LoadGameSlot	slot
MeltSnow	sceneId, duration, pace, waitForCompletion
MoveNPCToRoom	characterId, sceneId, spawnId, position
OpenInteractionCoin	entityId
PauseAnimation	sceneId, entityId
PauseFMV	No fields listed

ALPHABETICAL ACTION INDEX

Action names and fields

PauseMusic → ResumeMusic

BASIC action	Dictionary fields
PauseMusic	No fields listed
PauseSound	channel
PauseVoice	No fields listed
PlayAnimation	sceneId, entityId, clip
PlayGameplayRecording	path, speed
PlayGUIModelAnimation	guiId, elementId, animation, loop, speed
PlayMusic	assetId, skippable
PlaySound	assetId, channel
PlayVideo	assetId, fullScreen, skippable, muted
PlayVoice	voiceAssetId, voiceKey, text, wait
RemoveAttachPoint	entityId, point
RemoveDialogueChoice	dialogueId, nodeId, choiceId
RemoveDialogueNode	dialogueId, nodeId
RemoveInventory	itemId, count
ResetDialogueHistory	dialogueId, nodeId, choiceId
ResetObjective	questId, stageId, objectiveId
ResetQuest	questId
ResumeAnimation	sceneId, entityId
ResumeFMV	mode, endTime, scriptId
ResumeMusic	No fields listed

ALPHABETICAL ACTION INDEX

Action names and fields

ResumeSound → SetAutoInputSwitching

BASIC action	Dictionary fields
ResumeSound	channel
ResumeVoice	No fields listed
RunVisualScript	scriptId
SaveGame	No fields listed
SaveGameSlot	slot, saveName
SaveGameWidgetCommand	guiId, elementId, command, slot
Say	speakerId, text, waitForContinue, voiceAssetId, voiceKey, typewriter, typewriterSpeed, typewriterSound, typewriterSoundAssetId
ScreenTransition	transitionType, duration, color, direction, intensity
SeekFMV	time, mode, endTime, scriptId
SeekMusic	minutes, seconds
SeekVideo	minutes, seconds
SeekVoice	seconds
Set2DCamera	sceneId, position, zoom, rotation
Set2DEntityDepth	sceneId, entityId, depth
Set2DLayerVisible	sceneId, layerId, visible
Set2DParallax	sceneId, layerId, ratio
SetAttachPoint	entityId, point, position, rotation, bone, space, contactRadius, enabled
SetAutoInputSwitching	enabled

ALPHABETICAL ACTION INDEX

Action names and fields

SetCameraBehavior → SetMovementProfile

BASIC action	Dictionary fields
SetCameraBehavior	cameraId, enabled, mode, targetId, targetIds, lookAtId, targetOffset, damping, blendDuration, ease, priority, autoFrame, minDistance, maxDistance, framingPadding, deadZoneWidth, deadZoneHeight, pathPoints, respectBounds
SetCameraPriority	cameraId, priority
SetCameraTargets	cameraId, targetId, targetIds, lookAtId
SetCharacterActionEnabled	characterId, actionEvent, enabled
SetControlSetting	setting, value
SetDialogueChoiceEnabled	dialogueId, nodeId, choiceId, enabled
SetFlashlightMode	enabled, strength, radius, shape
SetGameLanguage	language, persist
SetGlow	entityId, enabled, variable
SetInputDevice	device
SetInteractionApproachEnabled	enabled, distance, faceTarget
SetInteractionMode	mode
SetLipSyncEnabled	characterId, enabled
SetMotion	sceneId, entityId, enabled, variable
SetMouseClicksEnabled	enabled
SetMousePointerVisible	visible
SetMousePointingEnabled	enabled
SetMovementProfile	profile

ALPHABETICAL ACTION INDEX

Action names and fields

SetObjectActionEnabled → SetVerbUsageEnabled

BASIC action	Dictionary fields
SetObjectActionEnabled	sceneId, entityId, actionEvent, enabled
SetObjectEnabled	sceneId, entityId, enabled
SetObjectInteractionApproach	sceneId, entityId, mode, distance, faceTarget
SetObjectVisible	sceneId, entityId, visible
SetPlayerInputEnabled	enabled
SetProximityInteractionEnabled	enabled
SetQuestHidden	questId, hidden
SetQuestStage	questId, stageId
SetQuestTracked	questId, tracked
SetRetroStyle	style, strength, pixelSize
SetRoomCameraBounds	sceneId, enabled, minX, maxX, minZ, maxZ
SetRoomDimension	sceneId, dimension, keep3DLayer
SetRuntimeControlsEnabled	enabled
SetSaveSlotImage	assetId
SetSceneBrightness	sceneId, brightness
SetSceneTint	sceneId, tintColor, strength
SetSnowAccumulation	sceneId, amount, duration, pace, waitForCompletion
SetTriggerEnabled	sceneId, entityId, enabled
SetVariable	name, value
SetVerbUsageEnabled	enabled

ALPHABETICAL ACTION INDEX

Action names and fields

SetVoiceSpeaker → StopMusic

BASIC action	Dictionary fields
SetVoiceSpeaker	characterId
ShakeScreen	direction, duration, power
ShowGUI	guiId
ShowGUIElement	guiId, elementId
ShowMessage	text, duration, blockInput, voiceAssetId, voiceKey, typewriter, typewriterSpeed, typewriterSound, typewriterSoundAssetId
SpawnObject	sceneId, entityId
StartCutscene	cutsceneId
StartDialogue	dialogueId
StartFMV	assetId, mode, endTime, scriptId
StartGameplayRecording	path
StartQuest	questId, stageId
StopAnimation	sceneId, entityId
StopAttachPointWatch	watchId
StopCameraOrbit	No fields listed
StopFMV	No fields listed
StopFollowCharacter	characterId
StopGameplayPlayback	No fields listed
StopGameplayRecording	No fields listed
StopMusic	No fields listed

ALPHABETICAL ACTION INDEX

Action names and fields

StopScreenShake → ZoomBlur

BASIC action	Dictionary fields
StopScreenShake	No fields listed
StopSound	channel
StopStartupSplashAudio	No fields listed
StopVideo	No fields listed
StopVoice	No fields listed
StopZoomBlur	No fields listed
SwitchCharacter	characterId
ToggleGlow	entityId, variable
ToggleMotion	sceneId, entityId, variable
ToggleRGBString	entityId, variable
TransferInventory	fromCharacterId, toCharacterId, itemId, count
UnlockHint	topicId, answerNumber
WatchAttachPointContacts	watchId, entityId, point, targetId, targetPoint, tolerance, swept
ZoomBlur	strength, duration, centerX, centerY, clearRadius, direction, includeGUI, wait