PCA 3D ADVENTURE STUDIO

Panels, Tabs and List Boxes

Build a notebook your player can actually use

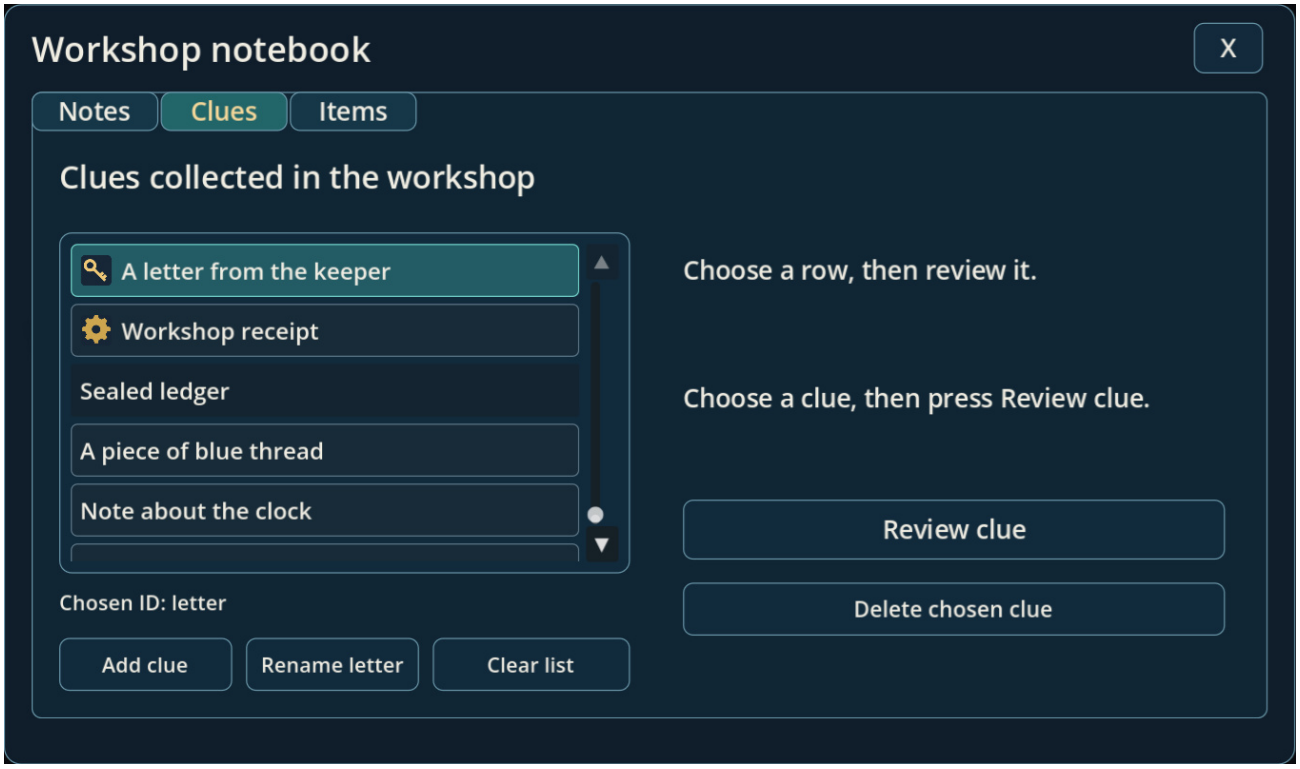

Figure 1. A running notebook with linked pages, a selectable clue list and a separate Review button.

A practical, step-by-step guide for Windows, macOS and Linux.

Editor reference: 0.86.3.4 • Companion to the GUI Designer tutorials

Before you begin

This guide starts with an ordinary panel, turns a panel into a set of tabs, then builds lists that the player can select and use. The running example is a workshop notebook with Notes, Clues and Items pages.

You can follow along in an existing project. Use a copy if you want to experiment. No separate practice project is needed, and none is supplied with this guide. The screenshots come from the editor and its game preview.

What you should already know

Be comfortable opening the GUI Designer, adding a Label or Button, changing an element’s position and saving your project. The earlier GUI tutorials cover those first steps. The GDScript examples here are complete small scripts, with their setup explained beside them.

When the guide says…	It means…
GUI	A whole interface, such as a notebook window or one notebook page.
Panel	An element inside a GUI. It can draw a surface or hold linked tab pages.
List box	An element that displays rows and keeps track of selected row IDs.
ID / name	The ID is a stable reference. The name is the readable label shown in the editor.

The field names and behavior were checked against 0.86.3.4. Later builds may move a control or change a limitation. The workflow is the same across desktop platforms; window decorations and file dialogs can look different.

> Element property fields normally update live. Whole-GUI settings and script editors have their own Apply controls. After applying those changes, use Save Project to keep the authoring work.

Contents

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

1. The notebook we are building 4
2. Find the controls in the designer 5
3. Make an ordinary panel 6
4. Plan the host and its pages 9
5. Create the page GUIs first 10
6. Turn a panel into tabs 11
7. Fill and test the linked pages 13
8. Style the tab strip 14
9. Keep page behavior predictable 16
10. The older child-element tab method 17
11. Add your first list box 19
12. Understand what a selection returns 21
13. Make the Review Button work 23
14. Give rows room to breathe 24
15. Icons, text and wildcard limits 26
16. Scrolling and navigation 27
17. Allow more than one selection 29
18. Use the list as a dropdown 30
19. Fill a list from an Array 32
20. Display the carried inventory 35
21. Change manual rows with actions 37
22. Delete the row the player chose 40
23. Use the same actions as visual nodes 41
24. A validation issue in this release 42
25. Open, close and restyle the notebook 43
26. Preview, saving and restoring state 44
27. Put the full notebook together 45
28. When a panel or tab looks wrong 46
29. Keep these references handy 48

1. The notebook we are building

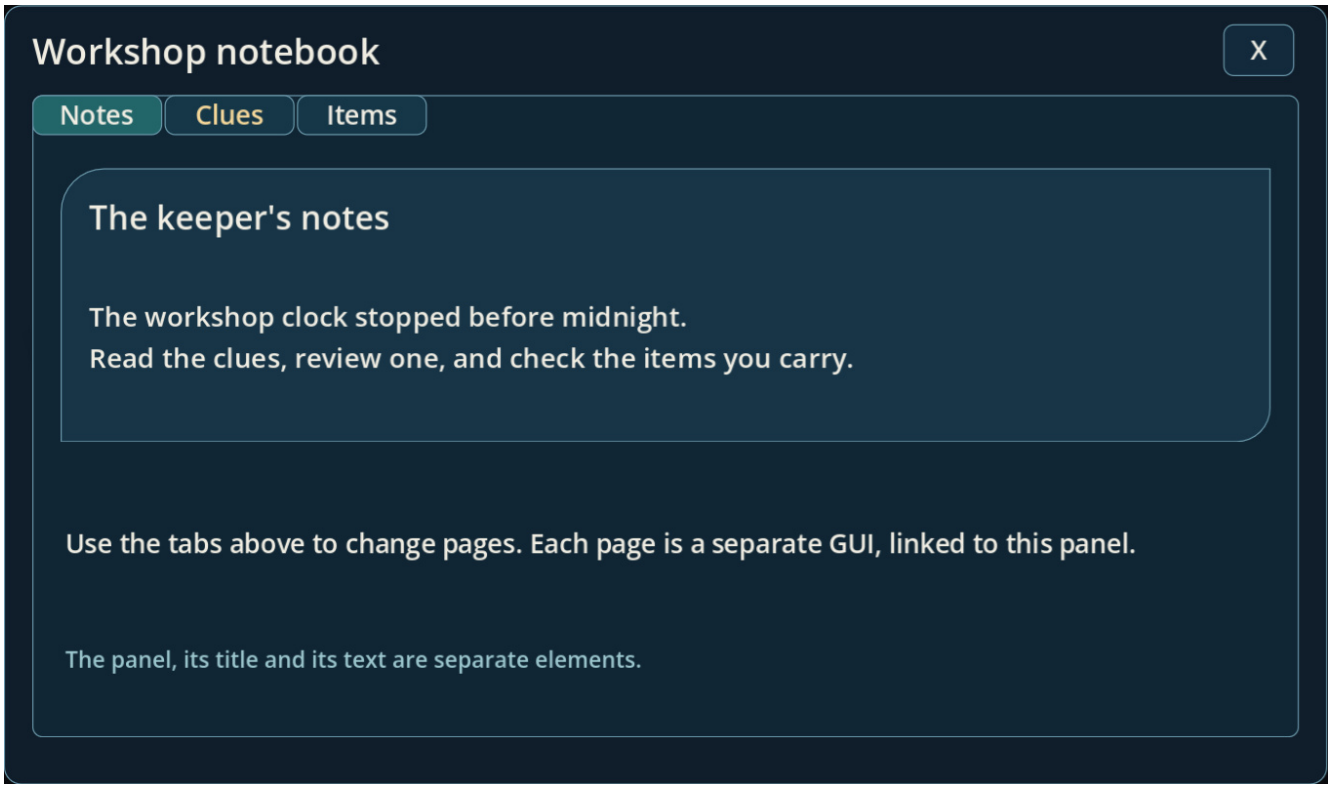

Figure 2. The Notes page in the running game. The title and Close button belong to the host GUI; the page content comes from another GUI.

Think of the notebook as a frame with interchangeable pages. The host GUI supplies the title, Close button and tabbed panel. Each tab points to a separate content GUI. Clicking Clues replaces the Notes content without opening a second notebook window.

Part	Example ID	Job
Host GUI	case_notebook	Opens and closes the whole notebook.
Tabbed panel	notebook_tabs	Draws the tab strip and displays the chosen page.
Page GUIs	notes_page, clues_page, items_page	Hold each page’s labels, lists and buttons.
Clue list	clue_list	Displays rows and writes the chosen row ID to a variable.

The same arrangement works for a journal, character sheet, quest log, help screen or shop. Start with one page that works, then add more.

2. Find the controls in the designer

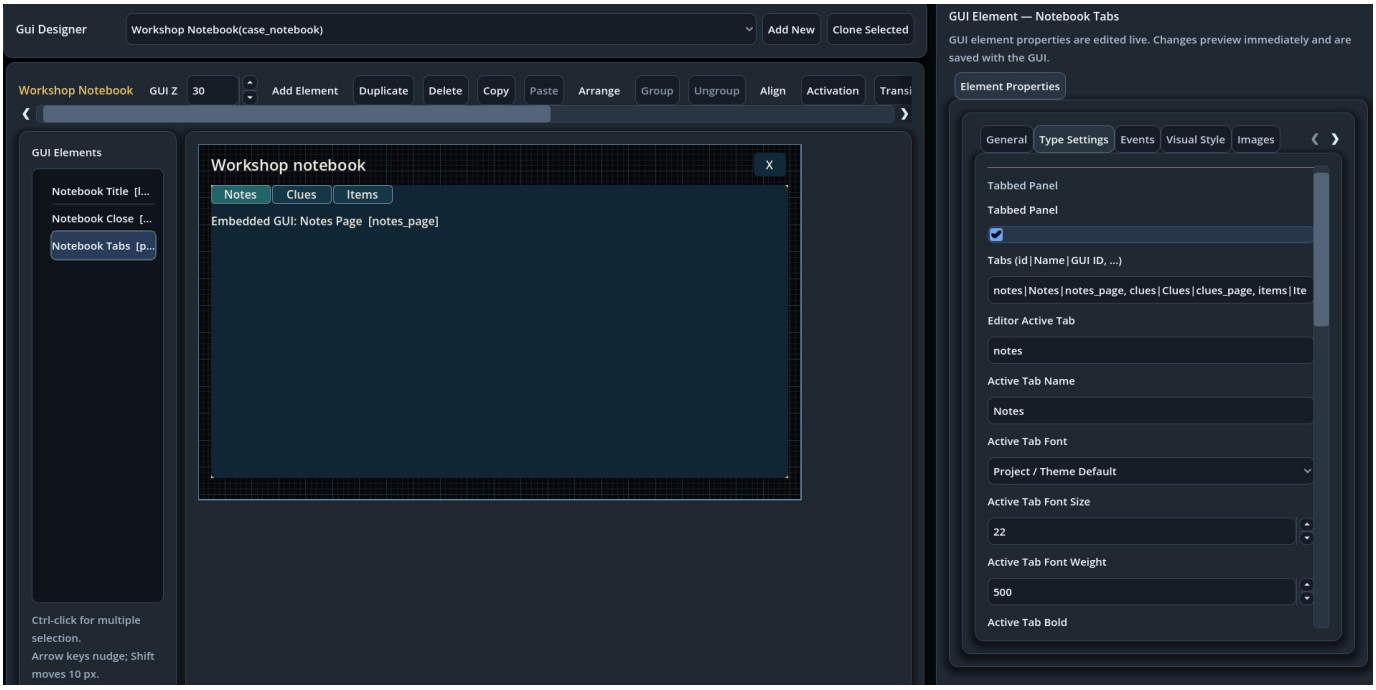

Figure 3. The GUI chooser is at the top, the element list is on the left, the canvas is in the middle and the selected element’s properties are on the right.

1. Open the GUI Designer and choose the GUI you want to edit from the top dropdown.
2. Use Add Element to add a Panel or List Box. Select it in the GUI Elements list if the canvas is crowded.
3. Use General for identity and geometry. Use Type Settings for tabs or list behavior. Use Visual Style for surface appearance and Images for image assets.
4. Save Project after a useful stage of the work. Test a running game before relying on data or events.

Canvas zoom only changes your editing view. It does not change the GUI’s authored width, height or runtime size. The notebook screenshots use a reduced editor zoom so the entire design fits beside the Inspector.

> The GUI Designer’s Preview switch is useful for local interaction. It is not a replacement for Play / Restart or Test Scene when you need live variables, inventory or actions.

3. Make an ordinary panel

A plain panel is a useful backing for text, buttons or a portrait. It draws a surface; it does not turn its own Text field into a heading. Add a Label for that.

1. Choose your page GUI, click Add Element and choose Panel.
2. Give the panel a clear ID, such as summary_panel, and an editor name such as Summary Panel.
3. In General, set Anchor to top-left. Enter X 24, Y 28, Width 1024 and Height 232 for the example Notes page.
4. Set the panel’s Z Index below the labels you will place over it. In the example, the panel is 0 and the labels are 1.

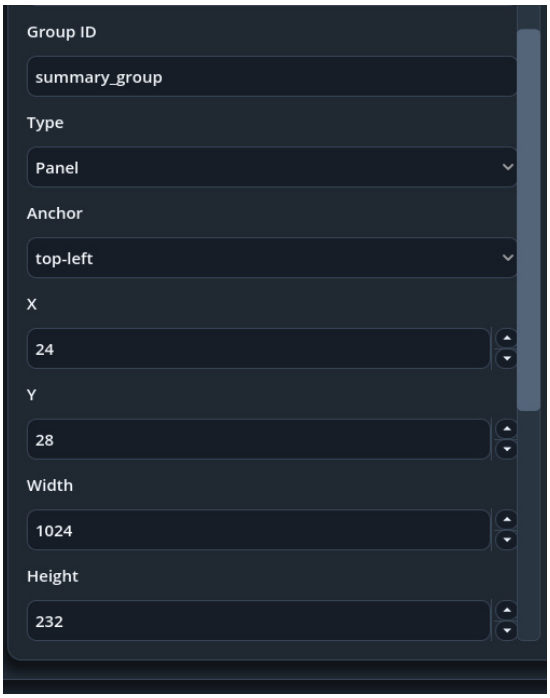

Figure 4. General holds the panel’s group, anchor and dimensions. These coordinates belong to its GUI.

A panel does not automatically collect nearby elements as children. A Label drawn over a plain panel remains a separate element in the GUI.

Shape the surface and its corners



Figure 5. One panel, four corner overrides. The example uses a curved top-left and bottom-right, with square opposite corners.

Field	How to use it
Border Radius	The common radius used when a corner has no override.
Top Left / Top Right / Bottom Right / Bottom Left Radius	Set each corner separately. -1 inherits the common radius; 0 makes a square corner; a positive number makes a curve.
Surface / border opacity	Controls how much of the surface or border is visible.
Surface and border color	Choose a readable contrast with the text and the room behind the GUI.

For the Notes card, use common radius 12, Top Left 36, Top Right 0, Bottom Right 28 and Bottom Left 0. The result is deliberately uneven so you can see which corner each field changes.

Choose restrained colors first. A very transparent panel over a busy room can make perfectly good text hard to read.

Images, layers and moving a panel layout



Figure 6. The plain Panel’s Images tab. The exact image fields vary by element type.

Background Image supplies the panel’s artwork. Media Asset can provide its content fallback. Hover Image is the alternate image while the pointer is over the surface. Background Fit controls how the artwork fits the rectangle. Use an image already imported into the project.

Do not assume every element offers a clicked image. In this build the plain Panel screen shown here exposes Media Asset, Background Image, Hover Image and Background Fit. Buttons and other element types have their own state-image options.

Add a heading Label at X 48, Y 48 and a body Label at X 48, Y 108. These positions are measured from the page GUI’s origin, not from summary_panel. Keep their Z Index above the panel.

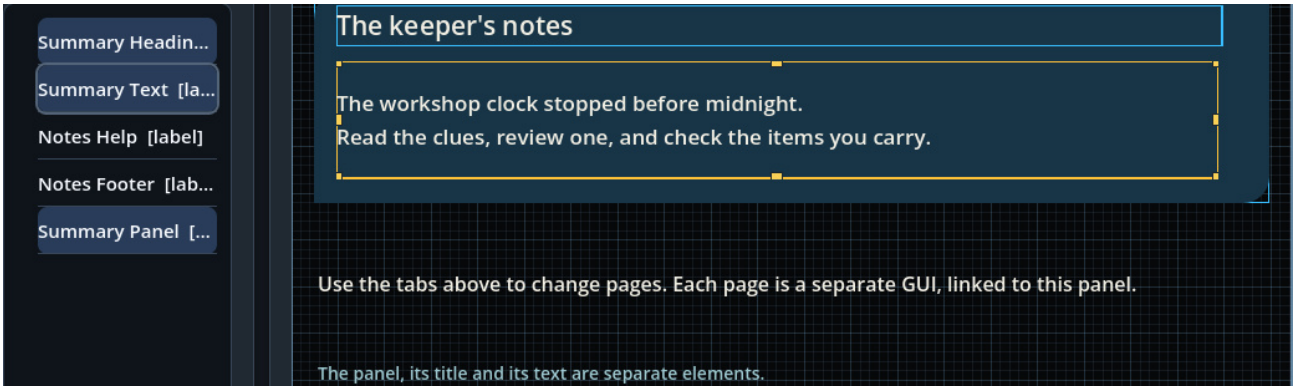

Figure 7. The panel and its two labels are selected together. Grouping keeps the arrangement convenient to edit.

Ctrl-click the panel and the labels in the element list, then use Group. A shared Group ID helps you move the arrangement together. Grouping is an editing relationship; it is separate from a tab’s page link or the older Parent Panel fields.

4. Plan the host and its pages

Set up the IDs before you start linking tabs. The three references below look similar, but each has a different job.

Reference	Example	Where it belongs
Panel ID	notebook_tabs	General → ID on the panel element.
Local tab ID	clues	First part of one entry in the Tabs field.
Page GUI ID	clues_page	Third part of that tab entry; identifies a whole GUI.
Visible tab caption	Clues	Second part of the entry. Players read this.

Editor Active Tab takes the local tab ID: clues. It does not take clues_page or notebook_tabs. The readable caption can change later without breaking references if you keep the IDs stable.

Layout	Example dimensions
Project GUI canvas	1280 × 720
Notebook host GUI	1120 × 660, centered
Panel inside the host	X 24, Y 76, Width 1072, Height 544
Each content page GUI	1072 × 510

The panel’s tab strip uses some of its height. With a tab font size of 22, the strip is 34 pixels high in this example, leaving 510 for page content. Leave room for the host title and Close button above it.

> Use unique GUI IDs and unique element IDs throughout the project. This guide uses distinctive IDs even on different page GUIs, which avoids ambiguous runtime references.

5. Create the page GUIs first

1. In the GUI Designer, use Add New to create Notes Page, with ID notes_page.
2. Open GUI Window settings. Set GUI Width 1072 and GUI Height 510.
3. Turn off the window background, border frame and title bar for this embedded page. It will sit inside the host panel.
4. In the GUI’s runtime activation settings, use trigger-controlled activation rather than a page that opens itself at game start.
5. Repeat for Clues Page (clues_page) and Carried Items Page (items_page).

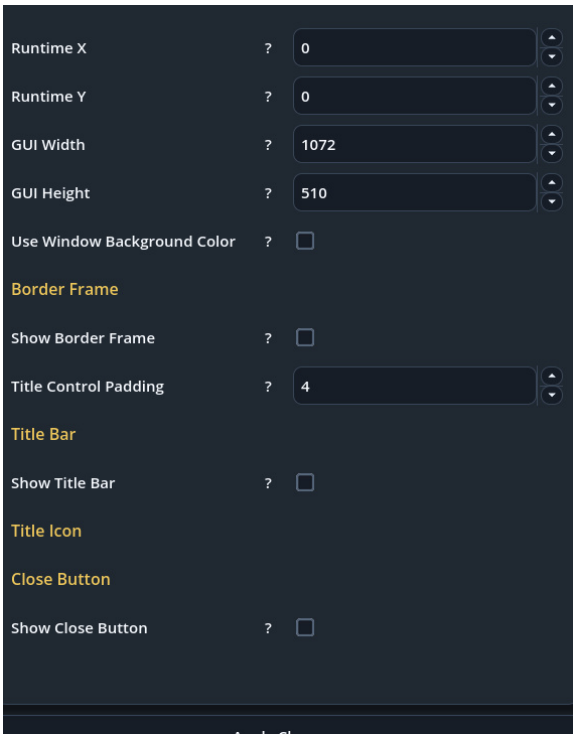

Figure 8. The Clues page’s GUI Window settings. Its canvas matches the usable space below the host tab strip.

Apply Changes for the GUI record. The page’s controls will use positions measured from that page’s own top-left corner. You can edit a page independently before the host is finished.

6. Turn a panel into tabs

1. Create the host GUI, Workshop Notebook, with ID case_notebook. Make its window 1120 × 660 and center it.
2. Add a title Label and a Close Button. Add a Panel with ID notebook_tabs at X 24, Y 76, Width 1072, Height 544.
3. Select the panel, open Type Settings and enable Tabbed Panel.
4. Enter the three tab entries shown on the next page. Set Editor Active Tab to notes.
5. Give the tab font a size of 22. Keep the host trigger-controlled so you can open it deliberately from a button or action.

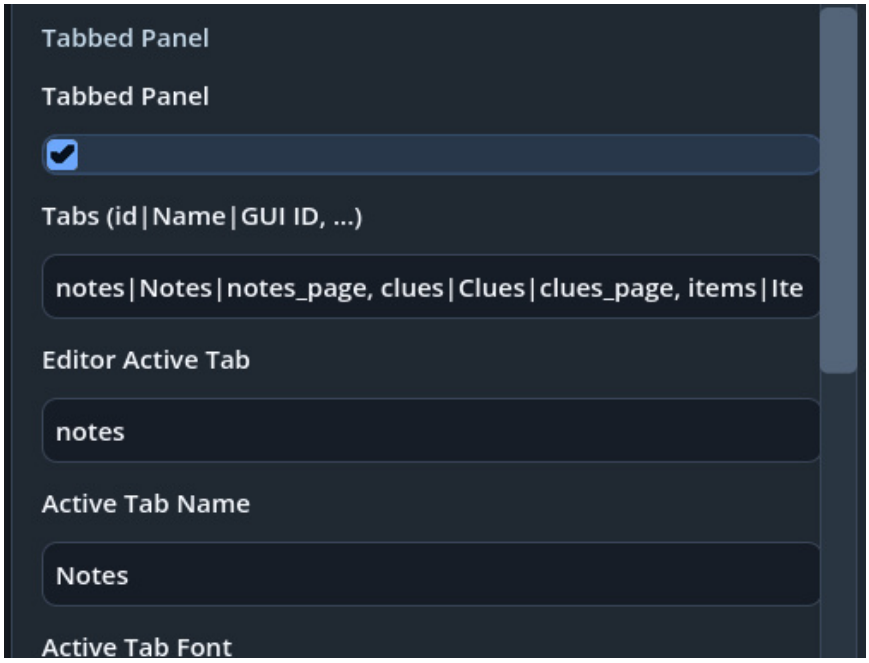

Figure 9. Tabbed Panel enables the tab strip. Tabs supplies the links; Editor Active Tab sets the initial page.

For a notebook that should block interaction with the room while open, use a modal host GUI. The embedded content GUIs can remain nonmodal; the host is the window the player opens and closes.

Write the tab links correctly

Each entry has three parts: local tab ID, visible caption and page GUI ID. Separate the parts with a vertical bar. Separate entries with a comma. Enter this as one line in Tabs:

```
notes|Notes|notes_page, clues|Clues|clues_page,
items|Items|items_page
```

Entry	What it does
notes|Notes|notes_page	Makes the Notes tab display the notes_page GUI.
clues|Clues|clues_page	Makes the Clues tab display the clues_page GUI.
items|Items|items_page	Makes the Items tab display the items_page GUI.

The line break above is for readability. The parser also accepts semicolons or newlines between entries. Use simple IDs without spaces. Avoid commas, semicolons and vertical bars inside captions because they are separators.

What the designer shows

The host canvas shows a hint such as Embedded GUI: Notes Page [notes_page]. That is expected. It identifies the link; it does not render the page’s actual controls inside the host designer. Choose Notes Page in the GUI dropdown to edit those controls.

> A page GUI can be embedded only once in the running interface. Do not assign the same page GUI to several host panels, link the host to itself or make circular page links. Clone the page GUIs as well as the host when you need an independent second notebook.

7. Fill and test the linked pages

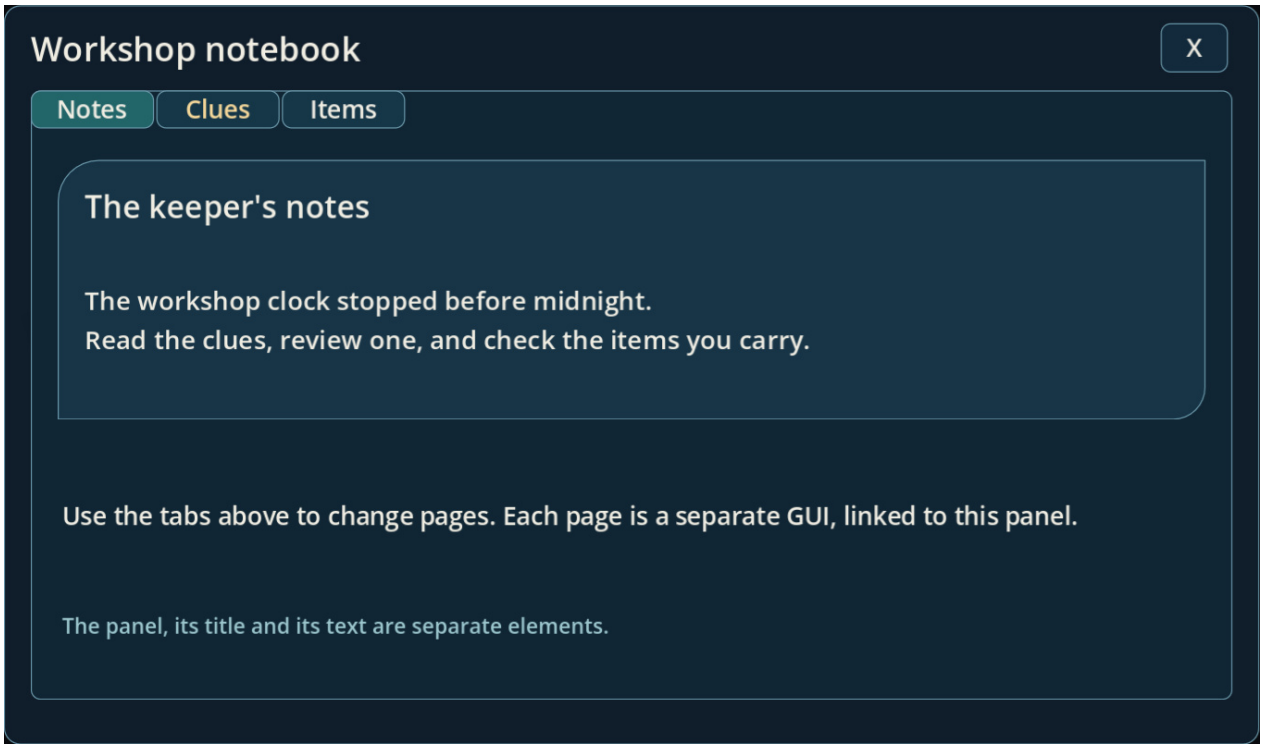

Figure 10. At runtime, the linked Notes page replaces the designer’s placeholder hint.

1. Choose Notes Page in the GUI dropdown. Add the plain Summary Panel and the two labels described earlier.
2. Choose Clues Page and add a heading, a List Box and a Review Button. Leave the list setup for Chapter 11.
3. Choose Carried Items Page and add its heading and List Box.
4. Save the project, start a game preview and open the host GUI. Click Notes, Clues and Items to check that each page is different.

The linked page is fitted to the content rectangle below the tabs. This does not scale every control to make a mismatched design fit. Author the page at the intended size and keep its elements within that area.

> Open and close case_notebook, the host. Showing clues_page independently is not the same as selecting Clues; it can put page content over the wrong place.

8. Style the tab strip

Start with the shared tab style, then change a named tab only when it needs to stand out. Short captions such as Notes, Clues and Items are easier to fit than long sentences.

Type Settings fields	What they change
Default Tab Font; Tab Font Size / Weight	The shared font and size for tabs without a named override.
Tab Bold / Italic / Underline / Strikethrough	Shared text decoration. Use it sparingly on navigation.
Tab Text Color / Opacity	Shared caption color and visibility.
Tab Background	The ordinary tab surface.
Tab Hover Background	The surface under the pointer.
Tab Selected Background	The surface of the currently selected tab.
Tab Border Color	The tab outline.

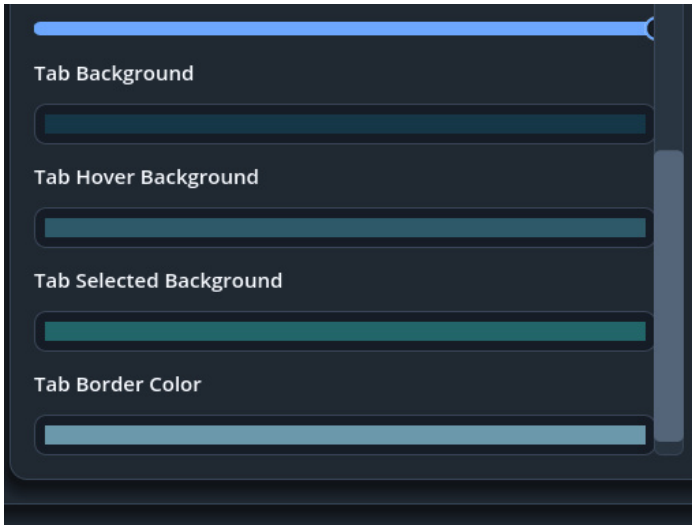

Figure 11. The common text and state-color controls. Selected Background is the active-page cue.

Use enough difference between the ordinary, hover and selected backgrounds that the player can tell which page is open without reading the content. Long tab strips do not automatically become a tidy second row; allow enough width and test your longest translated captions.

Style one named tab

The labels Active Tab Font and Active Tab Name can be misleading. Here, “Active” identifies the tab chosen for editing in Editor Active Tab. These fields store an override for that named tab.

1. Select notebook_tabs and enter clues in Editor Active Tab.
2. Use Active Tab Name to change that tab’s visible caption if needed.
3. Set Active Tab Text Color to a warm gold and enable Active Tab Bold for the Clues tab.
4. Return Editor Active Tab to notes if Notes should open first. Save and test.



Figure 12. Clues has its own gold caption. The selected background is a separate state setting.

The gold caption belongs to Clues even when Notes or Items is selected. It is not a generic “selected tab font” that moves to whichever tab the player clicks. The selected background does move with the active page.

Change	Keeps its reference?
Rename Clues to Evidence	Yes, if local ID clues and GUI ID clues_page stay the same.
Change local ID clues to evidence	Update Editor Active Tab and any legacy tab assignments that used clues.
Change clues_page to another GUI ID	The tab now displays that other page. Check the page exists.

Clicking a tab in designer Preview is useful for checking the local appearance. Set Editor Active Tab deliberately when you want to save the initial page. Do not assume a preview click changed that saved setting.

9. Keep page behavior predictable

In the running game, switching tabs changes which linked page is visible. The hidden pages are still part of the interface. A clue selection can remain selected when you leave Clues and come back during the same run.

What to test	Expected result
Click each tab	Only that tab’s page content is shown inside the panel.
Close the notebook	The host and its embedded pages disappear together.
Open the notebook again	The same widget instances can retain their current state.
Restart the game preview	Runtime-only changes begin again from authored data and startup logic.
Resize the GUI or change tab font size	No labels, dropdowns or buttons are clipped below the tab strip.

If you need the notebook to reopen at a particular page every time, test that requirement separately. The typed Editor Active Tab field is an initial authoring setting. This build does not provide a dedicated shared Set Active Tab action that this guide can recommend for safely switching an already embedded page at runtime.

Avoid changing the host panel’s tab properties through the general Change GUI Element Property action during play. Rebuilding an embedded host can disturb its attached content. Use the visible tab buttons for navigation in this workflow. Ordinary panel appearance changes are covered later.

> Do not hide linked pages with unrelated page-level show/hide actions. Keep notebook opening and closing on the host so the tab system stays in charge of its own pages.

10. The older child-element tab method

Existing projects may use the fields Legacy Parent Panel ID and Legacy Panel Tab ID. This is a different way to make tabs, without separate content GUIs. It is still useful to recognize when editing older layouts.

1. Make a tabbed Panel with ID legacy_tabs. Use Tabs: one|Page one, two|Page two. Notice there is no third part linking a GUI.
2. Add a Label in the same GUI. In General, set Legacy Parent Panel ID to legacy_tabs and Legacy Panel Tab ID to one.
3. Add another Label for the second page. Set its Parent Panel ID to legacy_tabs and its Tab ID to two.
4. Set the panel’s Editor Active Tab to one, then test both tab buttons in a running game.



Figure 13. The two legacy link fields on a Label. Enter the panel element ID and the local tab ID.

These Labels still use whole-GUI coordinates. Their X and Y values do not become offsets inside the panel, and Group ID is not a substitute for either link field.

Legacy pages: what changes and what does not

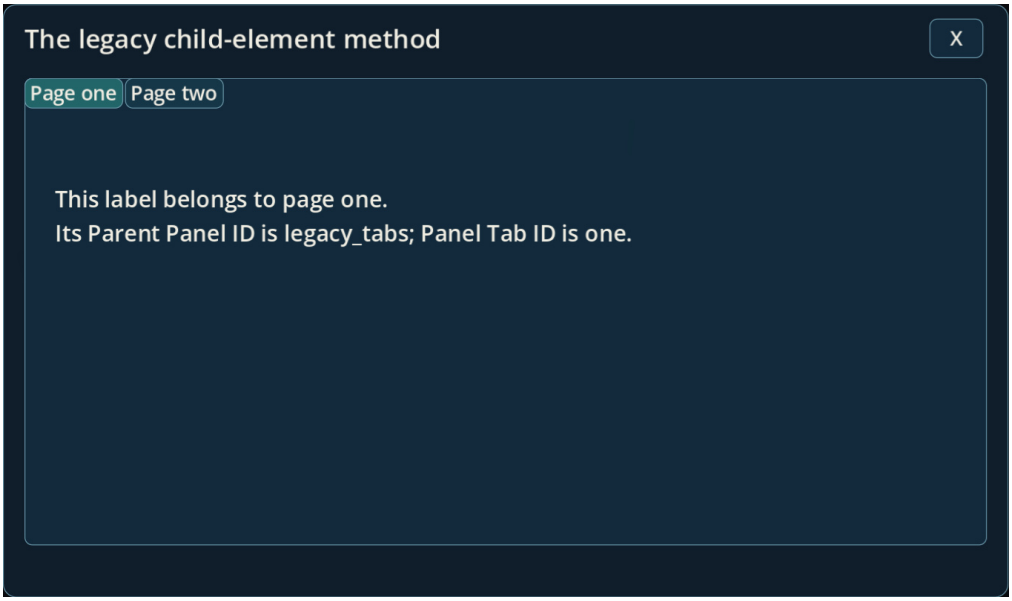

Figure 14. Page one shows its assigned Label. The other page’s Label is hidden.

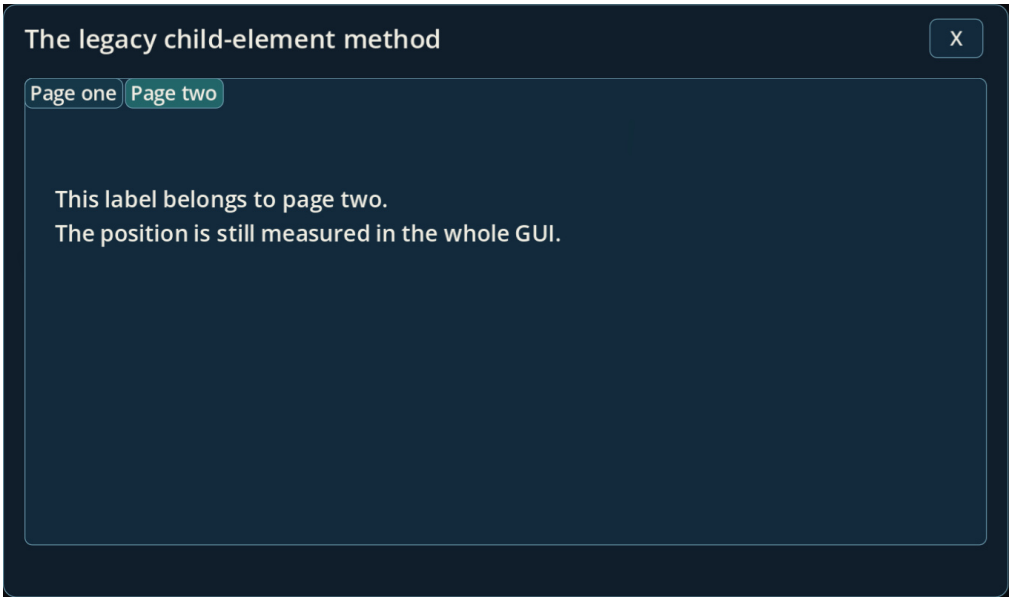

Figure 15. Page two shows a different Label in the same area.

Use separate linked GUI pages for new, substantial interfaces. Keep the legacy approach for existing same-GUI layouts or a small page arrangement that already works. Do not mix legacy children into a tab that also links a GUI: that tab displays its linked GUI instead of those assigned legacy controls.

11. Add your first list box

A list box is a set of rows inside one element. Each row has an ID, visible text, an optional icon and a disabled state. Begin with a manual list so you can see exactly where the rows come from.

1. Open Clues Page and add a List Box. Give it ID clue_list.
2. In General, use X 24, Y 88, Width 495 and Height 296. This leaves room for a heading and a separate Review Button.
3. Open Type Settings. Set Item Source to manual.
4. Leave Allow Multiple Selection and Dropdown View off for now.
5. Create a String variable called selected_clue with an empty starting value. Enter selected_clue in Selected Item Variable. Enter the ID alone, without a wildcard prefix.

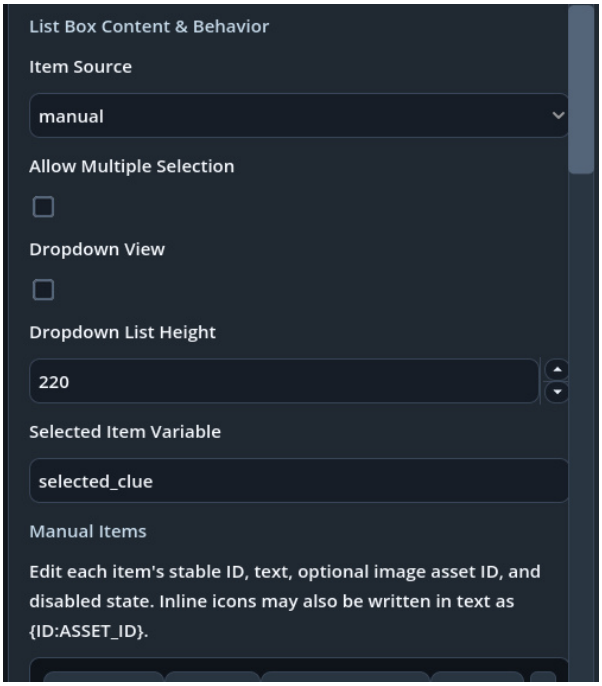

Figure 16. The first list settings: source, selection mode, dropdown behavior and the output variable.

The source chooses where rows come from. Selected Item Variable chooses where the player’s choice is written. Those are two different jobs.

Enter and edit the manual rows

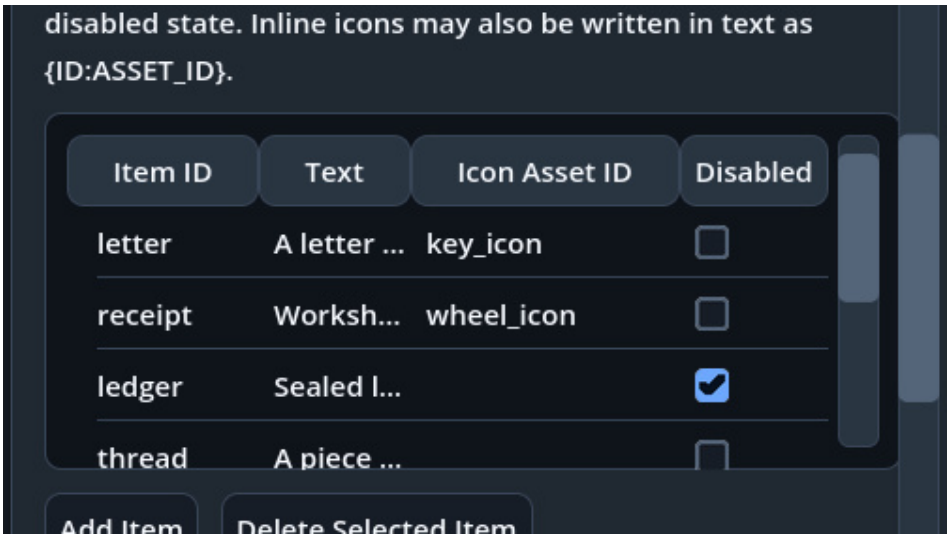

Figure 17. The Manual Items table. Text can be clipped in the editor cell even though the complete value is stored.

Item ID	Text	Icon Asset ID	Disabled
letter	A letter from the keeper	key_icon	Off
receipt	Workshop receipt	wheel_icon	Off
ledger	Sealed ledger	(empty)	On

1. Use Add Item to insert a row, then edit its Item ID and Text cells.
2. Enter an image asset’s ID in Icon Asset ID if the row needs a leading icon. This is a raw ID field, not a descriptive name.
3. Tick Disabled for the sealed ledger. It stays visible, but the player cannot select it.
4. Add a few more rows if you want to test scrolling. Use Delete Selected Item to remove the highlighted authoring row.

Keep row IDs stable. You can rename “Workshop receipt” to “Receipt from the repair shop” without changing receipt. That lets your scripts keep finding the same clue after wording or translations change.

> Duplicate IDs can be made unique by appending a suffix. Do not depend on this for story logic: assign deliberate IDs yourself, and check them after duplicating rows.

12. Understand what a selection returns

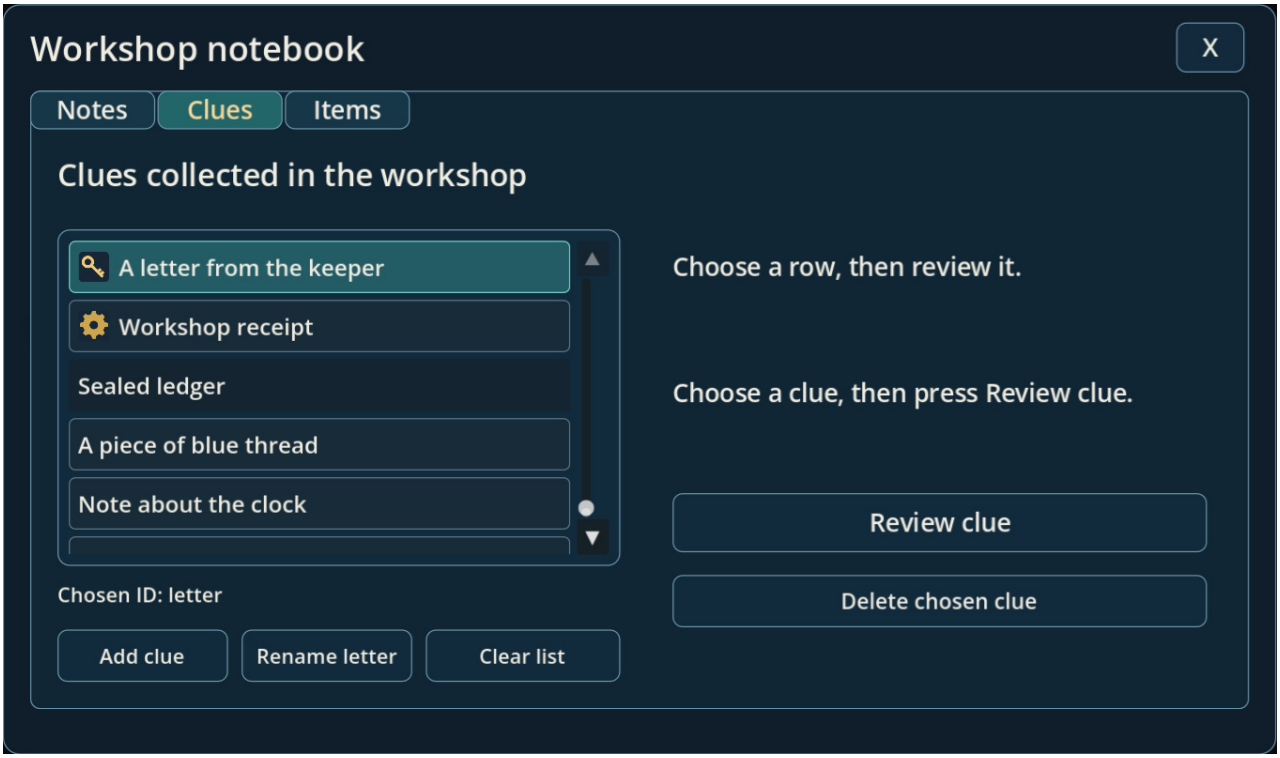

Figure 18. The selected row is highlighted. The Label below the list displays the returned ID: letter.

When the player clicks the keeper’s letter, selected_clue becomes the String letter. The visible sentence is not the returned value. Your response logic should compare IDs, not the words on screen.

To see this while testing, add a separate Label with Text: Chosen ID: $$selected_clue. A normal Label expands that variable wildcard during gameplay. The list’s row text has different rules, covered in Chapter 15.

Player action	Result in this single-select example
Clicks letter	selected_clue becomes "letter".
Clicks receipt	It becomes "receipt"; letter is no longer selected.
Clicks the disabled ledger	The selection does not change.
Clicks the selected row again	The selection clears to an empty String.

Always handle “nothing selected”

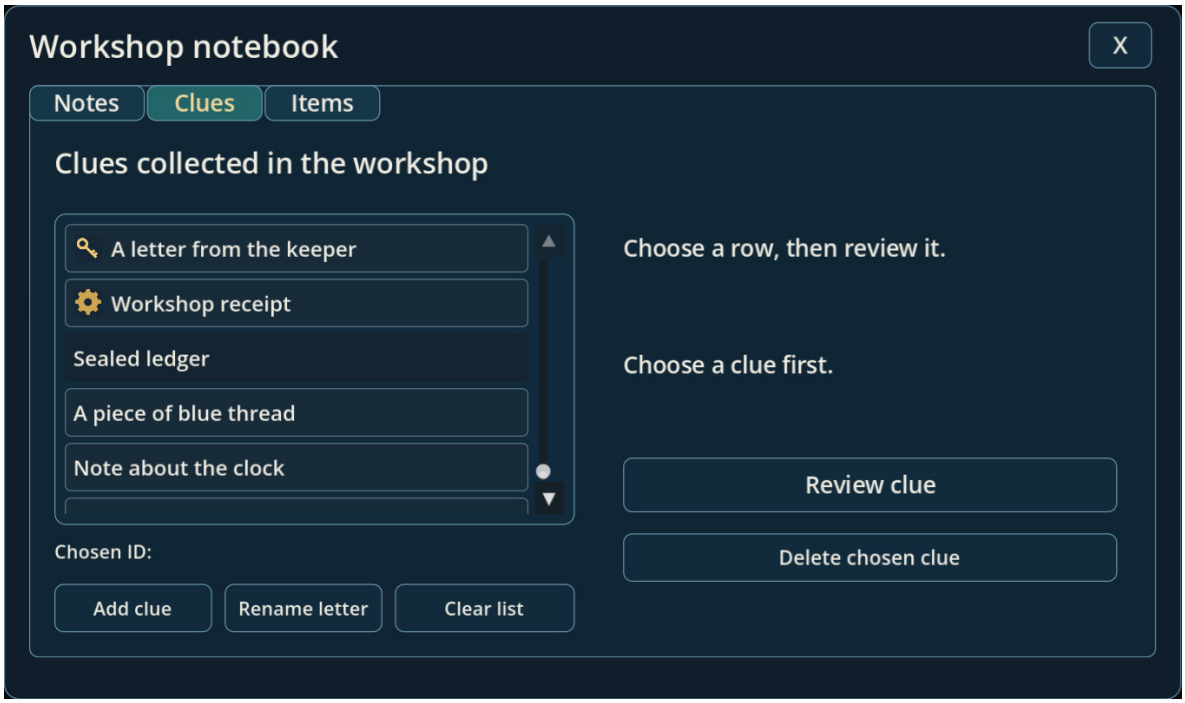

Figure 19. The chosen ID is empty, and Review clue asks the player to choose a clue first.

A single-select list can have no selection. This happens at startup, when the selected row is clicked again, or when its source row is removed. The Review Button must cope with that.

1. Start with selected_clue empty, rather than pre-filling it with letter.
2. Select the letter, click it again and check that its highlight disappears.
3. Press Review clue with no selection. Show a short instruction rather than trying to look up an empty ID.
4. Select the letter again, then review it. Check that the correct response appears.

Treat Selected Item Variable as an output from the widget. Setting selected_clue to letter with Set Variable does not, by itself, tell the list to highlight that row. A variable and the widget’s internal selection are separate pieces of state.

> For a manual list, List Box: Modify Item can change the selected property deliberately. For a live Array or inventory list, read Chapter 21 before doing that: it switches the list to a local copy.

13. Make the Review Button work

Create a GDScript record named Review Clue with ID review_clue. Keep the script base line, replace the run function with this version, then Apply Changes. Create a String variable clue_feedback for the response text.

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

func run(_context: Dictionary = {}) -> Variant:
    var chosen: String = str(
        Runtime.variables.get("selected_clue", "")
    )
    var response: String = "Choose a clue first."
    match chosen:
        "letter":
            response = "The keeper expected a visitor."
        "receipt":
            response = "The receipt lists a clock spring."
        "":
            pass
        _:
            response = "You examine the clue: " + chosen
    await SetVariable({
        "name": "clue_feedback", "value": response
    })
    return true
```

1. Add a Button on Clues Page with the caption Review clue.
2. In Events, set Click to run-visual-script and enter review_clue in Click Target. Despite its name, this behavior can run the GDScript record used here.
3. Add a separate Label whose Text is $$clue_feedback. Give it enough width and height for the response.
4. Run the game, select letter and press Review clue. Then repeat with receipt and with nothing selected.

Selecting a row does not automatically run this script. The separate button makes the order clear: choose first, confirm second.

14. Give rows room to breathe



Figure 20. Outer Padding and Item Padding affect different spaces. Item Row Height determines the room available to one row.

Field	Example	Effect
Outer Padding	10	Space between the list frame and its rows.
Item Padding	8	Inset inside an individual row.
Item Row Height	46	The row height used by the list layout.
Item Gap	6	Space between consecutive rows.
Item Icon Size	28	Size of a leading or inline icon.
Icon / Text Gap	8	Separation between an icon and nearby text.

Use Text & Content to set the list font and text color. The example uses size 20. Increase row height if larger text or icons feel cramped. Keep touch targets comfortably separated.

Rows are single-line entries, with clipping rather than a paragraph layout. Put a long clue description in a separate Label beside the list, as the Review example does.

Choose frame and row styles separately

There are two sets of surfaces: the frame around the whole list and the individual row backgrounds. Keep them distinct while you tune the layout.

Field	Meaning
Border Type	none, solid, double, dashed, dotted, inset or outset. Start with solid while learning.
Border Width / Border Radius	The outer list frame.
Item Background	Ordinary enabled row surface.
Item Hover Background	Row surface under the pointer.
Item Selected Background	Surface of a selected row.
Item Border Color / Selected Border Color	Ordinary and selected row outlines.
Item Border Width / Item Border Radius	Outline thickness and rounding for rows.
List Prefix	none, bulleted or numbered. A display prefix is not part of the row ID.

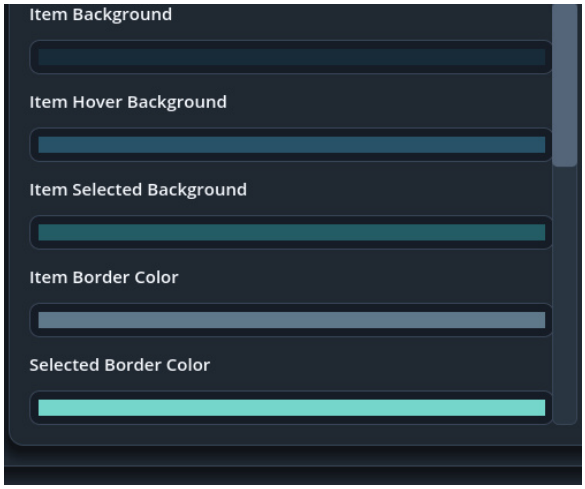

Figure 21. Ordinary, hover and selected row colors belong to the rows, not to the outer frame.

For clean outer-versus-row radius control, leave the four shared per-corner overrides at -1 and use Border Radius and Item Border Radius. Explicit shared corner overrides can also affect row styles in this build. Patterned borders need a separate visual check.

15. Icons, text and wildcard limits

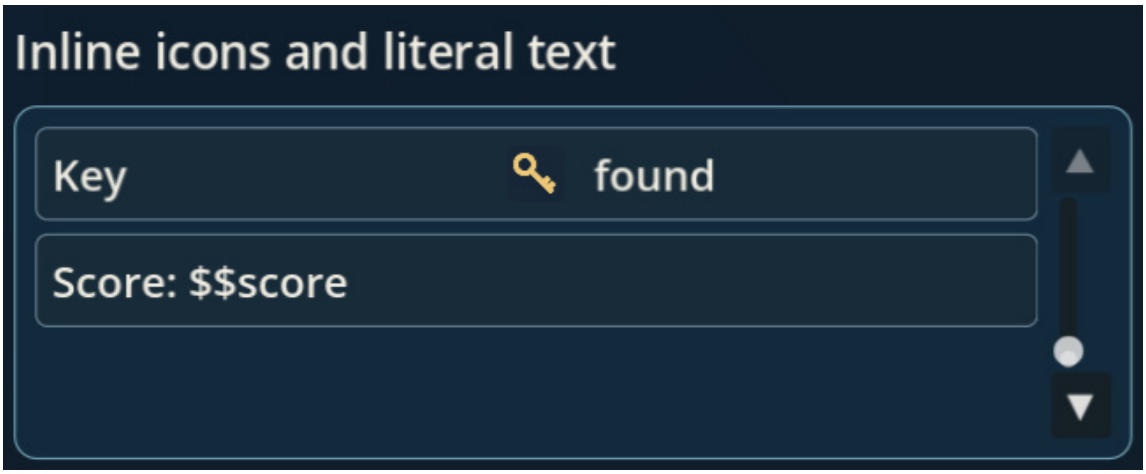

Figure 22. The lower-right list shows an inline key icon and literal $$score text. These are actual list rows.

A row’s Icon Asset ID supplies a leading image. You can also put an image marker inside the row’s Text, using the exact asset ID:

```
Key {ID:key_icon} found
```

The marker becomes the imported image when that asset can be found. Several markers can be used in one row. A missing or mistyped asset ID can leave the marker visible as text. Keep icons small enough to fit the row.

Text entered in a list row	Result in this build
A letter from the keeper	Plain row text.
Key {ID:key_icon} found	Text with an inline icon when key_icon exists.
Score: $$score	Literal $$score text; it is not expanded as a variable here.
@PlayerName or formatting tags	Do not rely on Label-style expansion or rich-text formatting in rows.

Build dynamic row text in the Array or script that supplies the rows. For translated row sentences, supply the appropriate localized strings through that data workflow. Do not assume a List Box runs the same text-expansion or translation path as a normal Label.

> Text & Content’s list Text field is the dropdown placeholder, not the manual item table. To change a row sentence, edit that row’s Text cell or its source data.

16. Scrolling and navigation

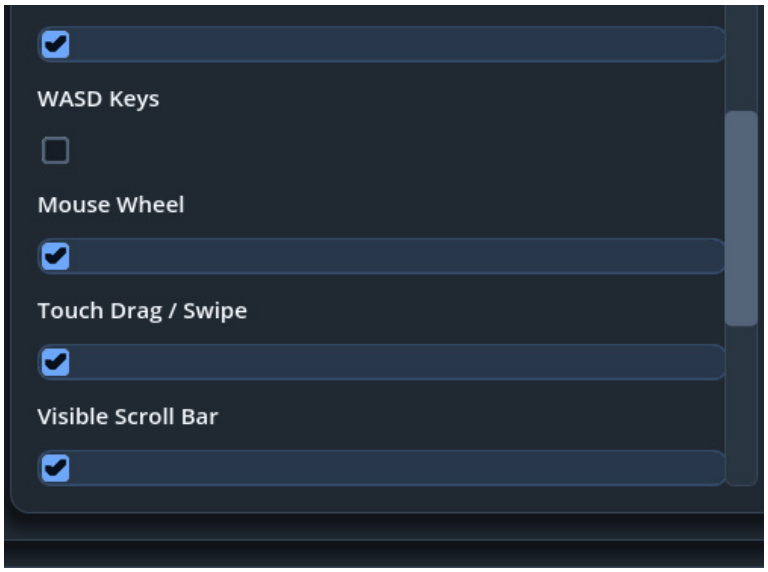

Figure 23. Independent switches let you choose the input methods supported by this list.

Option	What it enables
On-screen Arrow Buttons	Previous / next controls for scrolling through overflowing rows.
Keyboard Arrow Keys	Keyboard scrolling while the list has the relevant focus.
WASD Keys	Another keyboard scrolling option; consider conflicts with direct character movement.
Mouse Wheel	Wheel scrolling over the list.
Touch Drag / Swipe	Touch or drag gestures for scrolling.
Visible Scroll Bar	The synchronized scrollbar control when there is overflow.

These controls scroll the view. They are not a promise that pressing Down selects the next row, confirms a choice or triggers an adventure action. Test keyboard focus and gamepad behavior in the interface you actually intend to ship.

A row’s selection refresh can rebuild row buttons, so do not base a keyboard-only workflow on an untested focus assumption.

Place and skin the navigation controls

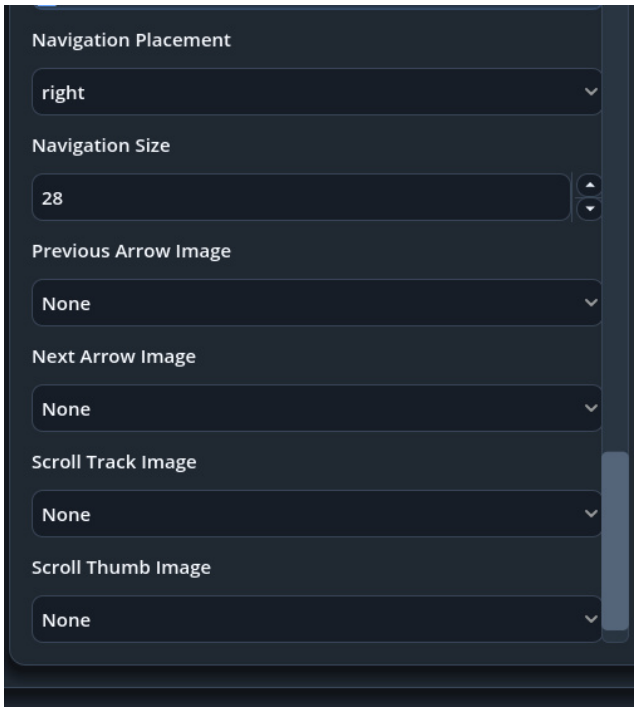

Figure 24. Navigation placement, size and optional image assets are separate from row icons.

Field	Use
Navigation Placement	left, right, top or bottom. This moves the navigation strip; the rows remain vertical.
Navigation Size	Thickness of the strip. The example uses 28.
Previous / Next Arrow Image	Replace the standard direction controls with your own imported images.
Scroll Track / Thumb Image	Artwork for the slider track and the part dragged along it.

Use a long enough list to test the controls. With no overflow, there is nothing to scroll. At an end of the range, the corresponding arrow is disabled. Wheel and arrow steps are based on the row height and gap.

For a small touch screen, increase Navigation Size and leave space between the scrollbar and story buttons. Check both a nearly empty list and a long one: a layout that works only with the sample’s seven clues is not finished.

17. Allow more than one selection

Enable Allow Multiple Selection when a player can choose several rows, such as clues to compare. Use an Array variable for the output, starting with an empty Array.

1. Create an Array variable named selected_many with initial value [].
2. Add a List Box, enter selected_many in Selected Item Variable and enable Allow Multiple Selection.
3. Add the letter, receipt and disabled ledger rows. Start a running preview.
4. Click letter and receipt once each. Both should be highlighted. Click letter again to remove just that choice.

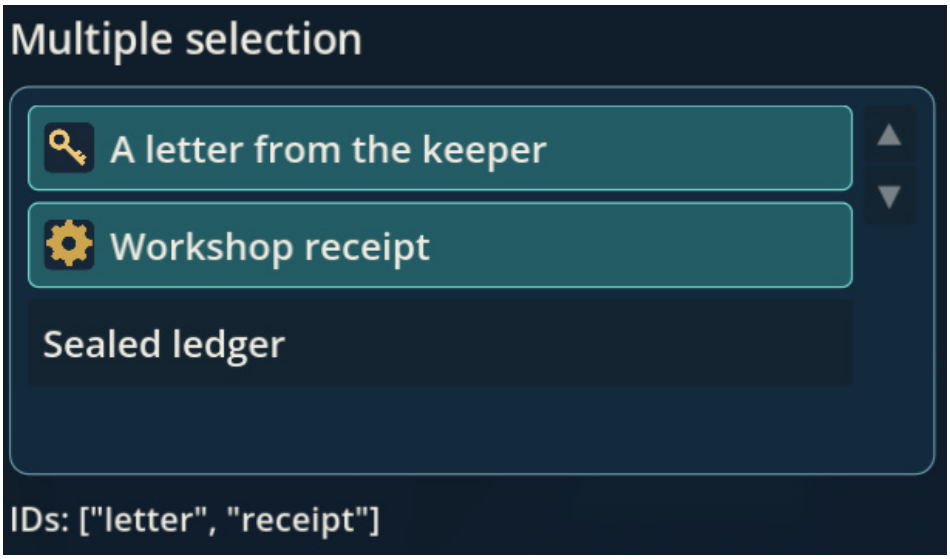

Figure 25. Two rows selected together. The output contains both stable IDs.

Selection mode	Output
Single selection	"letter" or an empty String.
Multiple selection	["letter", "receipt"] or an empty Array.

Ordinary row clicks toggle choices in multiple-select mode; Ctrl is not required. The selected-ID Array follows selection order. If your logic cares only about membership, check whether it contains an ID rather than depending on an index.

18. Use the list as a dropdown

Dropdown View gives the same list a collapsed header. This suits a destination selector, category filter or compact choice control. The data source and selection rules remain the same.

1. Add a List Box with ID route_list. Make it 525 pixels wide and 52 high.
2. Enable Dropdown View. Set Dropdown List Height to 190.
3. Use a String variable selected_route as Selected Item Variable.
4. In Text & Content, enter Choose a destination... in the dropdown placeholder text field.
5. Place the element high enough that its expanded list fits below it.

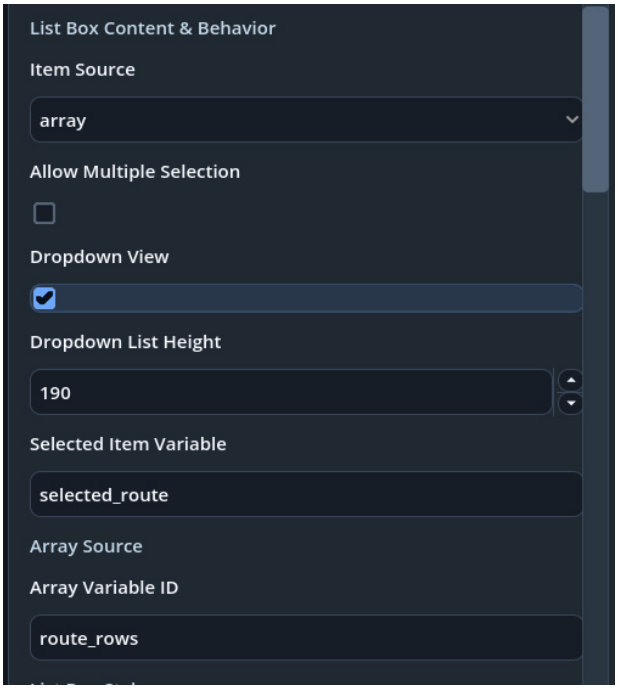

Figure 26. Dropdown View and Dropdown List Height. The source here is the route_rows Array described in Chapter 19.

The element’s own Height controls the collapsed button. Dropdown List Height controls the list below it when open. Increasing one does not replace the other.

Test the expanded dropdown

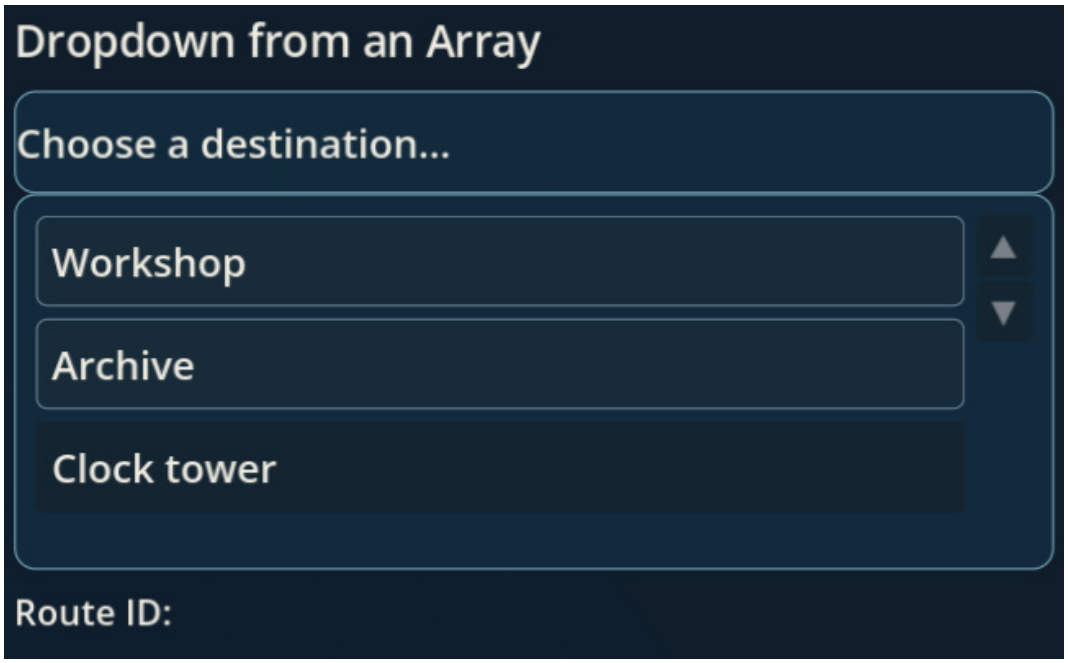

Figure 27. The route list opens downward. The clock tower row is disabled in its source data.

Click the collapsed header to open it. Choose Archive and the single-select list closes with Archive as its caption; selected_route becomes archive. In multiple-select mode, choosing a row leaves the list open so more choices can be made.

The list opens downward rather than automatically flipping above its header. It can still be clipped by a parent GUI or a tab content rectangle, even though it draws above nearby controls. Move it up or reduce Dropdown List Height if the bottom is cut off.

Keep dropdown row text simple. Inline icon markers are rendered in rows, but the collapsed caption is assembled from the text values; marker text can be visible there. A separate leading row icon is easier to manage.

> Do not rely on a click elsewhere closing every dropdown in this build. Check the header toggle and selection behavior directly. Test the longest caption and the smallest intended display.

19. Fill a list from an Array

Use an Array source when game data should decide which rows exist. For example, a clue list can grow as the player learns things, while the GUI keeps the same layout.

1. Create an Array variable named clue_rows. Give it the object records below as its initial value.
2. Select the List Box. Set Item Source to array and Array Variable ID to clue_rows.
3. Use a different variable, selected_array, for Selected Item Variable. Make that a String for a single-select list.
4. Start a game preview and check that the rows appear.

```
[
  {"id": "bell", "text": "The bell rang at midnight"},
  {"id": "window", "text": "The upper window was open"},
  {"id": "oil", "text": "Oil on the workshop floor"}
]
```

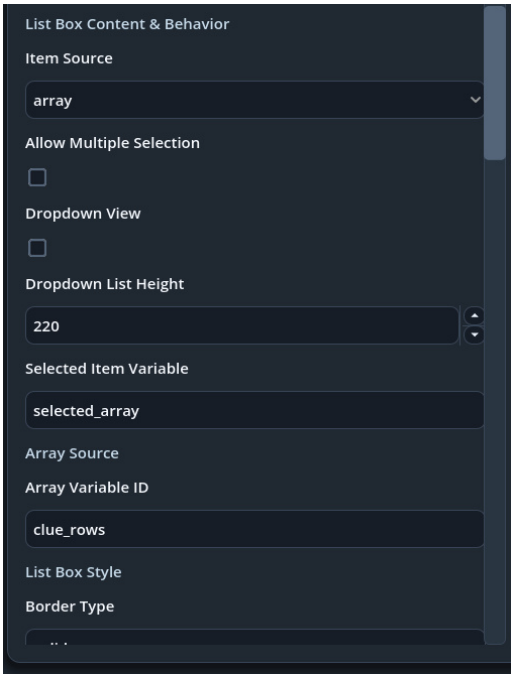

Figure 28. Array Variable ID points to the source Array. Selected Item Variable is still a separate output.

Use a real Array value. A String containing quoted JSON is still a String; it is not the same data type.

Give Array rows stable identities

Record field	Purpose
id	Stable row ID returned by selection. Give every meaningful row one.
text	Visible row sentence.
iconAssetId	Optional leading image asset ID.
disabled	true prevents selection; false permits it.

An Array of simple Strings also works, but the generated IDs follow positions: item_1, item_2 and so on. Insert or reorder a row and the meaning of a position-based ID can change. Object records avoid that problem.

```
[
  {
    "id": "archive",
    "text": "Archive",
    "iconAssetId": "key_icon",
    "disabled": false
  },
  {
    "id": "tower",
    "text": "Clock tower",
    "disabled": true
  }
]
```

The route dropdown uses this same pattern. Its source is route_rows and its selection output is selected_route. Selecting the Archive sentence returns archive, which you can use in a script that decides where to send the player.

The designer uses authored initial values to preview an Array list. It does not show the current values from a game that happened to be running earlier.

Update the data during play

Change the source variable through Set Variable or a script action so the GUI receives a state update. The following is a complete small GDScript example:

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

func run(_context: Dictionary = {}) -> Variant:
    var rows: Array = [
        {"id": "bell", "text": "The bell rang twice"},
        {"id": "floor", "text": "Fresh marks on the floor"}
    ]
    await SetVariable({
        "name": "clue_rows", "value": rows
    })
    return true
```

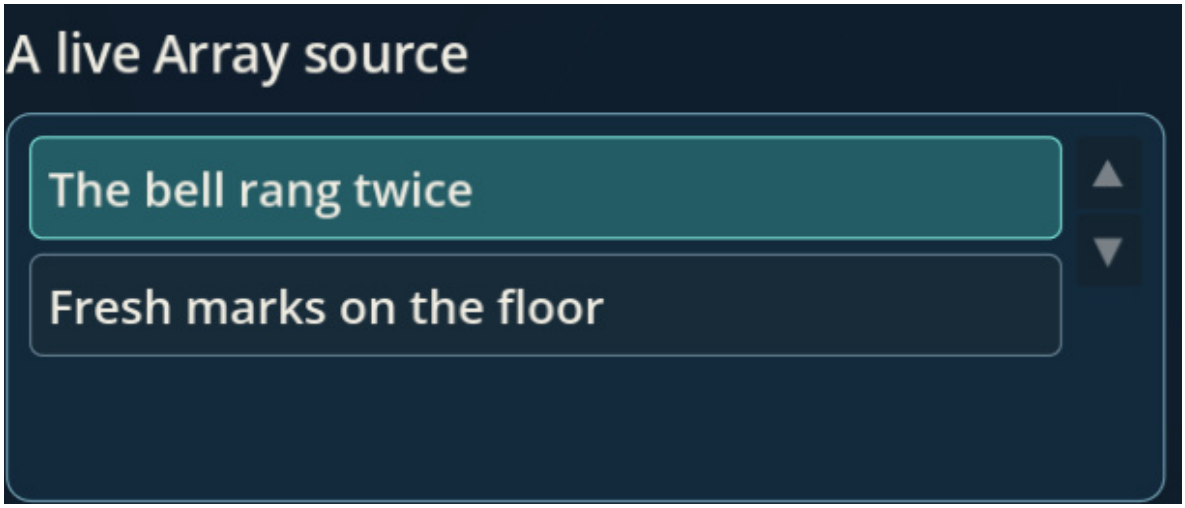

Figure 29. The live Array list now shows the changed bell sentence and the new floor row.

A selected row can remain selected when its ID is still present. In the test, bell stayed selected after its text changed. Removing bell from the source cleared that list’s selected ID.

For a growing journal, read the current Array, make a copy, append a new object with a stable ID, then pass the whole updated Array to Set Variable. Avoid directly changing a nested runtime value and assuming the GUI was notified.

> Leave list-item actions out of a source-driven list unless you deliberately want a local copy. The next chapters explain why that distinction matters.

20. Display the carried inventory

1. Add a List Box to Items Page, with ID carried_list.
2. Set Item Source to inventory.
3. Choose Current Active Player for Inventory Owner. For an NPC’s items, choose the specific-character mode and select that character definition.
4. Create a String variable selected_item, starting empty. Use it as Selected Item Variable.
5. Make sure the player actually owns at least one defined inventory item, then test the notebook in gameplay.

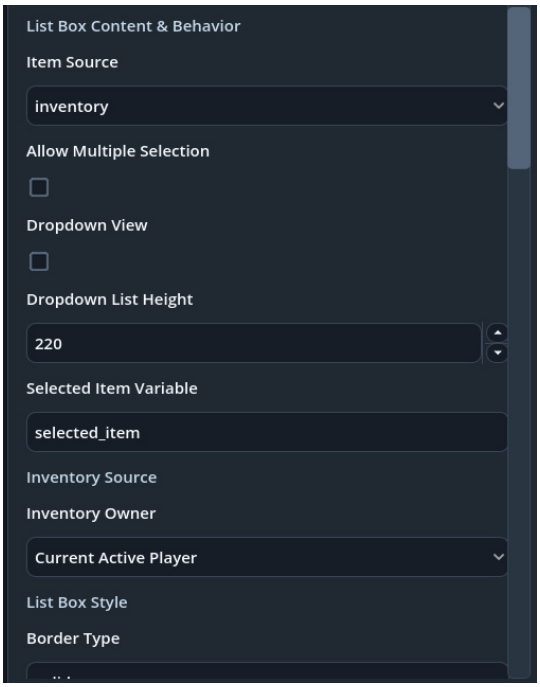

Figure 30. The inventory source uses the chosen owner’s carried items, rather than the Manual Items table.

Rows take their names and images from inventory definitions. Positive carried quantities appear in the list. Stack quantities greater than one are added to the display name, such as Bearing oil ×2.

> The designer can show inventory definitions as sample rows. That does not mean a new player carries every item. Compare the preview with actual starting inventory and the items awarded during play.

Selecting an item is not using it

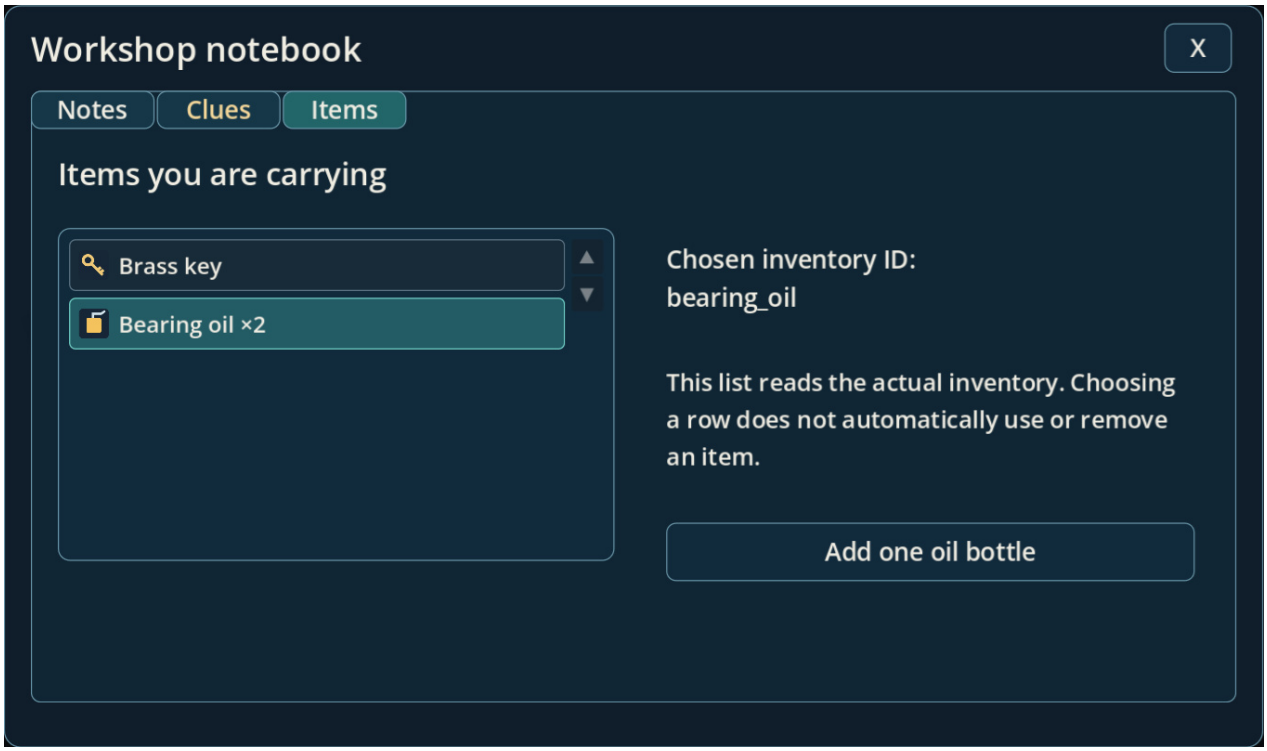

Figure 31. The player selected bearing_oil, then received another bottle. The live inventory list updated to ×2.

Selecting Bearing oil writes bearing_oil to selected_item. It does not automatically equip it, switch the adventure cursor, combine it with another item or remove it from inventory. Add a separate Use, Inspect or Give Button with the game logic you want.

Goal	Change the correct data
Give an item	Use Add Inventory with the inventory item ID and Count.
Remove an item	Use the inventory removal action.
Change a stack quantity	Use the inventory quantity action.
Only remove a row from a local list	List Box: Delete Item changes that widget’s view; it does not remove the carried item.

The example’s Add one oil bottle Button uses Add Inventory, item bearing_oil, Count 1. Because the item definition is stackable, the list shows the new total. Test a nonstackable item separately rather than assuming it behaves like a stack.

A specific-character list refers to the character definition ID. Make sure you chose the intended owner when a party or NPC inventory looks empty.

21. Change manual rows with actions

List actions are available through GUI events, action lists and visual scripting. They use the same GUI, element and row IDs in each place. For a first test, give the Clues page an Add clue Button.

1. Select Add clue and open Events. Choose the Click behavior List Box: Add Item.
2. Choose Clues Page as GUI and Clue List as List Box.
3. Use New Item ID clock_hand and Item Text A loose clock hand.
4. Choose wheel_icon as Optional Icon, leave Disabled false and use Insert Index -1 to append at the end.

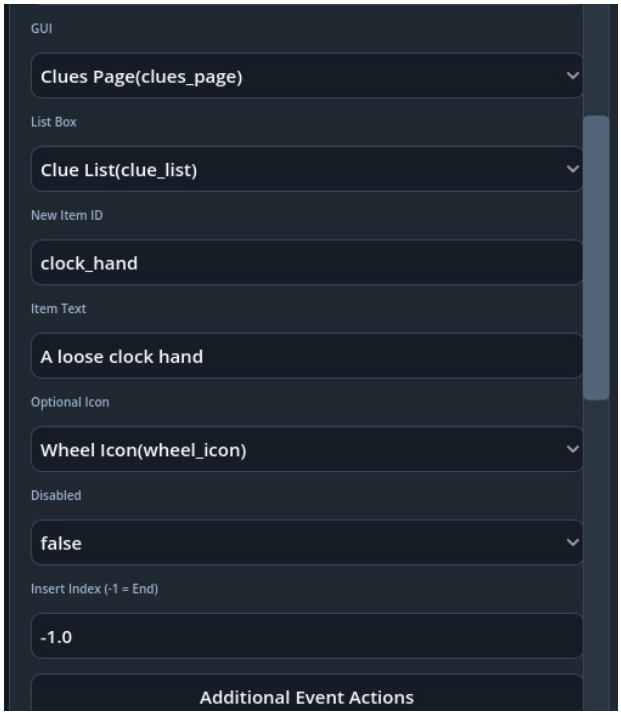

Figure 32. The single-action editor for Add Item. Dropdowns identify the target GUI and List Box.

Run the game and press Add clue. Scroll to the end if needed to find the new row. A second press can create a suffixed ID such as clock_hand_2. For a one-time discovery, guard the award with a story variable instead of allowing unlimited additions.

> 0.86.3.4 has a validation wrinkle with literal list-row IDs in these action records. Read Chapter 24 before using this route in a distributable project.

Add, modify, delete and clear

Action	What to supply	Result
List Box: Add Item	GUI, List Box, New Item ID, Text, Optional Icon, Disabled, Insert Index	Adds a row; -1 appends. Duplicate IDs can receive suffixes.
List Box: Modify Item	GUI, List Box, Item ID, Property, New Value	Changes a row field.
List Box: Delete Item	GUI, List Box, Item ID	Removes one row from the widget.
List Box: Clear Items	GUI and List Box	Empties that widget’s local rows.

Modify Item supports text, id, iconAssetId, disabled and selected. Use Boolean true or false for disabled and selected. For a rename that should preserve story references, change text rather than id.

```
await ListBoxModifyItem({
    "guiId": "clues_page",
    "elementId": "clue_list",
    "itemId": "letter",
    "property": "text",
    "value": "The keeper's letter"
})
```

The code above belongs inside the run function of a script based on script_base.gd. It changes only the displayed sentence for letter. The row ID remains letter, so Review Clue still recognizes it.

Changing selected through Modify Item can deliberately select or deselect a manual row and update its output variable. Do not use it casually on a live source list: even this property change activates the local-copy behavior described next.

The important source-copy rule

Add, Delete, Modify and Clear operate on the list widget. For a manual list that is straightforward. For an Array or inventory source, the first item action creates a local copy of the current rows and starts using it instead of the live source.

Before an item action	After an item action
Array list reads clue_rows.	It displays its local rows; later clue_rows changes no longer replace them.
Inventory list reads carried items.	It displays a local copy; item actions do not change actual carried quantities.
Source updates drive the view.	You must manage the copied view deliberately.

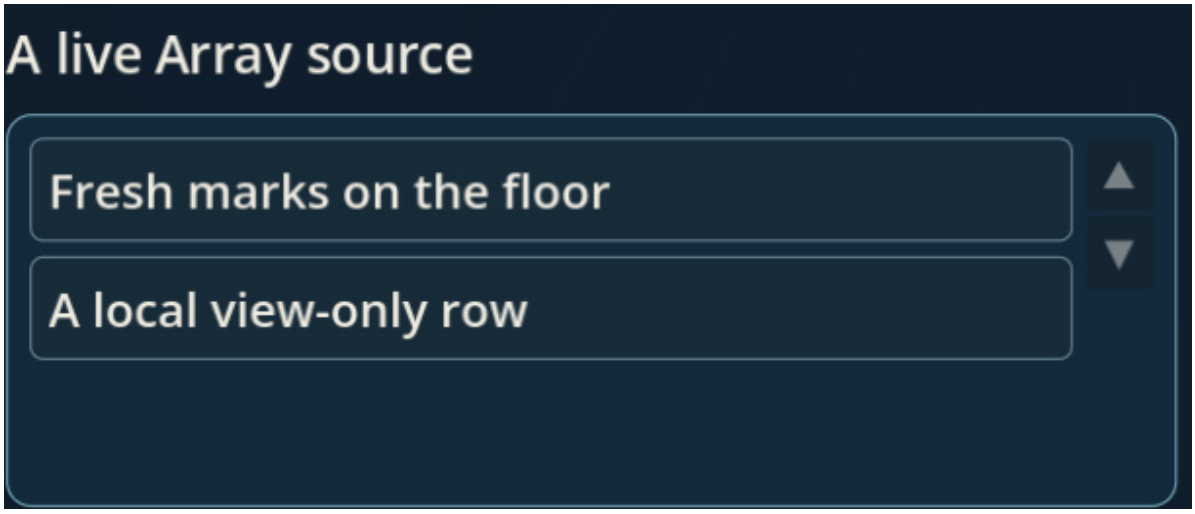

Figure 33. A source-copy test: the local extra row remains after the source Array was replaced.

Hiding and showing the GUI does not reset this copy. Changing the source setting on the existing widget is not a dependable reset either. A fresh game run creates fresh widgets from authored data.

For a live journal, change the Array with Set Variable. For carried items, change the actual inventory. Use list-item actions for manual lists or a deliberately independent view.

> This rule also applies to Modify Item → selected. It is not limited to adding or deleting rows.

22. Delete the row the player chose

A fixed Item ID field is fine for a known row. To delete whichever row the player selected, read selected_clue in a script and pass that value as itemId. Do not assume typing $$selected_clue into an ID field will expand it.

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

func run(_context: Dictionary = {}) -> Variant:
    var chosen: String = str(
        Runtime.variables.get("selected_clue", "")
    )
    if chosen.is_empty():
        await SetVariable({
            "name": "clue_feedback",
            "value": "Choose a clue first."
        })
        return false
    await ListBoxDeleteItem({
        "guiId": "clues_page",
        "elementId": "clue_list",
        "itemId": chosen
    })
    await SetVariable({
        "name": "clue_feedback",
        "value": "Removed from the list: " + chosen
    })
    return true
```

1. Create a GDScript record with this source and Apply Changes.
2. Add a Delete chosen clue Button and set its Click event to run that script.
3. Choose letter and press the button. The row disappears and the selected ID clears because that row no longer exists.

This example is for the manual clue list. It is not an inventory removal script and does not delete an inventory definition.

23. Use the same actions as visual nodes



Figure 34. The visual script uses the same List Box: Add Item operation as the Button’s action editor.

1. Create a visual script named Add Clue. Add the entry node for the interaction you intend to run.
2. Add a List Box: Add Item action node. Choose the same GUI and List Box targets used in Chapter 21.
3. Fill in clock_hand, A loose clock hand, the optional icon and Insert Index -1.
4. Connect the entry output to the action node. Apply or save the graph, then assign the visual script to the desired GUI Button or gameplay interaction.
5. Test once, then repeat the interaction to decide whether a one-time guard is required.

An action list is another place to arrange the same operation. Add Item can be followed by Set Variable or another story action. The fields do not change meaning just because the action is in a different editor.

For branching on a selected ID, compare the selection variable with the stable row ID. Use the available variable condition controls, or the complete GDScript example in Chapter 13 when you want the response logic written in one place.

> Do not treat the List Box element’s general Click or Hover event as a guaranteed per-row selection event. The select-then-confirm pattern gives your script a clear selected-ID input.

24. A validation issue in this release

During this guide’s tests, the literal row IDs letter and clock_hand in List Box action records were checked as though they had to be inventory item definitions. They do not: they are manual clue-row IDs. The game actions still worked in preview.

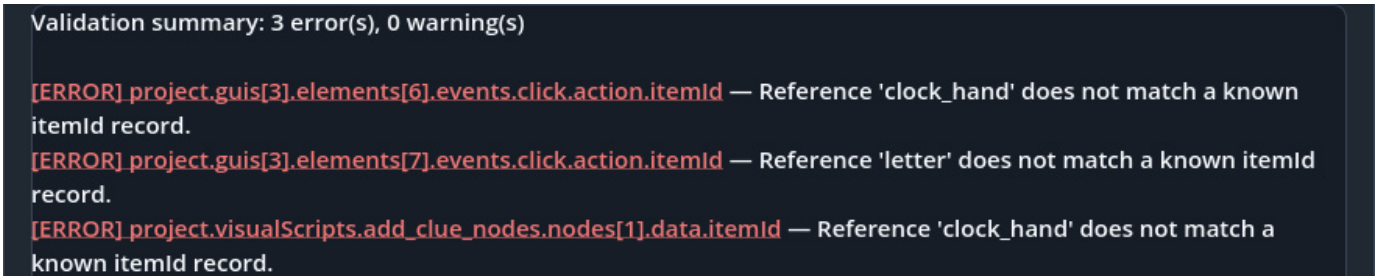

Figure 35. The validator reports itemId references from GUI and visual-script action records. These particular IDs belong to list rows.

For this specific situation, Preview with errors lets you test the unfinished project. It does not repair validation and does not remove the requirement to resolve build errors before distribution. Other errors in the window can be real, so read each one.

A practical workaround

Put the list operation inside a GDScript record and have the Button or action list run that script. The runtime method is the same, but the literal list-row ID is then inside script source rather than the misclassified action field. For example, use this body inside a run function:

```
await ListBoxAddItem({
    "guiId": "clues_page", "elementId": "clue_list",
    "itemId": "clock_hand",
    "text": "A loose clock hand",
    "iconAssetId": "wheel_icon",
    "disabled": false, "insertIndex": -1
})
```

Use the same approach for the rename example if its literal row ID is flagged. Revalidate after removing the replaced action record or node. Do not add fake inventory items just to make a clue-row warning disappear.

25. Open, close and restyle the notebook

Open the host GUI from a normal Button or gameplay action. If a small HUD contains the Open notebook Button, hide that HUD while the notebook is open so the controls do not overlap.

Button	Action list, in order
Open notebook	Show GUI → case_notebook; Hide GUI → notebook_hud.
Close notebook	Hide GUI → case_notebook; Show GUI → notebook_hud.

The equivalent GDScript calls, inside a run function, are:

```
await ShowGUI({"guiId": "case_notebook"})
await HideGUI({"guiId": "notebook_hud"})
```

Use Change GUI Element Property when story logic needs to alter an ordinary panel’s appearance. This example changes the background of the plain Summary Panel on Notes Page:

```
await ChangeGUIElementProperty({
    "guiId": "notes_page",
    "elementId": "summary_panel",
    "property": "background",
    "value": "#234b5b"
})
```

The same action is available in an action list or visual graph. Target the GUI that owns the element, then the element itself. Test the change in a running game; the GUI designer’s local Preview does not execute these story actions.

Use the ordinary Summary Panel for this example. Avoid rebuilding notebook_tabs while it owns linked page content.

26. Preview, saving and restoring state

Task	Use this
Move and resize controls	Designer editing, Preview off.
Try local list selection, scrolling or a dropdown	Designer Preview on.
Test a selected-ID variable or Review Button	Running game preview.
Check actual carried inventory	Running game with the intended player and item awards.
Keep GUI geometry and authored rows	Apply where required, then Save Project.
Keep a player’s changing journal	Store its source data in game state and test save/load.

Designer Preview can let you select rows locally, but it does not write into the real game’s variables or run the Review Button’s gameplay event. A response that stays unchanged there is not evidence that the script failed.

Runtime list actions do not rewrite the project’s authored Manual Items table. Saving the project in the editor is different from saving a player’s game.

For a journal that must survive save/load, prefer a source Array or story variables as the record of discoveries. Rebuild any deliberately local view from that saved data when appropriate. Do not rely on a widget’s temporary source copy, selected highlight or current tab being a complete save-game record.

Test a real save/load sequence: discover a clue, choose an item, close the notebook, save, change state, load, then reopen it. Check both the underlying data and what the player sees. A restored selected-ID variable alone need not restore the widget highlight.

> Start each basic test with empty selection variables. Add deliberate selection-restoration logic only after the list’s ordinary behavior is working.

27. Put the full notebook together

Use this order to build the example without trying to solve every setting at once. Finish and test each stage before adding the next.

1. Create the host case_notebook and the three page GUIs: notes_page, clues_page and items_page. Match their sizes to the host content area.
2. Add notebook_tabs to the host. Enable Tabbed Panel, enter the three links and set the initial local tab ID to notes.
3. Build Notes Page from a plain panel and separate Labels. Group the arrangement for editing.
4. Build Clues Page with a manual clue_list, selected_clue, clue_feedback and the Review script. Test selecting, deselecting and disabled rows.
5. Build Items Page with an inventory-source carried_list and selected_item. Award an actual item and verify it appears.
6. Add Open and Close action lists to show and hide the host. Do not show embedded pages independently.
7. Tune tab colors, list row spacing and navigation. Check your longest text and smallest intended window.
8. Add optional list actions or source-Array updates. Read the source-copy rule and resolve validation before making a release build.

A useful final test is deliberately uneventful: open the notebook, change tabs a few times, review a clue, close it and return to the room. The notebook should feel like one coherent interface, rather than several unrelated windows.

> The example’s Add clue, Rename letter, Delete chosen clue and Clear list Buttons are teaching controls. A finished adventure would usually trigger discoveries and removals through story interactions instead.

28. When a panel or tab looks wrong

Problem	Check this first
The panel’s text is missing.	A plain Panel does not draw a heading from its own Text field. Add a Label above it.
The panel covers my labels.	Set the panel’s Z Index below the labels. Check the labels are visible and enabled.
Moving the panel leaves its text behind.	The text is a sibling element. Select and Group the complete arrangement for editing.
The host shows only “Embedded GUI”.	That is the designer’s link hint. Edit the page GUI separately and test actual content in a running game.
A tab has no content at runtime.	Check the third entry part is an existing page GUI ID, and the page is not embedded by another host.
The wrong page opens first.	Editor Active Tab needs the local tab ID, such as notes, rather than notes_page.
One tab’s font stays gold when another is selected.	Named Active Tab style fields belong to that named tab. Use Selected Background for the current-page cue.
Controls are clipped below the tabs.	Account for tab-strip height; match page size to the usable area.
A copied notebook changes the original’s pages.	Clone and relink the page GUIs as well as the host.
Legacy controls never show.	Check Parent Panel ID and local Tab ID. A tab with a linked GUI does not also display its legacy children.

After fixing a structural page link, restart the game preview to test fresh widgets. A previously running host can retain state from earlier interactions.

When a list does not behave as expected

Problem	Check this first
The list is empty in gameplay.	Check Item Source. Array needs a real Array; inventory needs actual carried items for the chosen owner.
Rows appear in the designer but not in the game.	Inventory preview can show definitions. Array preview uses authored initial data. Neither proves the live source is populated.
Selection variable never changes.	Use a running game, enter the variable ID alone, and choose an enabled row.
Clicking a row does not run my response.	Connect a separate confirmation Button that reads the selected-ID variable.
The selection unexpectedly becomes empty.	Clicking an already selected row toggles it off. Removing a selected source row also prunes the selection.
Setting the variable does not highlight a row.	The variable is selection output. It is not a command to select that widget row.
New source data stopped appearing.	A list-item action may have activated a local copy. Change the actual data for live-source lists.
Deleting an inventory row did not remove the item.	Use an inventory action to change carried items. List actions change the widget.
The list shows $$score literally.	List row text does not use normal Label wildcard expansion. Build the sentence in its source data.
The dropdown is cut off.	It opens downward and can be clipped by its GUI or tab parent. Move it up or reduce the expanded height.
The scrollbar is absent or does nothing.	Check for overflow, supported input switches and the list’s focus. Navigation scrolls; it does not automatically select rows.

29. Keep these references handy

What you are referring to	Example
Host GUI	case_notebook
Tabbed panel element	notebook_tabs
Local tab ID	clues
Linked page GUI	clues_page
List element	clue_list
Row ID	letter
Single-select output variable	selected_clue → "letter" or ""
Multiple-select output variable	selected_many → ["letter", "receipt"] or []
Array source variable	clue_rows → Array of row records
Inventory selection output	selected_item → a carried inventory item ID

The two useful text patterns

```
Tab entry:
clues|Clues|clues_page

Inline row icon:
Key {ID:key_icon} found

Normal Label showing a selected ID:
Chosen ID: $$selected_clue
```

Keep IDs stable, let the source own its data and use a separate confirmation action when a choice should affect the story. Those three habits make a tabbed notebook much easier to maintain as your adventure grows.

Screenshots and runtime checks: PCA 3D Adventure Studio 0.86.3.4. The key and gear images shown are existing project assets. No practice files are included.