ScarOptions

Complete Options Menu - Blueprint Guide

One widget in. A finished settings screen out.

ScarOptions is the options menu you were going to spend a fortnight building. Five pages - Graphics, Display, Audio, Gameplay and Controls - build themselves in C++, save to the same file the engine already uses, and re-apply themselves on every launch. There is no save game to write, no ini to edit, and nothing to lay out unless you want to.

What makes it quick to live with

- **Zero setup, genuinely.** Reparent a Widget Blueprint to `ScarOptionsMenuWidget` and add it to your menu. Or skip even that and call `Open Scar Options Menu` with an empty class - the built-in layout is a full-screen, blurred, tabbed panel that works with no art at all.
- **It does not fight your project.** ScarOptions never replaces UGameUserSettings, so a project that already subclasses it - and there can only ever be one - still works. No DefaultEngine.ini edit anywhere.
- **Every row can be undone.** A small amber arrow appears beside any setting that differs from what it was when the panel opened, and puts that one value back.
- **No Apply button, on purpose.** Changes write to disk a moment after the last edit, and closing the panel flushes anything still pending.
- **Edit the panel, do not rebuild it.** One button generates a Widget Blueprint of the built-in layout into your project, already named and wired, so you can restyle it and keep the rest. Or hand-place any control you like - they are matched by name, and what you do not place is still built for you.
- **Rebinding for both input systems.** Enhanced Input and the legacy Action/Axis mappings are both detected and listed together, with one row per axis direction so rebinding "forward" cannot silently cost the player "back".
- **Good defaults before anyone opens it.** On the first launch on a machine, ScarOptions benchmarks the hardware and picks a scalability preset that fits.

Plugin version *1.0* | ***Engine versions*** *UE 5.2 - 5.8* | ***Blueprint-first, C++ compatible***

Contents

1. Two-minute setup

There is no configuration step. Enable the plugin and pick whichever of these two routes suits your menu.

Route A - your own Widget Blueprint

1. Right-click in the Content Browser, **User Interface** → **Widget Blueprint**. Name it **WBP_Options**.
2. Open it, **File** → **Reparent Blueprint**, choose **ScarOptionsMenuWidget**.
3. Add it to your menu however you add any widget - **Create Widget** then **Add to Viewport**, or drop it inside an existing panel.

Leave the Designer canvas completely empty. C++ builds the whole panel on construct. You only add widgets in the Designer when you want to place things yourself - see section 9.

Route B - one node, no Blueprint at all

Drop `Open Scar Options Menu` into any graph and leave **Menu Class** empty. It creates the built-in widget and adds it to the viewport.

```
Open Scar Options Menu
    Menu Class  = (empty -> built-in layout)
    Z Order     = 100
    Return      = the menu widget, if you want to keep a reference
```

Closing the panel
The X and any Back button fire **On Close Clicked** - a dispatcher, so bind it from whatever opened the menu: a pause screen that has to come back, a controller that wants gameplay input again - and **BP On Back Clicked** for a Blueprint subclass. Then the menu removes itself and hands input back. Switch **Remove On Close** off to close it your own way instead: swap to a home screen, play a fade. Settings are already saved by the time either fires, so there is nothing to confirm.

That is the entire setup. Everything below is optional refinement.

2. What the player sees

Five tabs across the top, a scrolling list of rows under them. Every row is a label on the left, a control on the right, and an undo arrow that appears only once that row has been touched.

Graphics

Overall Quality is the preset - Low, Medium, High, Epic, Cinematic. Changing it moves every per-feature row below it, exactly as the engine's own scalability system does. The eight per-feature rows are the knobs the engine exposes independently:

View Distance • Anti-Aliasing • Post Processing • Shadows • Textures • Effects • Foliage • Shading

Motion Blur closes the page. It drives **r.MotionBlurQuality** and, unlike the rows above it, the engine does not persist it on its own - ScarOptions does.

Display

Row	What it does
Resolution	Built from the platform's real supported list. Falls back to the four common sizes if the platform reports none.
Window Mode	Fullscreen, Borderless Fullscreen, Windowed.
VSync	Straight to `Set VSync Enabled`.
Frame Rate Limit	30 / 60 / 120 / 144 / 240 / Uncapped by default. The list is editable in Project Settings; 0 always renders as Uncapped.
Resolution Scale	Rides the engine's normalized 0-1 scale and shows the real percentage, read back after the engine clamps it to what the platform allows.
Brightness / Gamma	Mirrors DisplayGamma, but survives a restart - the engine does not save gamma by itself.

Audio

Master, Music, Sound Effects and Voice Chat, all 0-100%.

The Master slider works before you configure anything
It drives the audio device's primary volume, which scales every sound in the game. The other three need a Sound Class each - point them at your own assets in Project Settings and all four sliders drive real buses. Until then Master alone is still a working volume control.

Gameplay

Mouse Sensitivity, which applies itself to both input systems, and Camera Shake, which your game applies with one node - see section 8.

Controls

One rebind row per action, one per axis direction, and a **Reset All Bindings** row at the bottom. Covered properly in section 5.

3. Where settings are stored

Everything lands in one file:

```
Saved\Config\<Platform>\GameUserSettings.ini
```

That is the same file the engine already writes. Settings the engine owns - resolution, window mode, VSync, frame rate, every scalability level - stay in **UGameUserSettings** where they have always lived. Settings it does not own - the four volumes, sensitivity, camera shake, the two toggles, gamma and motion blur - live in **UScarOptionsUserSettings**, which writes to the same file.

Why not subclass UGameUserSettings

The usual way to add settings is to subclass UGameUserSettings and point `GameUserSettingsClassName` at it in DefaultEngine.ini. That is a fine choice inside a single game and a bad one for a plugin: a project can only ever have *one* such class, so a plugin that claims it is unusable for anyone who already subclasses it - which is most projects of any size.

So ScarOptions keeps its own config object instead. What you get:

- **No ini edit.** Nothing to add, nothing to get wrong, nothing to document for your team.
- **No conflict.** Your project keeps whatever UGameUserSettings class it already has.
- **Same file on disk.** One place, one restart, everything the player chose.

Applying on launch

A game-instance subsystem re-applies audio, gamma and motion blur on every start. The engine restores none of those on its own, which is the usual reason a hand-built options menu "forgets" the volume between sessions. It defers by one tick, because at subsystem-initialize time the audio device and RHI may still be coming up.

On the very first launch on a machine it also runs a hardware benchmark and applies the scalability preset that fits, so the game looks and runs right before anyone opens a menu. It runs once, ever, and can be switched off in Project Settings.

Reset to Defaults never re-benchmarks
The first-run flag is deliberately not cleared by a reset. Resetting is a player action and must not stall the game on a hardware benchmark. Use `Auto Detect Best Settings` - or the Auto-Detect button in the editor tab - when you actually want the benchmark to run again.

4. Project Settings

Edit → **Project Settings** → **Plugins** → **Scar Options.** Every field here is optional; the menu works with the page untouched.

Group	What it controls
Sound Classes	Master / Music / SFX / Voice `USoundClass` assets. Leave empty and the Master slider still works via the device primary volume.
First Run	**Run First Time Auto Detect** - the once-ever hardware benchmark. On by default.
Controls	**Auto Discover Bindings** (on by default) lists every action and axis your project defines. Turn it off and fill **Rebindable Actions** / **Rebindable Axes** to control exactly what the player may touch. **Hidden Bindings** removes specific names from either system. **Include Enhanced Input Bindings** adds the Enhanced Input side; **Input Mapping Contexts** registers contexts the game has not applied yet, so their keys can be rebound from a menu first.
Defaults	What a fresh install starts from, and what Reset restores: the four volumes, sensitivity, camera shake, gamma, motion blur, and the two toggles.
Ranges	**Min/Max Mouse Sensitivity** and **Min/Max Gamma** are the ends of those sliders. **Frame Rate Limits** is the drop-down list; 0 renders as Uncapped.

Set the sensitivity range early
Sensitivity is the most game-dependent value in the whole menu - a slow tactical shooter and a twitchy arena game want completely different ends. The default 0.1x to 3.0x is a starting point, not an answer. Set it once, before players build muscle memory on a range you later change.

5. The Controls page

Click a row, press a key. Escape cancels and leaves the row exactly as it was. Delete or Backspace clears the binding. Mouse side-buttons bind like any other key - left-click is excluded, because that is how the row was clicked in the first place.

It takes effect immediately

A rebind writes straight into the project's action and axis mappings and rebuilds the key maps, so it is live on the *next key press* - not the next launch. It then persists to:

```
Saved\Config\<Platform>\Input.ini
```

One row per axis direction

An axis such as `MoveForward` is really two bindings - W at +1 and S at -1. Giving it a single row would mean rebinding "forward" silently destroys "back". ScarOptions builds one row per direction and labels them `MoveForward (+)` and `MoveForward (-)`, editing only the mapping with that exact key and scale.

Analog sticks and mouse axes are skipped - they are not single keys, and binding one to a keyboard key would break the axis.

Reset All Bindings is a real reset

It reads `Config/DefaultInput.ini` - the file `SaveKeyMappings` never writes to - so it restores what the game actually shipped with, rather than re-saving whatever the player already changed. It only touches binds the player was allowed to edit, so debug and dev-only binds are never destroyed. If no defaults are found it logs a warning and does nothing, rather than wiping every binding.

Both input systems, on one page

Enhanced Input and the legacy Action/Axis mappings are unrelated systems, and ScarOptions reads both. Whatever your project defines is detected and listed together, so a game built on either - or midway through migrating between them - needs no configuration at all. A player rebinding "Jump" does not care which system it came from, so the page does not make them care either.

System	What is listed, and what it takes
Legacy Action / Axis	Every action and axis the project defines. Axes get one row per direction. Rebinds persist to Saved/Config/<Platform>/Input.ini.
Enhanced Input	Every mapping in every Input Mapping Context under /Game, with no flags needed. One a designer marked player-mappable is rebound by Enhanced Input itself; every other is rebound by ScarOptions editing the context in memory, saved to GameUserSettings.ini and restored whenever a game world loads. Axes - thumbsticks, mouse movement - are never listed.

No flags, and what that means in the editor
Enhanced Input's own rule is that only mappings marked player-mappable can be rebound. ScarOptions keeps that route for mappings you have marked - Player Mappable Key Settings from UE 5.3 with Enable User Settings ticked, Is Player Mappable on 5.2 - and takes the engine's older route for everything else: the mapping context is edited in memory, which is exactly how rebinding worked before 5.3. Those edits are undone when the session ends, so Play In Editor never leaves a context changed; the one thing not to do is save an Input Mapping Context while playing in the editor. Contexts outside /Game are not found on their own - list them in Project Settings. **ScarOptions.DumpBindings** shows exactly what the page can see.

Restyling a rebind row

With no art, each row builds its own button and caption. To style them, make a Widget Blueprint parented to `ScarOptionsKeyBindRow`, add a Button named `KeyButton` with a TextBlock named `KeyLabel` inside it, and set it as **Key Bind Row Class** on the menu widget.

6. Undo and autosave

The amber arrow

When the panel opens, every row's value is recorded. From then on, a row whose value differs from that record shows a small undo arrow on its right; clicking it puts that one setting back. A panel nobody has touched shows no arrows at all, so the arrows are a live diff of what this visit changed - not a row of decorations.

The baseline is captured on construct, which means closing and reopening the panel makes the current values the new baseline. That is deliberate: once the player has left the screen, those values are simply what they have.

Autosave

Changes are written to disk a short time after the last edit - 0.35 seconds by default. The delay is a debounce: dragging a slider fires hundreds of change events, and without it every one of them would be a disk write.

Closing the panel flushes any write still pending, so the last change a player makes before hitting the X is never lost.

Because of all that, there is no Apply or Save button in the default layout - there is nothing left for them to do. If you want them back for familiarity, tick **Show Legacy Action Buttons** and the Apply / Save / Reset / Cancel row is built at the bottom.

Live apply

Apply On Every Change is on by default, so the player *sees* the effect of a graphics setting the moment they change it rather than after committing. Turn it off if the apply cost is painful on your minimum-spec hardware.

7. Blueprint reference

Anywhere - the settings object

`Get Scar Options Settings` returns the live settings object from any graph. Everything below is called on it.

Node	Purpose
Set / Get Master Volume Music, Sfx, Voice	0-1. Setters apply immediately.
Apply Audio Settings	Push all four volumes into the audio engine.
Set / Get Mouse Sensitivity	Clamped to the range in Project Settings.
Set / Get Camera Shake Scale	0-1 multiplier. See section 8.
Set / Get Gamma Value	Applies to DisplayGamma and persists.
Set / Get Motion Blur Enabled	Drives r.MotionBlurQuality.
Apply Scar Options	Re-push audio, gamma and motion blur into the running game.
Set To Defaults	Restore the Project Settings defaults. Never re-benchmarks.
Save / Load	Write to, or re-read from, GameUserSettings.ini.
Auto Detect Quality Now	Benchmark and apply. This is the menu's Auto-Detect.
Has Run First Time Setup	True once the once-ever benchmark has happened.
Get Engine Settings	The engine's own UGameUserSettings, if you need it directly.
On Scar Option Changed	Event. Fires with the id of whatever changed - bind this instead of polling on Tick.

Helper nodes

Node	Purpose
Open Scar Options Menu	Create the menu and add it to the viewport.
Play Camera Shake Scaled	Start a shake already multiplied by the player's setting.
Apply Camera Shake Setting	The multiplier on its own.
Apply Mouse Sensitivity	A value multiplied by the player's sensitivity.

Node	Purpose
Get Scar Options Version	The plugin's VersionName.
Get Engine Version String	The engine this build was compiled against.
Are Options Ready	True once the boot pass has applied the saved options.

On the menu widget

Node / Event	Purpose
On Close Clicked	**Dispatcher.** Fired by the X and any Back button, before the menu removes itself. Bind it from whatever opened the menu.
BP On Back Clicked	**Event.** The same moment, for a Blueprint subclass of the menu to override.
BP On Category Changed	**Event.** Fires on every tab change, for restyling a custom tab strip.
Show Category	Switch page from your own button.
Get Active Category	Which page is showing.
Auto Detect Best Settings	Benchmark, apply, and refresh every control.
Reset To Defaults	Engine defaults plus the ScarOptions defaults.
Cancel Changes	Revert to the last values written to disk.
Undo Setting	Revert one row by id - see the id list below.
Refresh All Controls From Settings	Re-read every control. Rarely needed.
Auto Save Now	Force the write immediately.
Reset Key Bindings To Defaults	Same as the button on the Controls page.

Setting ids

For `Undo Setting` and for reading `On Scar Option Changed`:

```
Resolution      WindowMode      OverallQuality   ViewDistance
AntiAliasing    PostProcessing  Shadows          Textures
Effects         Foliage         Shading          VSync
FrameRate       ResolutionScale Gamma            MotionBlur
MasterVolume    MusicVolume     SfxVolume        VoiceVolume
MouseSensitivity  CameraShake
```

8. Camera shake and sensitivity

Most of this menu applies itself, Mouse Sensitivity included. Camera Shake is the one setting the engine has no global hook for - there is no such thing as a project-wide shake multiplier - so your game applies it at the point of use, with one node.

Camera Shake

Replace `Start Camera Shake` with `Play Camera Shake Scaled` and you are done - it applies the player's multiplier for you, and skips the shake entirely when they have turned it to zero rather than starting one at zero amplitude that still occupies a modifier slot for its full duration.

```
Play Camera Shake Scaled
    Player Controller = Get Player Controller
    Shake Class       = your shake
    Base Scale        = 1.0
```

If you start shakes some other way, `Apply Camera Shake Setting` gives you the multiplier on its own.

Mouse Sensitivity

Applies itself. The legacy mouse axes get the setting through the sensitivity the project configured for them, scaled; Enhanced Input gets it through a **Scar Options Mouse Sensitivity** modifier that ScarOptions adds at runtime to every mapping that reads a mouse axis. At 1.0 nothing changes. To apply the setting to something else, add that modifier to the action or mapping yourself; to take over entirely, switch **Auto Apply Mouse Sensitivity** off in Project Settings and run your look input through `Apply Mouse Sensitivity` before it becomes rotation.

Reacting the moment it changes
Both setters fire **On Scar Option Changed** with the **MouseSensitivity** or **CameraShake** id, so anything that caches the value can refresh itself the instant the player moves the slider rather than at its next use.

9. Styling and custom layout

Three ways in, in increasing order of how much you want to change: adjust properties and change nothing else; start from a generated copy of the built-in panel; or place every control yourself.

Start from a copy of the built-in panel

Tools → Scar Options → Create Editable Menu Blueprint. This generates a Widget Blueprint of the whole panel into your project and opens it - background, header, title, tab strip, and every setting row laid out as it appears in game. Move things, restyle them, delete what you do not want.

It behaves like the built-in panel because it *is* the built-in panel: every widget is named so the plugin binds to it, the category tabs still filter the rows, values still load and save, and the drop-downs and tabs are styled by the same code the runtime uses. Nothing about the generated asset is a copy that can drift - the colours and metrics are read from the widget's own defaults when it is built.

A second button, **Create Editable Menu (frame only)**, generates just the surround and leaves the rows to build themselves at runtime as they always have. Reach for it when you only want to restyle the frame and would rather not have thirty rows in your hierarchy.

Both are also available from the console as `ScarOptions.GenerateMenu` and `ScarOptions.GenerateMenu frame`, so the asset can be regenerated from a script or a build step. Run that way on an editor with no window - `-nullrhi` with `-ExecCmds` - the command saves the new asset to disk itself, since there is nobody there to press Ctrl+S.

> **Why it is generated rather than shipped**
> Shipping a Widget Blueprint would mean a Content folder every project cooks whether it was customised or not, an asset that cannot open on an engine older than the one it was saved in, and an update path that either overwrites your edits or cannot reach them. Generating it avoids all three: the asset is made by your engine, in your project, only when you ask - and from that moment it is yours. The plugin itself still ships no content.

Two things are deliberately left out of the generated asset. **Resolution** has no options, because the list comes from the player's own display at runtime and anything written in would be a guess. And the rebind rows are a named empty container rather than a list, because they are built from whatever bindings your project defines.

Styling without touching layout

Select the widget and use the Details panel. Everything is grouped:

Group	What is in it
Text	Label and value colours, font sizes, the panel title.
Layout	Label column width, row padding, content padding, minimum row height.
Background	Blur on/off, blur strength, tint colour.

Group	What is in it
Dropdown	Closed-state background, text colour, list item / hover / selected colours, font size, padding.
Buttons	Normal / hovered / pressed / text colours, optional 9-slice texture and its margin, size, font size, padding.
Tabs	Selected and unselected colours for both the tab and its text, tab size, font size.
Undo	Icon texture, button size, icon colour. With no texture, C++ draws a readable text arrow instead.

Building your own look

Everything on the panel can be hand-placed in the UMG Designer, and controls are matched **by name**. Drop a Slider on your canvas, call it `MasterVolumeSlider`, and it *is* the master volume slider: ScarOptions binds it, pushes the saved value into it when the panel opens, and writes the player's changes back. Style it however you like - the plugin never touches its appearance.

Anything you do not supply is still built for you, so this is a dial rather than a switch:

You supply	You get
Nothing	The full built-in panel.
A few named widgets	Yours where you placed them, built rows for everything else.
Your own layout and controls	Only what you placed. C++ builds no rows at all.

Hand-placing even one control tells ScarOptions the layout is yours, so it stops building its own panel around it.

Layout slots

Widget name	Type	Receives
ControlsContainer	Vertical Box	Every row you did not place yourself
TabBar	Horizontal Box	The five category tabs
TitleLabel	Text Block	The panel title
KeyBindContainer	any Panel	The key-rebind rows
CloseButton / BackButton	Button	Both fire On Close Clicked, then close

Widget name	Type	Receives
ApplyButton / SaveButton ResetButton / CancelButton	Button	Optional, legacy

Controls

Page	Widget names
Graphics	OverallQualityCombo, ViewDistanceCombo, AntiAliasingCombo, PostProcessCombo, ShadowsCombo, TexturesCombo, EffectsCombo, FoliageCombo, ShadingCombo (Combo Box String) MotionBlurCheck (Check Box)
Display	ResolutionCombo, WindowModeCombo, FrameRateCombo (Combo Box String) VSyncCheck (Check Box) ResScaleSlider, GammaSlider (Slider) ResScaleValueLabel, GammaValueLabel (Text Block)
Audio	MasterVolumeSlider, MusicVolumeSlider, SfxVolumeSlider, VoiceVolumeSlider (Slider) MasterVolumeLabel, MusicVolumeLabel, SfxVolumeLabel, VoiceVolumeLabel (Text Block)
Gameplay	SensitivitySlider, CameraShakeSlider (Slider) SensitivityLabel, CameraShakeLabel (Text Block)

Three rules worth knowing

- **Sliders are forced to 0-1.** Every handler maps that range into the real one itself, so a slider left at some other range would drive its setting to the wrong place - silently, which is worse.
- **Drop-downs keep options you typed**, because selection is by index. The two exceptions are **Resolution** and **Frame Rate Limit**, which ScarOptions always fills: a hand-typed resolution list is wrong on every machine but yours, and frame-rate entries have to stay in step with the limits in Project Settings.
- **Value labels are optional.** Supply `MasterVolumeLabel` and it gets "75%" written into it; leave it out and nothing is drawn.

Rebind rows

Make a Widget Blueprint parented to `ScarOptionsKeyBindRow` containing a Button named `KeyButton`, a Text Block named `KeyLabel` inside it, and - worth adding - a Text Block named `ActionLabel`, which receives the binding's name. Set it as **Key Bind Row Class**, drop a `KeyBindContainer` where you want the list, and the rows are yours. Place your own reset button and call `Reset Key Bindings To Defaults` from it.

Name your close button something else to own it completely
If you want to wire a close button entirely in Blueprint, either untick **Auto Bind Close Button** or simply give your button any name other than CloseButton or BackButton. C++ then never binds it, never styles it and never replaces it - two handlers fighting over one click is a bug that is genuinely unpleasant to find.

Input behaviour

Auto Capture Mouse On Show (on by default) takes UI-only input on construct and re-asserts the cursor while the panel lives, defending against other UI that hides the cursor on its own tear-down. On close it hands input back as GameAndUI, because whatever opened the options is usually still on screen and still needs a cursor.

Restore Cursor On Destroy is off by default. Turn it on only when the options panel is the only thing on screen and you want the cursor hidden and GameOnly input restored when it closes.

10. Console commands

Command	Effect
ScarOptions.AutoDetect	Benchmark this machine and apply the preset that fits.
ScarOptions.Reset	Restore the Project Settings defaults and save.
ScarOptions.Reapply	Re-apply saved audio, gamma and motion blur.
ScarOptions.DumpBindings	Log every binding the Controls page can list - legacy actions and axes, then Enhanced Input. Run it first when the page looks wrong.
ScarOptions.RebindKey <Name> [<DefaultKey>] <Key>	Rebind an Enhanced Input mapping without the menu: a player-mappable name, or an action name and its default key. Omit the key to unbind.
ScarOptions.GenerateMenu	Generate the editable Widget Blueprint. Add **frame** for the surround only. Editor only.

The editor tab

Tools → **Scar Options** opens a panel with the plugin and engine versions, a shortcut to the Project Settings page, the two buttons that generate an editable copy of the menu (section 9), and the two actions worth having without launching the game: **Run Hardware Auto-Detect Now** and **Reset Saved Player Options**. The last is the fastest way to see exactly what a brand new player sees.

11. Engine compatibility

ScarOptions ships as **one zip per engine release, UE 5.2 through 5.8**. Download the one matching your engine.

Each zip contains source written for that engine and nothing else. There are no version macros in the code you receive - no `#if ENGINE_MAJOR_VERSION` ladders, no branches for engines you are not using. If you open the 5.4 zip you are reading plain 5.4 code, which matters the moment you want to fork it or step through it in a debugger.

Three things inside, and only three:

- `ScarOptions.uplugin` - the descriptor, stamped to that engine version.
- `Source/` - the two modules, runtime and editor.
- `Resources/` - the plugin icon.

No Config folder, no Content folder, no sample maps, nothing to delete before you ship. The plugin adds no assets to your project and cooks nothing you did not ask for.

Installing

Unzip into your project's `Plugins/` folder so you end up with `YourProject/Plugins/ScarOptions/`, then regenerate project files and build. The plugin is source-only, so it compiles alongside your game module - there are no prebuilt binaries to mismatch your engine build.

The source is C++14-clean and pure ASCII, so nothing depends on a newer language standard than the oldest supported release sets by default, and no compiler has to guess at a file's encoding.

12. Troubleshooting

Symptom	Cause and fix
The panel is empty	A `ControlsContainer` was added in the Designer but the widget is not reparented to ScarOptionsMenuWidget - so nothing populates it. Check the parent class.
Clicking a button does nothing the first time	Something else on screen is taking focus. ScarOptions already makes its own buttons fire on mouse-down for exactly this reason; if a button you added yourself behaves this way, that is why.
Volume sliders move but nothing gets quieter	Only Master works without configuration. Assign your Sound Class assets in Project Settings and the other three drive real buses.
Settings do not survive a restart	Check that the game is shutting down cleanly - the write happens moments after a change, not at exit. Confirm by looking for the values in Saved\Config\<Platform>\GameUserSettings.ini.
Gamma or volume resets every launch	That is the engine's behaviour and exactly what the boot subsystem exists to fix. If it is happening, the subsystem is not running - check the plugin is enabled for the target you are running.
The Controls page is empty on an Enhanced Input project	No Input Mapping Context with a key a player could press was found. Contexts are found under /Game with Auto Discover Bindings on, and from Project Settings → Plugins → Scar Options → Input Mapping Contexts; a context whose only keys are axes lists nothing. Run **ScarOptions.DumpBindings** to see exactly what the page can see.
Reset All Bindings does nothing	No defaults were found in `Config/DefaultInput.ini`. ScarOptions skips the reset rather than wiping every binding, and logs a warning saying so.
A rebind row says PRESS A KEY forever	The row lost keyboard focus before a key arrived. Click it again. Clicking elsewhere while listening cancels on purpose.
Undo arrows never appear	They only show for rows on the page you are looking at, and only once a value differs from what it was when the panel opened. Reopening the panel makes the current values the new baseline.
Camera shake option seems to do nothing	Your game is still calling `Start Camera Shake`. Swap it for `Play Camera Shake Scaled` - see section 8.

Support

Discord: https://discord.gg/a3QdMX9JZj