
Scar Rewind System

Complete User Guide - Version 4.0

Rewind time. Now for everyone in the session.

Version 4.0 makes Scar Rewind System fully multiplayer. Drop the component on a replicated pawn, wire Start Rewind to a key, and the rewind runs for every player in the session - server authoritative, smooth on every client, and about thirty bytes of bandwidth per rewind. Single player projects keep working exactly as they did, with no changes at all.

What you get

- **One node, everywhere.** [Scar Start Global Rewind](#) and [Scar Start Rewind For Actor](#) route themselves. The same node is correct in single player, on the server, and on a client.
- **Server authoritative.** The server validates history, cooldown and energy, and owns the final position. A modified client cannot teleport itself, and can only ever ask to rewind a pawn it actually controls.
- **Smooth on every client.** Each machine replays its own recorded history from a shared timeline, so playback runs at the client's own framerate rather than at the network update rate.
- **Nobody misses it.** Ghosts, sounds, badges and a full-screen effect all reach every player, plus events that tell your HUD who rewound.
- **Almost no bandwidth.** A rewind is one small replicated struct, not a per-frame transform stream. A world rewind costs the same whether it moves five Actors or five hundred.
- **Still zero configuration.** No GameMode changes, no replication graph work, no extra Actors to place. The plugin manages its own networking.

Plugin version 4.0.0 | *Engine versions* UE 5.2 - 5.7 | *Zero dependencies* | *Blueprint-first*

(c) 2026 Black Scar Studio | blackscarstudio.com

Table of Contents

Section	Topic
1	What Is New in Version 4.0
2	How Multiplayer Rewind Works
3	Multiplayer Quick Start
4	Single Player Quick Start
5	Creating Your Ghost Blueprint Actor
6	Component Properties Reference
7	Blueprint Node Reference
8	The Scar Rewind Blueprint Library
9	Events and Delegates
10	Making Sure Every Player Notices
11	Global World Rewind
12	Pickup Exclusion from Global Rewind
13	Batch Operations via Subsystem
14	Security and Authority
15	Game Design Recipes
16	Multiplayer Troubleshooting
17	Performance and Bandwidth
18	Testing and Compatibility
19	Version 3.0 Crash Fixes
20	File Reference

What Is New in Version 4.0

Version 3.0 was a complete single player rewind system. Version 4.0 keeps every one of those features and adds full multiplayer replication on top, without changing how anything behaves in a single player project.

Multiplayer

Feature	What it means
Replicated per-actor rewind	A rewind on any Actor with the component is seen by every player. The server validates and owns the outcome.
Global multiplayer rewind	One node rewinds the world for the entire session. Every player gets the effect, the sound and the event.
Client prediction	A client starts its own playback immediately instead of waiting a round trip, then reconciles smoothly to the server's answer.
Smooth reconciliation	When a rewind ends, clients blend onto the server's authoritative resting transform instead of snapping.
Replicated energy and cooldown	Server owned, replicated only to the owning player, so the HUD is always right and cannot be faked.
Client request channel	Clients ask the server through a per-player proxy the plugin spawns and owns. Nothing to set up.
Rewind cues for observers	Ghost, start and stop sounds, and an optional marker Actor, all visible to the other players rather than only to the rewinder.
Blueprint node library	26 nodes callable from any Blueprint with no component or subsystem reference.

Improvements that also help single player

- **Rewind speed is now accurate.** Playback samples history by timestamp rather than by snapshot index. [Rewind Speed](#) 1.0 now genuinely means real time. Previously, because idle snapshots are skipped, a rewind travelled further than the speed suggested whenever the Actor had been standing still.
- **[Get Recorded Duration](#) is now exact.** It measures real elapsed time from the stored timestamps instead of estimating from the snapshot count.
- **Post process fades.** A new [Post Process Blend Time](#) eases the effect in and out instead of popping it on, which matters a lot when several rewinds overlap.

-
- **A clear warning for the most common setup mistake.** If the component is on a non-replicated Actor in a networked game, the log says so in plain English.

Upgrading from 3.0

Nothing to do. Existing Blueprints keep working, every 3.0 node and property still exists, and in a standalone game every new routing node falls through to the same local path 3.0 used. [Start Global Rewind](#) gained an optional Instigator pin, tucked behind the advanced arrow, so existing graphs look unchanged.

How Multiplayer Rewind Works

You do not need to read this section to use the plugin. It is here because the design explains why rewind stays smooth on a bad connection, and knowing it helps when you tune the settings.

A rewind is not streamed

The obvious way to network a rewind is to move the Actor on the server and let normal movement replication carry it to the clients. That is expensive, and it still looks wrong: updates arrive at the network update rate rather than at the player's framerate, and a teleport backwards defeats the movement smoothing that would normally hide the gaps.

Scar Rewind System does something cheaper and smoother.

1. Every machine already records its own history

Recording was always local, on the server and on every client alike, so this costs nothing extra. Each machine has its own buffer of where the Actor has been.

2. The server publishes one small struct

When a rewind starts, the server replicates when it started, where in the timeline it started, and how fast it plays back. That is the entire network cost of the rewind.

3. Every machine replays its own buffer

The playback cursor is a pure function of synchronised server time, so each machine independently arrives at the same moment in the timeline and draws it at its own framerate. There is no drift to accumulate, and a player who joins mid-rewind lands at exactly the right point.

4. The server stays in charge of the result

The server runs the same playback, and its transform is the real one. When the rewind ends it sends the authoritative resting transform, and clients ease onto it over Reconciliation Blend Time so there is never a visible snap.

The cursor

```
Cursor = StartCursorTime - (ServerTimeNow - StartServerTime) * PlaybackSpeed
```

Because this is a function of shared time rather than a running total, a dropped packet or a stalled frame cannot make a client drift out of step. It simply evaluates the formula again on the next frame and is correct.

Why sampling by time, not by index

Each machine skips recording snapshots for an Actor that has barely moved, and they each skip slightly different ones. That means snapshot number 40 is a different moment on the server than it is on a client. Timestamps are absolute and shared, so sampling by time is what lets every machine agree.

What this means in practice

- Rewinding one Actor and rewinding five hundred cost the same bandwidth.
- Playback is as smooth as the client's framerate, not as smooth as its ping.
- Remote players rewind through the history that client actually saw, which is offset by roughly half the round trip - the same offset that already exists for remote Actors during normal play, so it reads as consistent.
- The server can refuse a rewind and the client rolls its prediction back smoothly.

Multiplayer Quick Start

If your pawn is already replicated, this is the whole setup. There is no GameMode change, no replication graph work, and no Actor to place in the level.

1. Enable the plugin

Edit, then Plugins. Search Scar Rewind System, tick Enabled, restart the editor.

2. Make sure your Actor replicates

Open your pawn Blueprint. In Class Defaults, under Replication, tick **Replicates**. This is the one thing that is easy to forget - the rewind travels through the component, and a component cannot replicate if its Actor does not.

3. Add the component

Components panel, Add Component, search **Scar Rewind Component**. Recording starts automatically on Play.

4. Wire your input to Start Rewind and Stop Rewind

Exactly as in single player. Drag in the component and call **Start Rewind** on pressed and **Stop Rewind** on released. The node works out for itself whether it is on the server or a client and does the right thing.

5. Test with two players

In the editor toolbar, set Number of Players to 2 and Net Mode to **Play As Listen Server** (or Play As Client for a dedicated server). Press Play and rewind on one window - watch it happen in the other.

That really is all of it

You do not call a Server RPC, you do not check Has Authority, and you do not write an RPC of your own. **Start Rewind** routes itself: on the server it starts the authoritative rewind for everyone, and on a client it asks the server and predicts locally so it feels instant.

Choosing who is allowed to rewind

The component's **Rewind Net Mode** decides this. It is on the Multiplayer category in the Details panel.

Mode	Behaviour	Use it for
Client Request (default)	A client presses rewind, the server checks history, cooldown and energy, then runs it for everybody.	Player abilities. This is what almost every game wants.
Server Only	Clients cannot start a rewind at all. Only server code can.	Boss mechanics, scripted events, traps, admin commands.

Mode	Behaviour	Use it for
Local Only	No replication whatsoever. Each machine rewinds only itself.	Purely cosmetic rewinds, or a single player mode inside a multiplayer project.

Single Player Quick Start

The Scar Rewind System is fully Blueprint-native. Every feature is accessible through Blueprint nodes. No C++ is required at any stage.

1. Enable the Plugin

Go to Edit, then Plugins. Search for Scar Rewind System. Check Enabled. Click Restart Now. The plugin appears under the Gameplay category.

2. Add the Component to Your Actor

Open any Blueprint Actor. In the Components panel click Add Component and search Scar Rewind Component. Select it. Recording begins automatically when you press Play, no extra setup needed.

3. Wire Up Input, Start Rewind

In the Event Graph, right-click and add your Input Action event (for example, IA_Rewind). Drag the Scar Rewind Component reference into the graph. Connect the Started or Pressed exec pin to the [Start Rewind](#) node.

4. Wire Up Input, Stop Rewind

From the same input event, connect the Completed or Released exec pin to the [Stop Rewind](#) node on the component. That is the complete minimal setup.

5. Test In-Editor

Press Play. Enable **Show Debug** on the component to see a live status overlay floating above the Actor. It shows state, buffer fill, energy, and in a networked game which machine you are looking at.

TIP

Enhanced Input is recommended but not required. The component works with any input method, including legacy Input Action, keyboard events, or custom input systems.

Creating Your Ghost Blueprint Actor

The ghost trail spawns a Blueprint Actor you create and fully control. This gives you complete creative freedom. Any mesh, material, particle effect, or animation can serve as the ghost echo. The component handles spawn, position updates, and destruction automatically.

1. Create a New Blueprint Actor

In the Content Browser right-click, choose Blueprint Class, then Actor. Name it something like BP_RewindGhost.

2. Add a Static Mesh Component

Inside BP_RewindGhost, click Add, then Static Mesh Component. Assign the same mesh your character or prop uses.

3. Apply a Translucent Material

Create a Material with Blend Mode set to Translucent and Lighting Model set to Unlit. Set Opacity to around 0.35. Apply it to the mesh. You can also add a Niagara component for a particle-based echo instead of or alongside the mesh.

4. Disable Collision

Select the mesh component. In Details, Collision, set Collision Presets to NoCollision. This prevents the ghost from blocking the player or other actors.

5. Leave Replicates switched OFF

This matters in multiplayer. Every machine spawns its own copy of the ghost locally, which is cheaper and appears instantly. A replicated ghost would be duplicated and arrive late. The plugin forces replication off on the spawned ghost anyway, and tags it so a world rewind never rewinds the effect itself.

6. Assign in the Rewind Component

Select your Actor that has the Scar Rewind Component. In Details, Scar Rewind System, Ghost Trail, set Ghost Actor Class to BP_RewindGhost and enable Show Ghost Trail.

7. Set the Linger Duration

Set Ghost Trail Linger Duration to how many seconds the ghost should stay visible after rewind ends. 1.5 seconds is a good starting point; 0 destroys it immediately.

Who sees the ghost

By default everyone does - **Show Ghost To Other Players** is on. That is usually what you want, because the ghost is how other players read that somebody rewound. Turn it off if the ghost should be private to the rewinding player.

Component Properties Reference

All properties are editable from the Details panel in the Blueprint editor. No code is required to configure any of these settings.

Core

Property	Default	Description
Buffer Duration	10 s	Seconds of history stored. Raise for longer rewinds, up to 60 seconds.
Recording Interval	0.05 s	Snapshot frequency. 0.05 seconds equals 20 snapshots per second.
Rewind Speed	1.5x	Playback speed multiplier. 1.0 equals real time, 2.0 equals twice as fast.
Location Threshold	2 cm	Minimum movement before saving a snapshot. Skips idle frames.
Rotation Threshold	0.5 deg	Minimum rotation before saving a snapshot.
Preserve Pawn Rotation	On	For Pawn owners, only the location travels back, not the rotation. Keeps the camera from snapping to a stale facing direction. Turn off only if body facing during rewind is meaningful in your game.

Multiplayer

Property	Default	Description
Rewind Net Mode	Client Request	Who may start a rewind. Client Request, Server Only, or Local Only. See Section 3.
Predict Rewind On Client	On	The client starts its visuals immediately instead of waiting a round trip. The server still decides; a refusal rolls back smoothly.
Reconciliation Blend Time	0.15 s	How long clients take to ease onto the server's resting transform when a rewind ends. This is the setting that removes the final snap. 0 corrects instantly.

Property	Default	Description
Suspend Movement During Rewind	On	Stops the movement component simulating while the rewind plays. Without it, Character Movement fights the playback and produces stutter.
Block Input During Rewind	On	Ignores player input on the owning controller so the player cannot walk away from their own rewind.
Suspend Movement Replication During Rewind	On	Every client already replays the rewind locally and smoothly, so streaming the same movement again would only fight them and waste bandwidth.

Energy System

Property	Default	Description
Use Rewind Energy	Off	Enable built-in stamina. Rewind stops automatically at zero energy.
Max Rewind Energy	100	Maximum size of the energy pool.
Energy Drain Rate	20 / s	Energy consumed per second while rewinding.
Energy Recharge Rate	10 / s	Energy restored per second when idle.
Min Energy To Start Rewind	10	Minimum energy required to begin a new rewind.

Energy in multiplayer

The server owns the real value and replicates it only to the player who owns the Actor, so it costs no bandwidth for anyone else and cannot be faked. The owning client predicts the same drain and recharge locally so the bar moves smoothly between updates. A world rewind spends nobody's energy - it is a world event, not the player's ability.

Cooldown and Axis Lock

Property	Default	Description
Rewind Cooldown	0 s	Seconds before another rewind can start after one ends. 0 means no cooldown.
Lock Location X	Off	Prevent rewinding the X world-space position.
Lock Location Y	Off	Prevent rewinding the Y world-space position. Use this for 2D sidescrollers.
Lock Location Z	Off	Prevent rewinding height. Useful for top-down games.
Lock Rotation	Off	Prevent rewinding rotation. Position still rewinds normally.

Ghost Trail

Property	Default	Description
Show Ghost Trail	Off	Spawn the ghost Blueprint Actor during rewind playback.

Property	Default	Description
Ghost Actor Class	None	The Blueprint Actor class to spawn as the ghost. Create your own translucent Blueprint and assign it here.
Ghost Trail Linger Duration	1.5 s	How long the ghost stays visible after rewind ends. 0 destroys it immediately.
Show Ghost To Other Players	On	Show this Actor's ghost to everyone, not just its owner. This is how the rest of the session reads that a player rewound.

Rewind Cues

New in 4.0. These are the cheapest way to make sure a rewind is impossible to miss. All of them play locally on every machine, so they cost no bandwidth.

Property	Default	Description
Rewind Start Sound	None	Played at the Actor's location on every machine when a rewind starts.
Rewind Stop Sound	None	Played at the Actor's location on every machine when it ends.
Rewind Marker Actor Class	None	A Blueprint Actor spawned locally on every machine and attached to the rewinding Actor for the whole rewind. Use it for a Niagara distortion, a floating icon, or a nameplate badge. Destroyed automatically.

Post Process

Property	Default	Description
Use Post Process Effect	Off	Automatically activate a post process effect during rewind.
Post Process For Observers	Off	Also apply the effect when somebody ELSE rewinds. The rewind post process is unbound, meaning full screen, so by default it only appears on the machine that owns the rewinding Actor. Turn this on for a game where every rewind should be felt by everyone.
Post Process Chromatic Aberration	2.0	RGB fringe strength. Gives a time-glitch look. 0 means none.
Post Process Vignette	0.6	Screen-edge darkening during rewind. 0 means none.
Post Process Desaturation	0.8	Colour drain amount. 1.0 equals full grayscale.
Post Process Blend Time	0.25 s	Fades the effect in and out instead of popping it on. Matters when several rewinds overlap.

Priority and Debug

Property	Default	Description
Rewind Priority	5	1 to 10. Higher priority means a larger buffer allocation in the global memory budget.

Property	Default	Description
Show Debug	Off	Draws live state text above the Actor, including which machine you are looking at (AUTH, LOCAL or PROXY) and whether the rewind is predicted. Compiled out in Shipping.
Show Debug Path	Off	Draws the recorded movement path as coloured dots. Also compiled out in Shipping.

Blueprint Node Reference

Every function listed here is available as a Blueprint node on the component. Drag the Scar Rewind Component reference into the Event Graph and pull off a wire to search for any of these nodes.

Core Nodes

Function - Node	Description
Start Rewind	Begin rewinding. Routes itself: single player runs it, the server starts the authoritative rewind for everyone, a client asks the server. Blocked automatically if the buffer is empty, rewind is already active, cooldown is running, or energy is too low.
Stop Rewind	Stop rewinding. Actor stays at the rewound position. Recording resumes automatically.
Clear Buffer	Erase all history. Cannot rewind past this call. All bookmarks are also cleared.
Set Recording Enabled (bool)	Pause or resume snapshot recording without affecting active playback.

Multiplayer Nodes

You normally only need [Start Rewind](#). These exist for when you want to be explicit about which side is acting.

Function - Node	Description
Authority Start Rewind, returns Deny Reason	Server-side. Starts the authoritative rewind and tells every client. Returns None on success, or the reason it was refused. Does nothing on a client.
Authority Stop Rewind	Server-side stop. Broadcasts the authoritative resting transform.
Request Rewind From Server	Client-side. Asks the server to rewind this Actor. Safe to call on the server too.
Request Stop Rewind From Server	Client-side stop request.
Start Rewind Local Only	Rewind on this machine only, with no replication. Other players will not see it.

Function - Node	Description
Stop Rewind Local Only	Stop a local-only rewind.
Can Start Rewind Now, returns Deny Reason	Ask whether a rewind could start right now without starting one. Handy for greying out a UI button.
Get Last Deny Reason, returns Deny Reason	Why the last request was refused. None if it succeeded.
Has Rewind Authority, returns bool	True if this machine owns the authoritative copy of this Actor.
Is Networked Game, returns bool	True when the game is actually networked rather than standalone.
Is Locally Controlled Owner, returns bool	True if the owner is a pawn controlled by this machine's local player.
Get Rewind Net State, returns struct	The replicated description of the rewind currently in flight.
Get Synced Time Seconds, returns float	Network synchronised time, identical on the server and every client. All rewind timestamps live on this clock.

Bookmark Nodes

Bookmarks save a named moment in history. **Restore Bookmark** is an instant teleport - it does not play through the intervening history, it jumps directly to that moment.

Function - Node	Description
Save Bookmark (Name)	Save a named moment at the current buffer head. Up to 16 per component. Re-saving the same name updates it.
Restore Bookmark (Name), returns bool	Instantly move the Actor to the saved position and restore tracked variables. Returns false if the name was not found.
Delete Bookmark (Name)	Remove a single bookmark by name.
Clear All Bookmarks	Remove all bookmarks. Does not clear the snapshot buffer.

TIP

Call **Save Bookmark** when the player enters a safe zone. On death, call **Restore Bookmark** to teleport back instantly with no animation required. In multiplayer, do this on the server so the result replicates.

Energy Nodes

Function - Node	Description
Get Current Energy, returns float	Raw energy value from 0 to Max Rewind Energy.
Get Energy Ratio, returns float	Energy as a 0 to 1 fraction. Wire directly into a Progress Bar Percent binding.
Set Energy (float)	Force-set the energy value. Clamped to 0 through Max Rewind Energy. In multiplayer call this on the server; the new value replicates to the owning client automatically.

State Query Nodes

Function - Node	Description
Is Rewinding, returns bool	True while rewind playback is active on this machine.
Is Recording, returns bool	True while new snapshots are being saved.
Is On Cooldown, returns bool	True while the cooldown timer is running.
Get Rewind State, returns Enum	Returns Idle, Recording, Rewinding, or Paused.
Get Buffer Size, returns int	Number of snapshots currently in the buffer.

Function - Node	Description
Get Buffer Capacity, returns int	Maximum snapshot capacity of the buffer.
Get Buffer Fill Ratio, returns float	Buffer fill as 0 to 1. Use for a rewind-meter progress bar.
Get Recorded Duration, returns float	Seconds of history currently stored, measured from real timestamps.
Get Cooldown Remaining, returns float	Seconds until cooldown expires. Returns 0 when ready.
Get Last Snapshot Custom Floats, returns float[]	The most recently recorded custom float values. Index matches registration order.

The Scar Rewind Blueprint Library

New in 4.0. These nodes work from **any** Blueprint with no component reference and no subsystem to fetch first. Right-click in any graph and type **Scar Rewind**.

World wide rewind

Node	Description
Scar Start Global Rewind (Duration, Instigator)	The one you want. Rewinds the world for every player. Routes itself: single player runs it, the server rewinds for everyone, a client asks the server. Duration 0 uses the full buffer.
Scar Stop Global Rewind	Stops it, with the same routing.
Scar Server Start Global Rewind	Server-side only. Returns a deny reason.
Scar Server Stop Global Rewind	Server-side stop.
Scar Request Global Rewind (Client)	Explicit client ask.
Scar Request Stop Global Rewind (Client)	Explicit client stop ask.
Scar Start Global Rewind (Local Only)	This machine only, no replication.
Scar Stop Global Rewind (Local Only)	Stops a local-only world rewind.
Scar Is Global Rewind Active, returns bool	True while a world rewind is running here.
Scar Get Global Rewind Instigator, returns Actor	Who triggered it, for HUD text.

Per player and per Actor

Node	Description
Scar Start Rewind For Actor (Actor)	The one you want. Rewinds one Actor, replicated so every player sees it. Predicts on the client, validated by the server.
Scar Stop Rewind For Actor (Actor)	Stops it, with the same routing.

Node	Description
Scar Server Start Rewind For Actor	Server-side only. Returns a deny reason.
Scar Server Stop Rewind For Actor	Server-side stop.
Scar Request Rewind For Actor (Client)	Explicit client ask. The server only accepts it for a pawn this client controls.
Scar Start / Stop Rewind For Actor (Local Only)	This machine only, no replication.
Scar Server Rewind All Players, returns int	Rewinds every player pawn that has the component, all at once. Returns how many actually started.
Scar Server Stop All Player Rewinds	The matching stop.
Scar Is Actor Rewinding, returns bool	Query.
Scar Get Rewind Component, returns Component	The component on an Actor, or null.

Utility

Node	Description
Scar Has Rewind Authority, returns bool	True on the server, and in a standalone single player game.
Scar Is Networked Game, returns bool	True when this is a real networked session.
Scar Get Synced Time Seconds, returns float	The shared clock all rewind timestamps live on.
Scar Get Deny Reason Text, returns Text	Turns a refusal into a ready-to-display sentence, for example "Not enough rewind energy." Wire it straight into a Text block.
Scar Get Rewind Subsystem, returns Subsystem	The subsystem, if you want it directly.

Events and Delegates

All events are BlueprintAssignable. In the Blueprint editor, select the Scar Rewind Component in the Components panel, then look in the Details panel under Events and click the green plus button next to any event to add it directly to the graph.

On the component

On Rewind State Changed (blsRewinding)

Fires on this machine when rewind starts (true) or stops (false). Use this to play sounds, trigger VFX, update UI, or lock player movement.

On Rewind Net State Changed (Actor, blsRewinding, blsLocalPlayer) - new in 4.0

Fires on **every** machine, server and all clients, when this Actor starts or stops a replicated rewind. The blsLocalPlayer pin tells you whether to say "You rewound" or "Alice rewound". This is the event to use for anything other players should see.

On Rewind Request Denied (Reason) - new in 4.0

Fires on the client whose request the server refused. Wire the Reason into [Scar Get Deny Reason Text](#) to show the player why nothing happened.

On Rewind Buffer Exhausted

Fires when the buffer is fully consumed and rewind auto-stops. Show a no-more-rewind notification here, or trigger a screen shake or sound.

On Energy Changed (CurrentEnergy)

Fires whenever energy changes by more than 0.01. Bind a Progress Bar to this instead of polling every tick. Fires during both drain and recharge, and after each replication update in multiplayer.

On Bookmark Saved (BookmarkName)

Fires when Save Bookmark succeeds. Use this for a checkpoint-saved notification.

On the subsystem

On Any Rewind Net State Changed (Actor, blsRewinding, blsLocalPlayer) - new in 4.0

The same information as the component event, but raised for **every** Actor in the level. Bind this once in your HUD or GameState instead of hunting down every rewind component in the game.

On Global Rewind Net Event (Instigator, blsActive, Duration) - new in 4.0

Fires on every machine when a world wide rewind starts or stops, and carries who triggered it. This is the one to bind for a session-wide "Alice rewound time!" banner.

On Global Rewind State Changed (blsActive)

The simpler 3.0 version of the same signal, without the instigator. Still present and still works.

NOTE

To bind an event in Blueprint: select the component in the Components panel, open the Details panel, scroll to the Events section, and click the green plus button next to the event you want. Unreal creates the event node in the Event Graph automatically. For subsystem events, use [Get World Subsystem](#) and Assign to the delegate.

Making Sure Every Player Notices

A rewind that other players do not notice reads as a bug - a player teleporting backwards for no visible reason. Version 4.0 gives you four layers to make it unmistakable, and the defaults already cover most of it.

Layer 1 - The ghost

Assign a Ghost Actor Class and turn on Show Ghost Trail. **Show Ghost To Other Players** is on by default, so everyone sees the echo travelling backwards. With a linger duration, the ghost stays behind for a moment at the origin point, which reads clearly even to a player who looked over late.

Layer 2 - Sound

Set **Rewind Start Sound** and **Rewind Stop Sound**. These play at the Actor's location on every machine, so the sound is positional - a player hears which direction the rewind came from. For a world rewind, set the subsystem's **Global Rewind Start Sound**, which plays as a 2D sound so nobody can be out of earshot.

Layer 3 - A marker on the rewinding player

Set **Rewind Marker Actor Class** to a Blueprint Actor. It is spawned locally on every machine, attached to the rewinding Actor for the whole rewind, and destroyed automatically. Put a Niagara time-distortion effect in it, a floating clock icon, or a widget component nameplate that reads "REWINDING".

Layer 4 - The screen

For a world rewind, **Use Global Post Process** is on by default and applies a full screen effect on every player's machine. For a single player's rewind, the post process stays local to that player by default, because a full screen tint every time somebody else rewinds gets tiring fast - turn on **Post Process For Observers** if your game wants everyone to feel it.

Layer 5 - Your HUD

Bind **On Any Rewind Net State Changed** on the subsystem in your HUD. You get the Actor, whether it started or stopped, and whether it is the local player. That is everything you need for a banner, a kill-feed style line, or a marker on a minimap.

Cosmetics are never replicated

Every ghost, marker, sound and effect is spawned locally on each machine. That is deliberate: each client already knows the rewind is happening from the one replicated struct, so replicating the effects would cost bandwidth and make them arrive late. Leave **Replicates** switched off on your ghost, marker and FX Blueprints.

Global World Rewind

Global World Rewind rewinds every eligible movable Actor in the level simultaneously with a single Blueprint node. Enemies, projectiles, props, and vehicles are all rewound together. Actors with a Scar Rewind Component use their own per-actor buffer. All other eligible movable Actors use the global snapshot cache built by the subsystem.

In multiplayer it does all of that for every player at once, for the same one-struct cost. Use **Scar Start Global Rewind** from the Blueprint library, or fetch the subsystem with **Get World Subsystem** and call the same node there.

Global Rewind Nodes

Function - Node	Description
Start Global Rewind (Duration, Instigator)	Rewind every eligible movable Actor, for every player. Duration 0 uses the full buffer. Duration 3.0 rewinds exactly 3 seconds back.
Stop Global Rewind	Stop global rewind. All Actors resume from their rewound positions.
Is Global Rewinding, returns bool	True while global rewind is active on this machine.
Get Global Rewind Instigator, returns Actor	Who triggered the running world rewind, or null.
Has Global Rewind Authority, returns bool	True on the server, and in standalone.

Global Rewind Settings

Property	Default	Description
Global Buffer Duration	10 s	History kept for Actors without a Scar Rewind Component.
Global Record Interval	0.05 s	Snapshot frequency for the global cache.
Global Rewind Speed	1.5x	Playback speed for all Actors during global rewind.

Property	Default	Description
Allow Client Triggered Global Rewind	On	Whether a client may ask the server for a world rewind. The server still validates and still owns the outcome. Turn off for a competitive game where only server logic should rewind the world.
Min Time Between Global Rewinds	0 s	Anti-spam guard. A second request inside this window is refused with the On Cooldown reason. 0 disables it.
Suspend Movement Replication During Global Rewind	On	Clients already replay the rewind locally and smoothly, so streaming the same movement again would fight them.

Global Visuals

Property	Default	Description
Use Global Post Process	On	Full screen effect on every player's machine while the world rewinds.
Global Post Process Chromatic Aberration	3.0	RGB fringe strength.
Global Post Process Vignette	0.7	Screen-edge darkening.
Global Post Process Desaturation	0.85	Colour drain. 1.0 equals grayscale.
Global Post Process Blend Time	0.3 s	Fade in and out rather than popping.
Global Rewind FX Actor Class	None	A Blueprint Actor spawned locally on every machine for the whole world rewind. Niagara storm, screen warp, clock overlay, announcement widget - anything.
Global Rewind Start / Stop Sound	None	Played as a 2D sound on every machine.

Energy and cooldown

A world rewind does not spend any player's rewind energy and does not trigger their personal cooldown. It is a world event, not their ability. Use **Min Time Between Global Rewinds** if you need to rate-limit it.

Pickup Exclusion from Global Rewind

Global rewind does not affect pickups. Rewinding the world while the player has already collected an item would cause confusing behaviour, for example a weapon disappearing from the player's hand or a health pack reappearing and then vanishing again as the timeline moves. The system provides several automatic methods to keep pickups out of the global rewind.

Method 1 - Tag the Pickup Actor (Recommended)

Add the tag **ScarIsPickup** to any Actor's Tags array in the Details panel. The global rewind system will skip this Actor entirely, whether or not it has been picked up. Use this for weapons, health packs, ammo, keys, collectibles, and any other item the player can interact with.

Method 2 - Ownership Detection (Automatic)

When a player picks up an item, most inventory and equipment systems set the item's Owner to the Pawn that collected it. The global rewind system detects this automatically and skips any Actor whose Owner is a Pawn subclass.

Method 3 - Attachment Detection (Automatic)

Many weapon systems socket an item onto the character mesh with Attach To Component and never set Owner at all. Any Actor attached to a Pawn, or attached to an Actor whose Owner is a Pawn, is skipped too. This is the check that prevents the classic "the weapon mesh drops to the floor but I can still fire it" bug.

Summary of All Exclusion Rules

Condition	Result
Actor has Mobility = Static	Always excluded. Static Actors cannot move.
Actor has tag ScarIgnoreGlobal	Excluded. The explicit opt-out.
Actor has tag ScarIsPickup	Excluded permanently, whether collected or not.
Actor's Owner is a Pawn	Excluded. Already held.
Actor is attached to a Pawn	Excluded. Already equipped.
Actor has a Scar Rewind Component	Excluded from the global cache - it rewinds itself instead.
Actor is an Info Actor	Excluded. No world presence. This also covers the plugin's own networking Actors.

IMPORTANT

The ScarIsPickup tag excludes the Actor permanently, even before the player picks it up. If you want un-picked items to rewind with the world but picked items to stay in the player's hands, rely on ownership and attachment detection only and do not add the tag.

Batch Operations via Subsystem

The Scar Rewind Subsystem lets you control every registered Scar Rewind Component in the level at once. All components register automatically at Begin Play, you do not need to manage this manually.

Access it with [Get World Subsystem](#) and choose Scar Rewind Subsystem in the Class dropdown, or use the Blueprint library nodes from Section 8, which do not need a reference at all.

Function - Node	Description
Rewind All Actors	Call Start Rewind on every registered component. Each one routes itself, so on the server this starts an authoritative rewind for every one of them.
Stop All Rewinds	Call Stop Rewind on every registered component.
Clear All Buffers	Call Clear Buffer on every registered component.
Get Registered Component Count, returns int	How many Scar Rewind Components are currently active in the world.
Scar Server Rewind All Players, returns int	Library node. Rewinds only the player pawns, server-side, and reports how many started.

Batch operations are server work

Call these on the server. A client calling [Rewind All Actors](#) would send one refused request per Actor, because a client may only ever rewind a pawn it controls.

Security and Authority

A rewind moves a player's position and spends a resource, so both are owned by the server. Nothing here needs configuring; it is how the plugin behaves by default. This section is a description of the model so you can satisfy yourself it is sound.

What a client could try	What actually happens
Rewind another player's pawn	Refused. Every request arrives on a proxy Actor owned by that client's PlayerController, and the server checks the target is a pawn that controller actually possesses, or an Actor whose ownership chain leads back to it.
Claim to be somebody else when triggering a world rewind	Ignored. The server substitutes the pawn the requesting connection really owns before it records who the instigator was.
Rewind with no energy, or during cooldown	Refused. The server re-checks history, cooldown and energy itself. The client's own copy of those values is advisory only and is never trusted.
Fake a longer or faster rewind	Not possible. Start time, start cursor and playback speed all come from the server's replicated struct. A client that predicts locally is overwritten by those values the moment the server confirms.
Keep rewinding after the server says stop	Not possible. The server publishes the authoritative resting transform and the client blends onto it. A prediction the server never confirms is rolled back automatically after a short timeout.
Spam world rewinds	Governed by Allow Client Triggered Global Rewind and Min Time Between Global Rewinds on the subsystem. Turn the first off and clients cannot trigger one at all.
Send a malformed request packet	Rejected before it reaches gameplay code. The Server RPCs are validated, and an out-of-range duration is refused outright rather than silently clamped.

What the client does own

Only appearance. A client renders the rewind from its own recorded history, spawns its own ghost and effects, and may start showing a rewind a fraction of a second before the server has confirmed it. None of that is authoritative, and none of it can move the Actor as far as the rest of the session is concerned.

Energy and cooldown replicate only to the player who owns the Actor (**COND_OwnerOnly**), so they cost other connections no bandwidth and no other client can read them.

Locking it down completely

For a competitive game: set **Rewind Net Mode** to **Server Only** on every component and turn off **Allow Client Triggered Global Rewind**. Clients then have no path to start a rewind at all, and every rewind comes from your own server-side gameplay code.

Game Design Recipes

Ready-made configurations for common rewind gameplay patterns. All settings are applied in the Details panel of the Scar Rewind Component or via Blueprint nodes. No code is required.

Multiplayer Rewind Ability

Every player can rewind themselves on a cooldown, and everyone else can see it happen.

- Actor: tick Replicates
- Rewind Net Mode, Client Request (default)
- Use Rewind Energy, On - the server validates it, so it cannot be cheated
- Show Ghost Trail, On, with Show Ghost To Other Players left on
- Set Rewind Start Sound so nearby players hear it
- Wire input to [Start Rewind](#) and [Stop Rewind](#) - nothing else

Boss Mechanic - The World Rewinds

A boss reaching a phase threshold rewinds the whole arena for every player.

- On the server, from the boss Blueprint, call [Scar Server Start Global Rewind](#) with Duration 4.0 and the boss as Instigator
- Set Global Rewind FX Actor Class to a Niagara storm Blueprint
- Set Global Rewind Start Sound to a stinger
- Bind [On Global Rewind Net Event](#) in the HUD to show a banner naming the boss
- Consider turning Allow Client Triggered Global Rewind off so only the boss can do it

Competitive - Server Only Rewind

Rewind exists as a scripted mechanic and players can never trigger it themselves.

- Rewind Net Mode, Server Only on every component
- Allow Client Triggered Global Rewind, Off on the subsystem
- Drive everything from server-side gameplay with the [Scar Server](#) nodes

Death Rewind

Player dies, the world rolls back three seconds, and the ghost shows where they were.

- Use Rewind Energy, On
- Show Ghost Trail, On, Ghost Trail Linger Duration 2.0
- Use Post Process Effect, On, Desaturation 0.8

-
- Call `Scar Start Global Rewind` with Duration 3.0 from your death event

Puzzle Environment Reset

Every puzzle prop has a component. A mistake resets everything.

- Add Scar Rewind Component to every puzzle Actor
- Call `Rewind All Actors` on mistake, from the server
- Call `Clear All Buffers` on success to prevent rewinding past the clean state

Checkpoint Restore

Bookmarks are an instant teleport rather than a played-back rewind.

- Call `Save Bookmark` (Checkpoint) at each safe zone trigger
- Call `Restore Bookmark` (Checkpoint) from your death or respawn event
- In multiplayer, do both on the server so the result replicates

Resource-Limited Rewind

Four seconds per use, twenty seconds to recharge, two seconds between uses.

- Use Rewind Energy, On
- Max Rewind Energy 100, Energy Drain Rate 25, Energy Recharge Rate 5
- Rewind Cooldown, 2.0
- Bind `On Energy Changed` to a Progress Bar Set Percent
- Bind `On Rewind Request Denied` and feed it through `Scar Get Deny Reason Text` to explain refusals

2D Platformer - Side-Scroller Lock

Stops the rewind pulling the character off the 2D plane or flipping the sprite.

- Lock Location Y, On (locks the depth axis)
- Lock Rotation, On (keeps the sprite facing correctly)

Multiplayer Troubleshooting

Symptom	Cause and fix
Other players do not see the rewind at all	The Actor is not replicated. The rewind travels through the component, and a component cannot replicate if its Actor does not. Tick Replicates in Class Defaults. The plugin also logs this in plain English - filter the Output Log by LogScarRewind .
Nothing happens when a client presses rewind	Rewind Net Mode is set to Server Only. Set it to Client Request, or drive the rewind from server-side code.
A client's world rewind is ignored	Allow Client Triggered Global Rewind is off on the subsystem, or Min Time Between Global Rewinds has not elapsed. Bind On Rewind Request Denied to see the reason.
The rewind is jittery on remote clients	Check that Suspend Movement During Rewind and Suspend Movement Replication During Rewind are both on. With them off, the movement component and normal replication both fight the local playback.
The Actor snaps at the end of a rewind	Raise Reconciliation Blend Time . 0.15 suits most games; a high-latency session may want 0.25.
A remote player's rewind tints my screen	Post Process For Observers is on. The rewind post process is unbound, so it covers the whole screen. Turn it off to keep it local to the rewinding player.
Two ghosts appear for one rewind	The ghost Blueprint has Replicates ticked, so it is both replicated by the server and spawned locally. Untick it - cosmetics are always spawned locally.
Rewind is refused right after joining	There is no recorded history yet, or the per-player request proxy has not arrived. Both resolve within a second or two of joining. Can Start Rewind Now tells you which.
Rewind goes further back than expected	In 4.0 playback is time based, so Rewind Speed 1.0 is exactly real time. If you tuned a value against 3.0's behaviour, it may now travel a slightly different distance for an Actor that spent time standing still. Adjust Rewind Speed to taste.

Performance and Bandwidth

The Scar Rewind System is built with performance as a first priority. Here is what you get out of the box and what to watch for in large scenes.

Bandwidth

A rewind replicates one small struct when it starts and one when it stops - roughly thirty bytes each. Nothing is sent per frame, and the cost does not grow with the number of Actors being rewound. A world rewind that moves five hundred Actors costs the same as one that moves five.

Energy and cooldown replicate only to the player who owns the Actor, so they cost other connections nothing. Ghosts, markers, sounds and post process are spawned locally on each machine and never touch the network.

CPU

Tick Disabled When Idle

The component enables Tick only during active rewind, reconciliation, energy management or debug display. A stationary Actor with the component attached and no active features has zero per-frame cost.

Timer-Based Recording

Recording uses a looping timer, not a per-frame Tick. Snapshots are taken at your configured Recording Interval regardless of frame rate.

Zero Runtime Allocations

The circular buffer is pre-allocated once in Begin Play. Pushing a snapshot overwrites existing memory in place. No garbage collection pressure during play.

Movement Threshold Skips Idle Actors

Actors that have not moved or rotated beyond the threshold values do not write a snapshot on that recording cycle. Stationary Actors consume no buffer space.

Binary Search Sampling

Finding the right moment in history is a binary search over timestamps, so it costs the same handful of steps whether the buffer holds fifty snapshots or five thousand.

FTickableGameObject Subsystem

The subsystem uses FTickableGameObject, not UActorComponent Tick. This avoids being blocked by ActorComponent tick prerequisites and runs more predictably across versions.

Global Rewind at Scale

The global recording iterator runs at Global Record Interval, not every frame. In scenes with 200 or more dynamic Actors, raise the interval to 0.1 and tag non-essential background props with `ScarIgnoreGlobal`.

Testing and Compatibility

Engine versions

Every released package is installed into a clean project and compiled before it ships. Version 4.0 builds with zero errors and zero C++ warnings on:

Engine	Result
UE 5.2	Builds clean
UE 5.3	Builds clean
UE 5.4	Not verified on the build machine. No API used by the plugin differs between 5.3 and 5.5, so it is expected to build, but it has not been tested.
UE 5.5	Builds clean
UE 5.6	Builds clean
UE 5.7	Builds clean

Multiplayer verification

The networking is not only compiled, it is run. A headless dedicated server and a separate connecting client, over a real network driver, execute an automated scenario covering a server-triggered rewind, a world rewind, and a client-requested rewind.

What was measured	Result
Client mirrors every server-side rewind, in step	Worst difference at a rewind's start or end edge: 0.107 seconds .
A world rewind reaches the client	Identical window on both ends.
A client-initiated request is accepted by the server	Round trip completed; the server rewound the requested Actor.
A watching client replays another Actor's history smoothly	The Actor travelled back 734 cm across 50 distinct positions with no forward jumps . A teleport would have produced one or two positions.
Ordinary movement replication resumes afterwards	The Actor continued moving normally, a further 3773 cm.

What was measured	Result
Client and server agree on the final position	10.3 cm apart after reconciliation.
Stability	No errors or warnings from the plugin, and neither process crashed.

What that fourth row means for your game

It is the difference between a rewind reading as a feature and reading as a bug. Other players watch the rewinding Actor sweep back along the path it took, at their own framerate, rather than snapping to a new position.

What the testing does not cover

Stated plainly so you know where to put your own testing effort:

- **Character Movement Component interaction.** The automated scenario uses a simple pawn and a simple Actor, so the code that suspends a Character Movement Component during a rewind is not exercised by it. Test this in your own project if your characters use Character Movement, which most do.
- **High player counts.** The scenario runs one server and one client.
- **Real world latency.** Everything runs on one machine, so round trip time is effectively zero. The design does not accumulate drift, because the playback cursor is a function of synchronised time rather than a running total, but that is reasoning rather than a measurement.

Version 3.0 Crash Fixes

Kept from the 3.0 guide for transparency. Version 2.0 compiled without errors but crashed Unreal Engine 5.2 at runtime. Eight root causes were identified and resolved, and all of those fixes are still in place in 4.0.

Fix	What was wrong, and what changed
1 - Raw AActor* map key	The global rewind cache keyed on a raw pointer, which dangled when an Actor was destroyed mid-session. Now keyed by FObjectKey with a TWeakObjectPtr inside the record, which is GC-safe and never dangling.
2 - Missing engine delegate	FWorldDelegates::OnWorldPreActorTick does not exist in UE 5.2. The subsystem now inherits FTickableGameObject and implements Tick, which is stable across all UE 5.x.
3 - Iterating actors during world transitions	The recording timer fired regardless of world state, so it could iterate a partially destroyed world. Now it accumulates delta time inside Tick and is guarded by WorldType checks.
4 - NewObject without proper attachment	The post process component was never attached to the owner's scene hierarchy, so PIE cleanup asserted. Now created with the owner as outer, then attached and registered in the correct order with a unique generated name.
5 - Ghost spawned as a bare engine Actor	The old ghost trail spawned a bare AActor or AStaticMeshActor and configured it from C++, which was brittle and inflexible. Replaced entirely by a TSubclassOf Blueprint class picker, so you assign any Blueprint Actor as the ghost with full creative control and no engine-internal assumptions.
6 - Deinitialize calling GetWorld on a null world	During editor shutdown and level streaming, GetWorld can return nullptr in Deinitialize, and the old code reached the timer manager through it. All GetWorld calls were removed from Deinitialize; cleanup now only resets flags and empties arrays, which is safe without a valid world.
7 - Cache iteration with invalidated keys	Related to Fix 1. Raw pointer keys could go stale between recording ticks. An explicit stale-entry purge pass now runs after each recording cycle, using TWeakObjectPtr validity checks.

Fix	What was wrong, and what changed
8 - Re-entrant map modification	StopGlobalRewind was called from inside the loop iterating the Actor cache, modifying the map mid-iteration and corrupting the hash table. Now every stop sets a deferred flag that Tick processes after the loop has finished.

File Reference

All plugin source files are in your project's `Plugins/ScarRewindSystem/` folder. You do not need to edit any of these files to use the plugin. This reference is for developers who want to extend the system.

File	Purpose
<code>ScarRewindSystem.uplugin</code>	Plugin descriptor, version info, and Fab marketplace metadata.
<code>ScarRewindSystem.Build.cs</code>	Module build rules. <code>PhysicsCore</code> , <code>RenderCore</code> and <code>NetCore</code> .
<code>ScarRewindSystemModule.h / .cpp</code>	Module startup and shutdown, and the <code>LogScarRewind</code> log category.
<code>ScarRewindSnapshot.h</code>	<code>FScarRewindSnapshot</code> struct. Edit <code>SCAR_MAX_CUSTOM_VARS</code> here.
<code>ScarCircularBuffer.h</code>	<code>TScarCircularBuffer</code> template. Allocation-free ring buffer.
<code>ScarRewindTypes.h</code>	Shared delegates, the <code>EScarRewindState</code> enum, and <code>FScarRewindBookmark</code> .
<code>ScarRewindNetTypes.h</code>	New in 4.0. The replicated state structs, the net mode and deny reason enums, and a full written explanation of the replication design.
<code>ScarRewindNetRelay.h / .cpp</code>	New in 4.0. The always-relevant relay Actor that carries global rewind state to every client, and the per-player proxy that lets a client send a Server RPC. Both are spawned and managed by the subsystem - you never place them.
<code>ScarRewindBlueprintLibrary.h / .cpp</code>	New in 4.0. The 26 standalone Blueprint nodes.
<code>ScarRewindComponent.h / .cpp</code>	Main component. Per-actor rewind, replication, client prediction and reconciliation, ghost trail, rewind cues, energy, bookmarks, axis lock, post process.
<code>ScarRewindSubsystem.h / .cpp</code>	World Subsystem. Batch operations, global world rewind and its replication, pickup exclusion, global visuals, <code>FTickableGameObject</code> ticking.