PS_Editor

Runtime Scene Editor Plugin

User Documentation

ASTERION | Unreal Engine 5.5

01

What is PS_Editor?

A runtime scene editor for Unreal Engine 5.5

PS_Editor enables your team to place, configure and serialize actors directly inside packaged builds, without needing the Unreal Editor. Perfect for scenario design, training simulations and level dressing workflows.

Runtime Editing

Full object placement, transform gizmos and property editing in packaged builds.

Scene Serialization

Save and load scenarios to JSON files. Versioned, mergeable, human-readable.

Spline System

Place and edit splines at runtime with handles, insert points, drag and drop.

AI Integration

Bridge plugin for AI behavior splines, auto-detected by the SplineNetwork.

Typical use cases

- Training and simulation scenario authoring
- In-game level editors for designers and end users
- Rapid iteration without opening the Unreal Editor
- Procedural content validation by non-technical teams

02 Plugin Architecture

PS_Editor

Main plugin (Runtime)

- PlayerController + Pawn
- SelectionManager + Gizmo
- SpawnManager + Catalog
- SceneSerializer (save)
- SceneLoader (gameplay API)
- SplineActor + interface
- UndoManager
- Main UMG widget

PS_EditorTools

Editor-only companion

- Factory for SpawnCatalog DataAsset
- Content Browser extension
- Create Spawnable Blueprint from mesh (right-click)
- Thumbnail capture utility
- Editor-only (WITH_EDITOR)

PS_BehaviorEditor

Bridge plugin (optional)

- Depends on PS_Editor + PS_AI_Behavior
- APS_BehaviorEditor_AISpline
- Detected by SplineNetwork
- Editable via PS_Editor UI
- Triggers RebuildNetwork on change
- EnabledByDefault = false

03 Main Features

Object placement

Click on catalog entry, drag in scene, snap to ground on End key.

Transform gizmo

Translate / Rotate / Scale with keyboard shortcuts Z / E / R.

Selection

Click to select, Ctrl+click for multi-selection, outline highlight.

Property editing

Transform + custom properties via EditableComponent configuration.

Splines at runtime

Place control points, insert mid-segment, drag handles, delete point.

AI splines

APS_BehaviorEditor_AISpline integrates with SplineNetwork automatically.

JSON scene save/load

Save scenarios with Save/Save As, load with Browse or from list.

Base Level selection

Thumbnail grid popup to choose the background sublevel.

Undo / Redo

Ctrl+Z / Ctrl+Y on transform, property, spawn, delete, spline actions.

Simulate mode

Start/stop AI logic, auto-restore character positions on stop.

Play / Return

Open gameplay map with scenario, come back to editor keeping state.

Network ready

Auto-call OnRep + ForceNetUpdate after property changes.

Setup steps

1 Enable plugins

In .uproject, enable PS_Editor (and optionally PS_BehaviorEditor if using AI splines).

2 Create MasterCatalog

Right-click in Content Browser -> Data Asset -> PS_Editor_MasterCatalog. Add sub-catalogs and base levels.

3 Configure GameMode

Create a map with PS_Editor_GameMode. Assign PS_Editor_PlayerController and PS_Editor_Pawn.

4 Assign MasterCatalog

On PlayerController: set MasterCatalogAsset to your MasterCatalog.

5 Prepare spawnable BPs

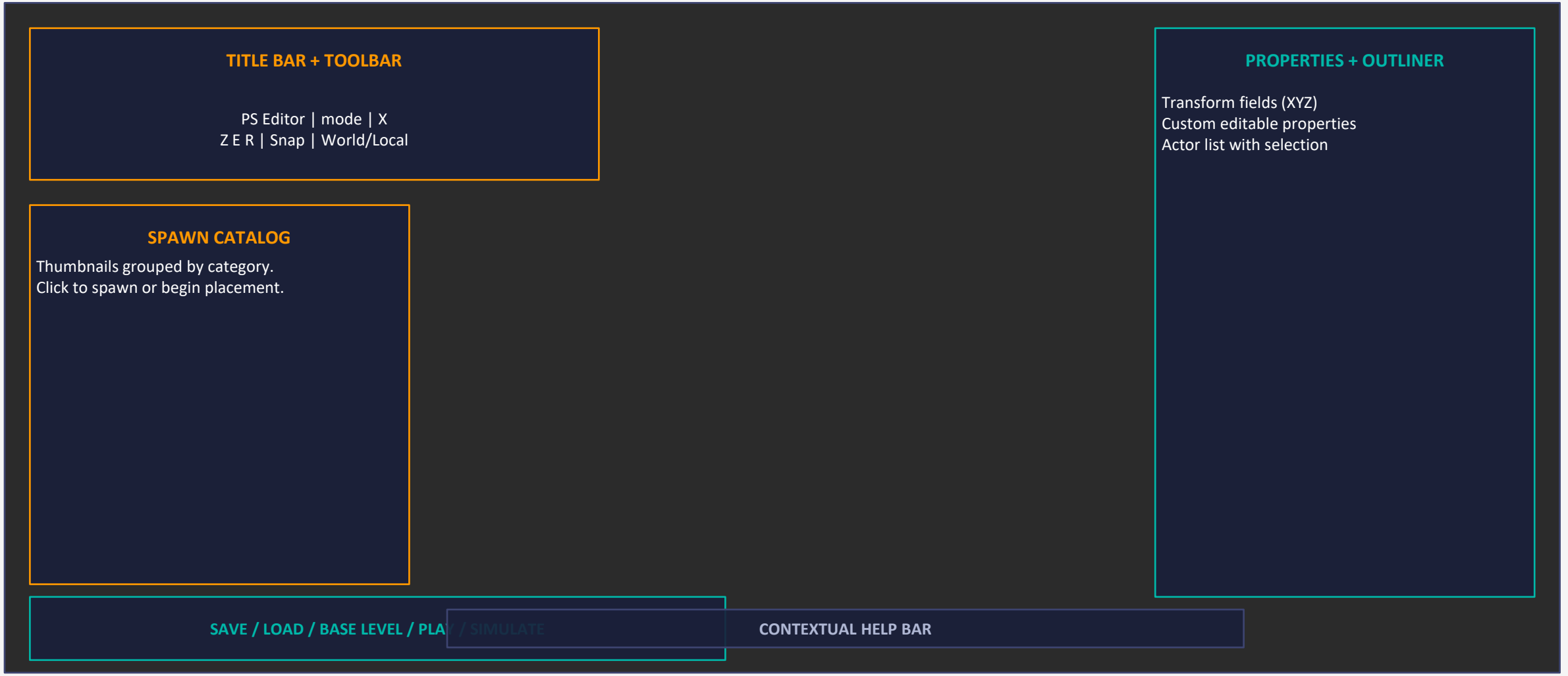
Add PS_Editor_Spawnable + PS_Editor_Editable components to your actor BPs.

6 Launch

Play the editor map. The runtime editor UI appears automatically via HUD.

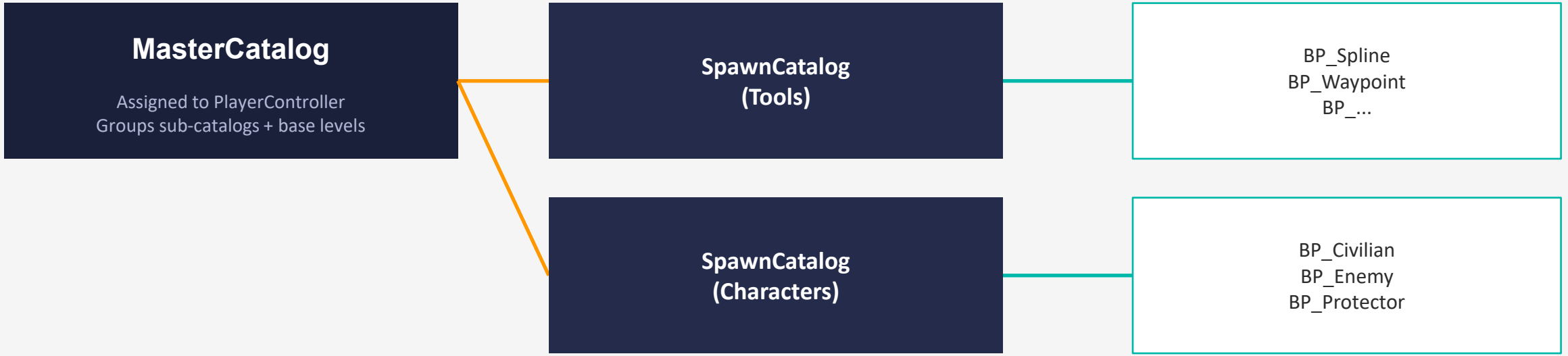
05 Editor UI Overview

The runtime editor is a Slate widget overlay with five main zones



06 Spawn Catalog

Two-level DataAsset hierarchy



Tip: Right-click a StaticMesh in the Content Browser

PS_EditorTools adds a "Create Spawnable Blueprint" entry that auto-generates a Blueprint with PS_Editor_Spawnable + PS_Editor_Editable components already configured. Perfect for bulk-populating your catalog from existing meshes.

07 Selection & Transform Gizmo

Selection

- Left click on an actor to select it
- Ctrl+click to add/remove from selection
- Click in empty space to deselect all
- Selected actors get a custom-depth outline
- Outliner shows all spawned actors
- Only actors tracked by SpawnManager are selectable

Transform modes

- Z = Translate (drag arrows / planes)
- E = Rotate (drag rings)
- R = Scale (drag handles)
- World / Local space toggle
- Snap to grid (translation, rotation, scale)
- End = Snap to ground beneath the selection
- Ctrl+D or Ctrl+C/V for duplicate / copy / paste

Custom runtime gizmo - implemented with BasicShapes + unlit material

Unreal's InteractiveToolsFramework is editor-only, so PS_Editor ships its own gizmo actor that works in packaged builds. Highlight color on hover, click and drag axis constraints, rotation delta accumulation, etc.

08 Property Editing

Expose any UPROPERTY to runtime editing

Add the PS_Editor_Editable component to your actor Blueprint and list the properties you want editable in its EditableProperties array. Supports: float, int, bool, FString, enums (dropdown), FVector, FRotator, FLinearColor, and any UPROPERTY exposed on the actor or its components.

1. Setup in BP (design time)

- Add PS_Editor_Editable component
- Expand EditableProperties array
- Select a component (dropdown)
- Select a property (filtered list)
- Flags: bNeedsReinit, bNeedsRespawn
- Path stored as Component.Property

2. Runtime editing

- Auto-generated UI per property type
- Transform fields always shown
- Import/export via FProperty reflection
- OnEditorPropertyChanged event fires
- OnEditorNeedsReinit if flag set
- OnEditorNeedsRespawn if flag set
- Auto-call OnRep + ForceNetUpdate

Generic runtime spline editing

APS_Editor_SplineActor provides a full spline authoring system: click to place control points, handles to drag them, insert mid-segment by clicking the tube, delete with Del key. Uses USplineComponent internally, visualized as a ProceduralMesh tube.

Start

Click the Spline entry in the catalog. Editor enters SplinePlacement mode.

Place

Left-click in the world to add control points. Snap to ground automatic.

Edit

Click a handle to select it, then drag with the gizmo to reposition.

Insert

Click anywhere on the tube to insert a new point at that location.

Delete

Select a handle + Del key to remove a single point.

Finish

Escape or the Done button to return to Normal mode.

IPS_Editor_SplineEditable interface

All spline editing code targets this interface rather than a concrete class. Any actor implementing IPS_Editor_SplineEditable (plus IPS_Editor_PointPlaceable for the placement flow) gets full editor support: handles, drag, insert, delete, undo, serialization. The bridge plugin's AISpline reuses this infrastructure with zero modification to PS_Editor.

AI Splines (Bridge Plugin)

Combine PS_Editor with PS_AI_Behavior via PS_BehaviorEditor

APS_BehaviorEditor_AISpline inherits from APS_AI_Behavior_SplinePath (detected by SplineNetwork subsystem) and implements IPS_Editor_SplineEditable. NPCs automatically follow splines placed in the editor. No dependency created between the two plugins - the bridge is a third plugin that depends on both.

AI-specific properties

- SplineCategory: Civilian / Enemy / Protector / Any
- bBidirectional: travel both directions
- SplineWalkSpeed: override NPC default speed
- Priority: selection weight at junctions
- All editable at runtime via the editor UI

Automatic integration

- SplineNetwork rebuilt on spline change (debounced 0.5s)
- BTTask_FindAndFollowSpline works out of the box
- Junction detection between splines automatic
- No changes to existing BTs required
- Create variants (Enemy/Civilian/Any) as BP children

```
APS_AI_Behavior_SplinePath    =>    APS_BehaviorEditor_AISpline    =>    your BP (Red / Green / Blue variants)
```

Scene Save & Load

Save scenarios

- Save: type a name, click Save button
- Save As: native Windows file dialog
- Stored in Saved/PS_Editor/Scenes/ as JSON
- Includes transforms, custom properties, spline data
- LevelName stored -> scenario tied to a base level
- Versioned schema (SceneData v3)

Load scenarios

- Click an entry in the saved scenes list
- Browse: native Windows dialog
- Scenes filtered by current BaseLevel
- Auto-switch BaseLevel if scenario specifies one
- OnPropertiesLoaded event on each spawned actor
- IsLoadingFromScene() true during load

Sample scenario JSON (excerpt)


```
{ "SceneName": "Warehouse_01", "LevelName": "Warehouse", "Actors": [ { "ClassPath": "/Game/BP_Civilian.BP_Civilian_C", "Location": { "X": 120.0,
"Y": 340.0, "Z": 0.0 }, "CustomProperties": { "CharacterType": "Civilian" } } ] }
```

12 Base Level Selection

Choose a background sublevel

The editor can stream any base map as a sublevel for visual context. Configure your base levels in the MasterCatalog DataAsset (display name + map path + thumbnail). A popup with a thumbnail grid lets you pick visually.

1 Configure

Fill the BaseLevels array in MasterCatalog. Each entry: DisplayName, MapPath, Thumbnail.

2 Click button

In the editor UI, click the BaseLevel button (bottom-left panel).

3 Select

A popup with a 3-column thumbnail grid opens. Click an entry.

4 Validate

Click Validate to load. Cancel to dismiss. 'None' unloads all.

5 Stream

Sublevel loads via LoadLevelInstance. Fade-in + loading overlay.

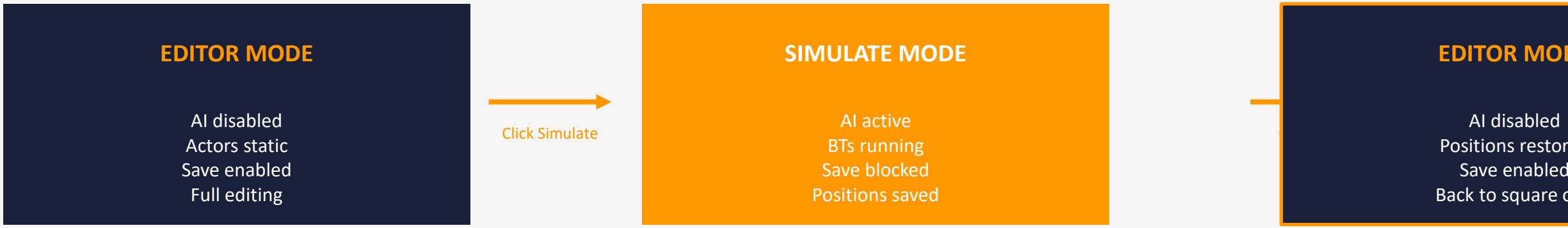
6 Filter

Saved scenes list filters automatically by current BaseLevel.

Simulate Mode

Preview AI behavior without leaving the editor

In Simulate mode, PS_Editor enables AI on all spawned pawns so you can verify behavior without a full Play cycle. When you stop, every actor returns to its original placement, letting you iterate quickly.



Under the hood

PS_Editor_PlayerController::StartSimulation captures every tracked actor's FTransform, enables ticking on each Pawn + AIController, and calls BrainComponent::RestartLogic. StopSimulation reverses everything. AI-heavy features (perception, gaze, movement) are all paused via tick disable, not just the BT.

14 Play & Return to Editor

Seamless transition from authoring to gameplay

Click Play to open the gameplay map with the current scenario loaded. PS_Editor_GameInstanceSubsystem persists scenario/base level across the OpenLevel, and provides ReturnToEditor to come back with your state intact.

1. Editor map (PS_Editor_GameMode)
2. Click Play -> auto-save, set PendingScenario on subsystem
3. OpenLevel to BaseLevel map (your GameMode)
4. Your GameMode::BeginPlay calls SceneLoader::LoadScene
5. Gameplay runs with your PlayerController, AI, etc.
6. Call Subsystem::ReturnToEditor(world) from your code
7. OpenLevel back to editor map, state restored

Blueprint Integration

Six lifecycle events via IPS_Editor_EditableInterface

`OnEditorPropertyChanged`

Fires after a property is modified in the editor UI.

Trigger: Every property change

`OnEditorNeedsReinit`

Property marked bNeedsReinit was changed. Re-run init logic.

Trigger: bNeedsReinit flag

`OnEditorNeedsRespawn`

Called on the NEW actor after a respawn. All props applied.

Trigger: bNeedsRespawn flag

`OnPropertiesLoaded`

All properties applied from JSON load. Server-side only.

Trigger: Scene load

`OnEditorModeChanged`

Entering (true) or leaving (false) editor mode.

Trigger: Editor <-> Play

`OnEditorTick`

Per-frame update while in editor mode.

Trigger: Editor tick

Helper: IsLoadingFromScene(Actor)

In BeginPlay, call UPS_Editor_SceneLoader::IsLoadingFromScene(self). If true, skip default init and wait for OnPropertiesLoaded - the properties haven't been applied yet. This avoids double-initialization when loading a scenario.

Automatic replication after property changes

ImportText_InContainer bypasses UE's standard property setter path, so replication and OnRep don't fire automatically. PS_Editor handles this for you.

SERVER

- ImportText applies value
- OnEditorPropertyChanged fires
- RepNotifyFunc found via reflection
- OnRep_X called on server
- ForceNetUpdate scheduled

REPLICATION

- Property value replicated to clients
- Standard UE lifetime replication
- No special handling needed
- Works with push model
- Respects COND_* conditions

CLIENT

- Replicated value arrives
- OnRep_X fires automatically
- Visual / logic update
- No PS_Editor code runs client-side
- Behaves like any standard actor

Result: edit once, replicate everywhere. Mark your properties as Replicated or ReplicatedUsing in your BP - PS_Editor handles the rest transparently.

Keyboard Shortcuts

AZERTY-optimized defaults

`Z / E / R`	Translate / Rotate / Scale mode
`Arrow keys`	Pan the camera
`A / Q`	Move camera up / down
`Right-click drag`	Free look (fly mode)
`Mouse wheel`	Zoom / adjust fly speed
`Alt + left-click`	Orbit around focal point
`End`	Snap selection to ground
`Del`	Delete selected actor or spline point
`Ctrl+Z / Ctrl+Y`	Undo / Redo
`Ctrl+D`	Duplicate selection
`Ctrl+C / Ctrl+V`	Copy / Paste
`Esc`	Cancel spline placement

Best Practices

Avoid widget weak ptrs in lambdas	In packaged builds, HUD widget pointers can be stale. Always fetch SpawnManager / SceneSerializer fresh from PlayerController via GetOwningPlayer().
Use hard references (TObjectPtr / TSubclassOf)	TSoftObjectPtr and TSoftClassPtr are not cooked into packaged builds. Stick to hard references for actor classes in catalogs.
Keep UFactory code in PS_EditorTools	UHT cannot resolve UFactory parent outside WITH_EDITOR. All editor-only factories live in the dedicated PS_EditorTools module.
Mark properties Replicated for multiplayer	PS_Editor calls OnRep automatically, but the replication system needs your UPROPERTY marked Replicated / ReplicatedUsing.
Prefer Live Coding when UE is open	Ctrl+Alt+F11 for cpp-only changes. CLI build only when UE is closed or when headers change.
Test in packaged builds regularly	Some timing issues (HUD deferred init, TWeakObjectPtr behavior) only appear in packaged. Don't trust PIE alone.

Thank you

Build scenes at runtime. Iterate faster. Ship scenarios without recompile.

PS_Editor | PS_EditorTools | PS_BehaviorEditor

ASTERION - Unreal Engine 5.5