FoxEasySave

Slot-Based Key-Value Save System for Blueprints

User Guide • Version 1.0

Alchemy Fox Studio

Supported Engine: UE 5.2, 5.3, 5.4, 5.6, 5.7

Overview

FoxEasySave is a slot-based key-value save system for Blueprints. Store and read typed values by name — no USaveGame subclass, no boilerplate — with auto-versioning, migration delegates and a player-profile pattern built in.

Installation

- **Fab Library:** add the plugin to your account, then install to the engine from the Epic Games Launcher.
- **Manual:** copy the plugin folder into your project's *Plugins/* directory and restart the editor.
- All nodes appear under **FoxEasySave** when you right-click in any Blueprint graph.

Quick Start

- Call a Save<Type> node with a Slot Name and Key to write a value (the slot is auto-created).
- Call Save Slot to commit the in-memory slot to disk.
- Call the matching Load<Type> node with the same Slot Name and Key to read it back.
- Use Set Active Profile + Save Active Profile To Disk for multi-profile games.

Examples

Example 1: Save and Load a Coin Count

Goal: Persist the player's coins to disk and read them back on the next launch.

Write the Value
When coins change, call **Save Int** with Slot Name "MainSave", Key "Coins" and your current coin value. This updates the in-memory slot.

Commit to Disk
Call **Save Slot** with Slot Name "MainSave" to write the slot to disk. Do this at a checkpoint or on quit — not every frame.

Read It Back
On game start, call **Load Int** with the same Slot Name and Key and a Default of 0. On a fresh install it safely returns 0 — no crash, no null check.

Play and Test
Earn coins, quit, relaunch — the saved total is restored. Delete the slot with **Delete Slot** to simulate a new player.

Example 2: A Settings Menu That Remembers

Goal: Store mixed-type options (volume, language, a toggle) and restore them when the menu opens.

Save Each Option
On Apply, call **Save Float** ("Volume"), **Save String** ("Language") and **Save Bool** ("Fullscreen") into a "Settings" slot, then **Save Slot**.

Restore on Open
When the settings widget constructs, call the matching **Load Float/String/Bool** with sensible defaults and push the values into your sliders and dropdowns.

Check Before Reading (optional)
Use **Has Value** to detect a first run and apply factory defaults, or **Get All Keys** to iterate everything stored in the slot.

Play and Test
Change settings, reopen the menu — every control shows the last saved value, across sessions.

Example 3: Multiple Save Profiles

Goal: Let the player pick between independent save profiles, each with its own data.

Select a Profile

When the player chooses a profile, call **Set Active Profile** ("Adventure 1"). All later saves can target **Get Active Profile** instead of a hard-coded name.

Save Into the Active Profile

Call your **Save*** nodes using **Get Active Profile** as the Slot Name, then **Save Active Profile To Disk** to commit just that profile.

Switch Without Losing Data

Use **Switch Active Profile** to save the current profile and switch to another in one call. Each profile's slot stays separate on disk.

List and Manage

Call **Get All Slot Names** to populate a profile-select screen, and **Duplicate Slot** to offer a "copy save" option.

Node Reference

Complete reference for all 37 FoxEasySave nodes organized by category. Each node includes its inputs, outputs, and a use-case hint.

1. Slot Management (6 Nodes)

1.1 Create Slot

Provides 1.1 Create Slot functionality.

Input	Type	Description
World Context	Object	World context
Slot Name	Name	Unique slot identifier
Output	**Type**	**Description**
Return Value	Boolean	True on success; false if the slot already exists

Use Case: Initialise a fresh slot before writing values — Save* setters also auto-create, so this is optional in the common case.

1.2 Delete Slot

Provides 1.2 Delete Slot functionality.

Input	Type	Description
World Context	Object	World context
Slot Name	Name	Slot to delete
Output	**Type**	**Description**
Return Value	Boolean	True on success

Use Case: "Delete Save Game" UI button.

1.3 Does Slot Exist

Provides 1.3 Does Slot Exist functionality.

Input	Type	Description
World Context	Object	World context
Slot Name	Name	Slot to check
Output	**Type**	**Description**
Return Value	Boolean	True if loaded in memory OR present as a save file on disk

Use Case: "Continue Game" button should be disabled if no save exists.

1.4 Get All Slot Names

Provides 1.4 Get All Slot Names functionality.

Input	Type	Description
World Context	Object	World context
Output	**Type**	**Description**
Return Value	Array of Name	All known slot names

Use Case: Populate the "Load Game" UI list.

1.5 Get Slot Info

Provides 1.5 Get Slot Info functionality.

Input	Type	Description
World Context	Object	World context
Slot Name	Name	Slot to query
Output	**Type**	**Description**
Return Value	FFoxSlotInfo	Slot Name / Last Saved Time / Version / Estimated Size Bytes / Value Count

Use Case: Display "Last Played: 2 hours ago" + "47 values" in load menu rows.

1.6 Duplicate Slot

Provides 1.6 Duplicate Slot functionality.

Input	Type	Description
World Context	Object	World context
Source Slot	Name	Existing slot to copy from
Target Slot	Name	New slot's name (must not already exist)
Output	**Type**	**Description**
Return Value	Boolean	True on success

Use Case: "Save As New" — duplicates current state to a new slot, then keep playing the original.

2. Save Values — Setters (8 Nodes)

2.1 Save String

Provides 2.1 Save String functionality.

Input	Type	Description
World Context	Object	World context
Slot Name	Name	Target slot (created if missing)
Key	Name	Value key (case-sensitive)
Value	String	Value to store — see per-node Type below
Output	**Type**	**Description**
Return Value	Boolean	True on success

Use Case: Player name, language code, item ID.

2.2 Save Int

Provides 2.2 Save Int functionality.

Input	Type	Description
World Context	Object	World context
Slot Name	Name	Target slot (created if missing)
Key	Name	Value key (case-sensitive)
Value	Integer	Value to store — see per-node Type below
Output	**Type**	**Description**
Return Value	Boolean	True on success

Use Case: Coin count, level number, score.

2.3 Save Float

Provides 2.3 Save Float functionality.

Input	Type	Description
World Context	Object	World context
Slot Name	Name	Target slot (created if missing)
Key	Name	Value key (case-sensitive)
Value	Float	Value to store — see per-node Type below
Output	**Type**	**Description**
Return Value	Boolean	True on success

Use Case: Volume, time played, percentages.

2.4 Save Bool

Provides 2.4 Save Bool functionality.

Input	Type	Description
World Context	Object	World context
Slot Name	Name	Target slot (created if missing)
Key	Name	Value key (case-sensitive)
Value	Boolean	Value to store — see per-node Type below
Output	**Type**	**Description**
Return Value	Boolean	True on success

Use Case: Unlocked, settings flags.

2.5 Save Vector

Provides 2.5 Save Vector functionality.

Input	Type	Description
World Context	Object	World context
Slot Name	Name	Target slot (created if missing)
Key	Name	Value key (case-sensitive)
Value	Vector	Value to store — see per-node Type below
Output	**Type**	**Description**
Return Value	Boolean	True on success

Use Case: Last position, last spawn point.

2.6 Save Transform

Provides 2.6 Save Transform functionality.

Input	Type	Description
World Context	Object	World context
Slot Name	Name	Target slot (created if missing)
Key	Name	Value key (case-sensitive)
Value	Transform	Value to store — see per-node Type below
Output	**Type**	**Description**
Return Value	Boolean	True on success

Use Case: Spawn transform, last full transform.

2.7 Save Object

Provides 2.7 Save Object functionality.

Input	Type	Description
World Context	Object	World context
Slot Name	Name	Target slot (created if missing)
Key	Name	Value key (case-sensitive)
Value	Soft Object Reference	Value to store — see per-node Type below
Output	**Type**	**Description**
Return Value	Boolean	True on success

Use Case: Asset references (no load).

2.8 Save Tag

Provides 2.8 Save Tag functionality.

Input	Type	Description
World Context	Object	World context
Slot Name	Name	Target slot (created if missing)
Key	Name	Value key (case-sensitive)
Value	GameplayTag	Value to store — see per-node Type below
Output	**Type**	**Description**
Return Value	Boolean	True on success

Use Case: Last selected category, current class tag.

3. Load Values — Getters (8 Nodes)

3.1 Load String

Provides 3.1 Load String functionality.

Input	Type	Description
World Context	Object	World context
Slot Name	Name	Slot to read from
Key	Name	Value key
Default Value	String	Returned when slot/key missing or type mismatch
Output	**Type**	**Description**
Return Value	String	Stored value or Default

Use Case: Read back a saved player name, language code or item ID on game start.

3.2 Load Int

Provides 3.2 Load Int functionality.

Input	Type	Description
World Context	Object	World context
Slot Name	Name	Slot to read from
Key	Name	Value key
Default Value	Integer	Returned when slot/key missing or type mismatch
Output	**Type**	**Description**
Return Value	Integer	Stored value or Default

Use Case: Restore the player's coin count, current level or high score when a slot loads.

3.3 Load Float

Provides 3.3 Load Float functionality.

Input	Type	Description
World Context	Object	World context
Slot Name	Name	Slot to read from
Key	Name	Value key
Default Value	Float	Returned when slot/key missing or type mismatch
Output	**Type**	**Description**
Return Value	Float	Stored value or Default

Use Case: Re-apply a saved volume, total play-time or completion percentage.

3.4 Load Bool

Provides 3.4 Load Bool functionality.

Input	Type	Description
World Context	Object	World context
Slot Name	Name	Slot to read from
Key	Name	Value key
Default Value	Boolean	Returned when slot/key missing or type mismatch
Output	**Type**	**Description**
Return Value	Boolean	Stored value or Default

Use Case: Check a stored unlock or settings flag to gate content on load.

3.5 Load Vector

Provides 3.5 Load Vector functionality.

Input	Type	Description
World Context	Object	World context
Slot Name	Name	Slot to read from
Key	Name	Value key
Default Value	Vector	Returned when slot/key missing or type mismatch
Output	**Type**	**Description**
Return Value	Vector	Stored value or Default

Use Case: Restore the player's last position or a saved spawn point.

3.6 Load Transform

Provides 3.6 Load Transform functionality.

Input	Type	Description
World Context	Object	World context
Slot Name	Name	Slot to read from
Key	Name	Value key
Default Value	Transform	Returned when slot/key missing or type mismatch
Output	**Type**	**Description**
Return Value	Transform	Stored value or Default

Use Case: Respawn an actor at its saved transform after loading a slot.

3.7 Load Object

Provides 3.7 Load Object functionality.

Input	Type	Description
World Context	Object	World context
Slot Name	Name	Slot to read from
Key	Name	Value key
Default Value	UObject (synchronously loaded)	Returned when slot/key missing or type mismatch
Output	**Type**	**Description**
Return Value	UObject (synchronously loaded)	Stored value or Default

Use Case: Resolve a saved soft asset reference and load the asset on demand.

3.8 Load Tag

Provides 3.8 Load Tag functionality.

Input	Type	Description
World Context	Object	World context
Slot Name	Name	Slot to read from
Key	Name	Value key
Default Value	GameplayTag	Returned when slot/key missing or type mismatch
Output	**Type**	**Description**
Return Value	GameplayTag	Stored value or Default

Use Case: Restore the last selected category or the player's current class tag.

4. Value Meta (4 Nodes)

4.1 Has Value

Provides 4.1 Has Value functionality.

Input	Type	Description
World Context	Object	World context
Slot Name	Name	Target slot
Key	Name	Value key
Output	**Type**	**Description**
Return Value	Boolean	True if the key is present

Use Case: First-launch detection, "user has explicitly set this" checks.

4.2 Remove Value

Provides 4.2 Remove Value functionality.

Input	Type	Description
World Context	Object	World context
Slot Name	Name	Target slot
Key	Name	Key to remove
Output	**Type**	**Description**
Return Value	Boolean	True if removed; false if the key didn't exist

Use Case: Cleaning up removed-feature save data.

4.3 Get Value Type

Provides 4.3 Get Value Type functionality.

Input	Type	Description
World Context	Object	World context
Slot Name	Name	Target slot
Key	Name	Value key
Output	**Type**	**Description**
Return Value	EFoxSaveValueType	None / String / Int / Float / Bool / Vector / Transform / ObjectRef / Tag

Use Case: Generic save-inspector UIs that switch behaviour per type.

4.4 Get All Keys

Provides 4.4 Get All Keys functionality.

Input	Type	Description
World Context	Object	World context
Slot Name	Name	Slot to enumerate
Output	**Type**	**Description**
Return Value	Array of Name	All keys; empty for unknown slots

Use Case: Save-game debugging, migration helpers.

5. Disk Operations (4 Nodes)

5.1 Save Slot

Provides 5.1 Save Slot functionality.

Input	Type	Description
World Context	Object	World context
Slot Name	Name	Slot to write
Output	**Type**	**Description**
Return Value	Boolean	True on successful disk write

Use Case: Commit in-memory writes. Standard "save on quit" pattern.

5.2 Load Slot

Provides 5.2 Load Slot functionality.

Input	Type	Description
World Context	Object	World context
Slot Name	Name	Slot to load
Output	**Type**	**Description**
Return Value	Boolean	True on successful load

Use Case: Restore a slot at game start. False = first-play case.

5.3 Save Slot Async (Latent)

Provides 5.3 Save Slot Async (Latent) functionality.

Input	Type	Description
World Context	Object	World context
Slot Name	Name	Slot to write
Latent Info	FLatentActionInfo	Auto-filled by the engine
Output	**Type**	**Description**
bSuccess	Boolean	True if the disk write succeeded

Use Case: Background autosave loops, big slots during gameplay.

5.4 Load Slot Async (Latent)

Provides 5.4 Load Slot Async (Latent) functionality.

Input	Type	Description
World Context	Object	World context
Slot Name	Name	Slot to load
Latent Info	FLatentActionInfo	Auto-filled by the engine
Output	**Type**	**Description**
bSuccess	Boolean	True on successful load

Use Case: Background load during a loading screen while another scene is being prepared.

6. Versioning & Migration (3 Nodes)

6.1 Get Slot Version

Provides 6.1 Get Slot Version functionality.

Input	Type	Description
World Context	Object	World context
Slot Name	Name	Slot to query
Output	**Type**	**Description**
Return Value	Integer	Stored version (default 1 if not explicitly set)

Use Case: Debug "did my migration run?".

6.2 Set Slot Version

Provides 6.2 Set Slot Version functionality.

Input	Type	Description
World Context	Object	World context
Slot Name	Name	Target slot
New Version	Integer	Version this slot now represents

Use Case: Bump the slot version once your migration delegate finishes transforming the data.

6.3 Register Migration Delegate

Provides 6.3 Register Migration Delegate functionality.

Input	Type	Description
World Context	Object	World context
Target Version	Integer	Version this delegate brings the slot up to
Delegate	FFoxMigrateSlotDelegate	Dynamic delegate signature (SlotName, FromVersion, ToVersion)

Use Case: Register all migrations once at game init in the GameInstance's `Event Init`.

7. Player Profile (4 Nodes)

7.1 Set Active Profile

Provides 7.1 Set Active Profile functionality.

Input	Type	Description
World Context	Object	World context
Profile Slot Name	Name	The slot to mark active

Use Case: Convenience pointer used by Save Active Profile To Disk.

7.2 Get Active Profile

Provides 7.2 Get Active Profile functionality.

Input	Type	Description
World Context	Object	World context
Output	**Type**	**Description**
Return Value	Name	Active profile slot name, or None

Use Case: "Save my coins to the active profile, whatever that is right now".

7.3 Save Active Profile To Disk

Provides 7.3 Save Active Profile To Disk functionality.

Input	Type	Description
World Context	Object	World context
Output	**Type**	**Description**
Return Value	Boolean	True on success; false if no active profile

Use Case: Persist active-profile state after any change.

7.4 Switch Active Profile

Provides 7.4 Switch Active Profile functionality.

Input	Type	Description
World Context	Object	World context
New Profile Slot Name	Name	Slot to become active after the save
Output	**Type**	**Description**
Return Value	Boolean	True on successful save+switch

Use Case: Player picks a different save slot in the menu — persist current progress then switch context.

Links & Support

- Documentation: alchemy-fox.de/FoxEasySave
- Studio & all plugins: alchemy-fox.de
- Fab store: fab.com/sellers/AlchemyFoxStudio
- Support: support@alchemy-fox.de

Supported Engine Versions: UE 5.2, 5.3, 5.4, 5.6, 5.7