FoxEasySoftRef

Soft References, Registry & Message Bus made Blueprint-friendly

User Guide • Version 1.0

Alchemy Fox Studio

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

Overview

FoxEasySoftRef turns three of Blueprint's most-avoided features - Soft References, Interfaces, and Event Dispatchers - into nodes you reach for without thinking. It ships an async Soft Loading library, Soft-Cast / Soft-Spawn helpers, a per-world Registry subsystem for finding actors without Cast, and a tag-routed Message Bus for decoupled pub/sub. Two drop-in components (Registry and Message Bus) handle registration and subscription automatically.

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 **FoxEasySoftRef** when you right-click in any Blueprint graph.

Quick Start

- Make a variable of type Soft Object Reference, then use Load Soft Object (Latent) and store the result in a variable immediately so the GC doesn't reclaim it.
- Add a Fox Registry Component to any actor, set its Unique Name or Tags in the Details panel, then find it from anywhere with Get Actor By Name or Get Actors With Tag - no Cast needed.
- Add a Fox Message Bus Component with Subscribed Tags filled in and bind its OnMessage event; from any sender call Broadcast Message with the same tag.

Examples

Example 1: Soft-load an asset and spawn it on demand

Goal: Spawn an enemy whose Blueprint class was never hard-referenced at startup, streaming it in only when a wave begins.

Add a soft variable
Create a variable of type Soft Class Reference (Actor) and point it at your enemy Blueprint. Because it's soft, the class is not loaded into memory at game start.

Pre-warm (optional)
At level start, call Load Soft Class (Latent) on the variable so the first spawn isn't stalled by streaming. Skip this if a one-frame hitch on first spawn is acceptable.

Spawn from the soft class
When the wave starts, call Spawn Actor From Soft Class (Latent) with the soft class and a Spawn Transform. Wire the latent's continuation to your post-spawn logic.

Check Success
Branch on the Success output before using Out Actor - both the load and the spawn step can fail. Store Out Actor in a variable if you need it later.

Example 2: Register an actor and look it up without Cast

Goal: Find the player and all guards from any Blueprint without hard-referencing their classes or using Cast To.

Register the player
On the player actor, add a Fox Registry Component and set Unique Name to 'Player.Main'. It auto-registers on BeginPlay and unregisters on EndPlay.

Tag the guards
On each guard actor, add a Fox Registry Component and add the tag 'Character.Guard' to its Tags container.

Look up by name
From any Blueprint, call Get Actor By Name with 'Player.Main' to get the player actor reference instantly via hash lookup.

Query by tag
Call Get Actors With Tag using 'Character.Guard' to get the array of every registered guard, then iterate it - empty array if none, safe to loop.

Example 3: Broadcast a game-wide event over the Message Bus

Goal: Raise an alarm that every nearby listener reacts to, without any of them knowing who sent it.

Subscribe the listeners
On each reacting actor, add a Fox Message Bus Component and add 'Game.AlarmRaised' to its Subscribed Tags. Bind the component's OnMessage event in the Event Graph.

Broadcast from the sender
When the alarm trips, call Broadcast Message with Message Tag 'Game.AlarmRaised', the Sender actor, and an optional Payload packed via Make Instanced Struct.

React in the handler
In each listener's OnMessage event, read Message.Sender and Message.Payload and run the reaction. Switch on Message.MessageTag if the actor subscribes to several tags.

Debug delivery
If nobody reacts, call Get Message Subscriber Count for 'Game.AlarmRaised' - a count of 0 means no one is listening for that tag.

Node Reference

Complete reference for all 30 FoxEasySoftRef nodes organized by category. Each node includes its inputs, outputs, and a use-case hint.

1. Soft Loading (5 Nodes)

1.1 Load Soft Object (Latent)

Provides 1.1 Load Soft Object (Latent) functionality.

Input	Type	Description
World Context	Object Reference	Auto-filled world context for the latent action.
Soft	Soft Object Reference	The soft object reference you want loaded.
Output	**Type**	**Description**
Out Object	Object Reference	The loaded UObject, or null on failure.
Success	Boolean	True if loaded successfully.

Use Case: Load an inventory item's mesh only when the player opens the inventory, not at game start.

1.2 Load Soft Class (Latent)

Provides 1.2 Load Soft Class (Latent) functionality.

Input	Type	Description
World Context	Object Reference	Auto-filled world context for the latent action.
Soft Class	Soft Class Reference	The class definition to load.
Output	**Type**	**Description**
Out Class	Class	The loaded UClass.
Success	Boolean	True if loaded successfully.

Use Case: Pre-load an enemy class before spawning a wave so the actual spawn call is instant.

1.3 Is Soft Object Loaded

Provides 1.3 Is Soft Object Loaded functionality.

Input	Type	Description
Soft	Soft Object Reference	The soft reference to check.
Output	**Type**	**Description**
Return Value	Boolean	True if the object is currently in memory.

Use Case: Skip the loading-bar UI when the asset is already resident in memory.

1.4 Is Soft Class Loaded

Provides 1.4 Is Soft Class Loaded functionality.

Input	Type	Description
Soft Class	Soft Class Reference	The class reference to check.
Output	**Type**	**Description**
Return Value	Boolean	True if the class is currently loaded.

Use Case: Drive a conditional 'Loading...' vs 'Ready' label based on whether the class is resident.

1.5 Get Loaded Soft Object

Provides 1.5 Get Loaded Soft Object functionality.

Input	Type	Description
Soft	Soft Object Reference	The soft reference to fetch.
Output	**Type**	**Description**
Return Value	Object Reference	The object if loaded, otherwise null.

Use Case: Take the fast synchronous path to grab an already-resident object, falling back to async load only on null.

2. Soft Cast (3 Nodes)

2.1 Soft Cast Object (Latent)

Provides 2.1 Soft Cast Object (Latent) functionality.

Input	Type	Description
World Context	Object Reference	Auto-filled world context for the latent action.
Object	Object Reference	The object to cast.
Target Class	Soft Class Reference	The class to cast to, referenced softly.
Output	**Type**	**Description**
Out Casted	Object Reference	The successfully-casted object, or null.
Success	Boolean	True if the cast succeeded.

Use Case: Replace a Cast To node so this Blueprint never gains a hard reference to the target class.

2.2 Soft Is A

Provides 2.2 Soft Is A functionality.

Input	Type	Description
Object	Object Reference	The object to test.
Target Class	Soft Class Reference	The class to test against.
Output	**Type**	**Description**
Return Value	Boolean	True if Object is an instance of Target Class.

Use Case: Filter actors by class without a Cast To node that would force a hard reference.

2.3 Get Soft Class From Class

Provides 2.3 Get Soft Class From Class functionality.

Input	Type	Description
Hard Class	Class	The hard class reference to convert.
Output	**Type**	**Description**
Return Value	Soft Class Reference	Equivalent soft class reference.

Use Case: Bridge a settings asset that stores hard class refs into a runtime API that expects soft refs.

3. Soft Spawn (2 Nodes)

3.1 Spawn Actor From Soft Class (Latent)

Provides 3.1 Spawn Actor From Soft Class (Latent) functionality.

Input	Type	Description
World Context	Object Reference	Auto-filled world context for the latent action.
Soft Class	Soft Class Reference	The actor class to spawn, referenced softly.
Spawn Transform	Transform	Where to spawn the new actor.
Output	**Type**	**Description**
Out Actor	Actor Reference	The spawned actor, or null on failure.
Success	Boolean	True if spawn succeeded.

Use Case: Spawn an item or enemy whose class you deliberately kept out of memory at startup.

3.2 Spawn Actor From Soft Class Deferred (Latent)

Provides 3.2 Spawn Actor From Soft Class Deferred (Latent) functionality.

Input	Type	Description
World Context	Object Reference	Auto-filled world context for the latent action.
Soft Class	Soft Class Reference	The actor class to spawn, referenced softly.
Spawn Transform	Transform	Where to spawn the new actor.
Output	**Type**	**Description**
Out Actor	Actor Reference	The actor; call FinishSpawning to complete it.
Success	Boolean	True if spawn succeeded.

Use Case: Set team color or owner on a soft-spawned actor before its BeginPlay logic runs, then call FinishSpawning.

4. Registry (9 Nodes)

4.1 Register Actor

Provides 4.1 Register Actor functionality.

Input	Type	Description
Actor	Actor Reference	The actor to register.
Unique Name	Name	Optional name for direct lookup.
Tags	GameplayTag Container	Tags this actor can be queried by.

Use Case: Programmatically register an actor whose lifecycle is driven by a manager rather than BeginPlay.

4.2 Unregister Actor

Provides 4.2 Unregister Actor functionality.

Input	Type	Description
Actor	Actor Reference	The actor to remove from the registry.

Use Case: Make a manually-registered actor stop appearing in lookups when it is destroyed.

4.3 Get Actor By Name

Provides 4.3 Get Actor By Name functionality.

Input	Type	Description
World Context	Object Reference	Auto-filled world context.
Unique Name	Name	The case-sensitive name to look up.
Output	**Type**	**Description**
Return Value	Actor Reference	The actor, or null if not registered.

Use Case: Fetch the player, main camera, or game manager by its well-known role name.

4.4 Get Actors With Tag

Provides 4.4 Get Actors With Tag functionality.

Input	Type	Description
World Context	Object Reference	Auto-filled world context.
Tag	GameplayTag	The tag to query for.
Output	**Type**	**Description**
Return Value	Array of Actor Reference	All matching actors.

Use Case: Find all guards at once by returning every actor registered under Character.Guard.

4.5 Get Actors With All Tags

Provides 4.5 Get Actors With All Tags functionality.

Input	Type	Description
World Context	Object Reference	Auto-filled world context.
Tags	GameplayTag Container	Required tags - all must match.
Output	**Type**	**Description**
Return Value	Array of Actor Reference	Actors that have every listed tag.

Use Case: Find guards that are also currently alarmed by intersecting Character.Guard and State.Alarmed.

4.6 Get Actors With Any Tags

Provides 4.6 Get Actors With Any Tags functionality.

Input	Type	Description
World Context	Object Reference	Auto-filled world context.
Tags	GameplayTag Container	Tags - at least one must match.
Output	**Type**	**Description**
Return Value	Array of Actor Reference	Deduplicated actors matching any tag.

Use Case: Find any hostile by gathering actors tagged Enemy OR Bandit OR Monster in one deduplicated list.

4.7 Get Actors With Interface

Provides 4.7 Get Actors With Interface functionality.

Input	Type	Description
World Context	Object Reference	Auto-filled world context.
Interface Class	Soft Class Reference	The interface to filter by, referenced softly.
Output	**Type**	**Description**
Return Value	Array of Actor Reference	Actors implementing the interface.

Use Case: Find every interactable in the level regardless of class via a soft interface reference.

4.8 Get First Actor With Interface

Provides 4.8 Get First Actor With Interface functionality.

Input	Type	Description
World Context	Object Reference	Auto-filled world context.
Interface Class	Soft Class Reference	The interface to find, referenced softly.
Output	**Type**	**Description**
Return Value	Actor Reference	The first matching actor, or null.

Use Case: Grab the single Game Manager actor by interface, faster than fetching the whole array.

4.9 Get Nearest Actor With Tag

Provides 4.9 Get Nearest Actor With Tag functionality.

Input	Type	Description
World Context	Object Reference	Auto-filled world context.
Location	Vector	Reference point for the distance query.
Tag	GameplayTag	Required tag.
Max Radius	Float	Distance limit; 0 means unlimited.
Output	**Type**	**Description**
Return Value	Actor Reference	Nearest matching actor, or null.

Use Case: Find the nearest enemy guard to the player's current position within a search radius.

5. Interface Helpers (4 Nodes)

5.1 Does Implement Interface

Provides 5.1 Does Implement Interface functionality.

Input	Type	Description
Actor	Actor Reference	The actor to test.
Interface Class	Soft Class Reference	The interface to test for, referenced softly.
Output	**Type**	**Description**
Return Value	Boolean	True if the actor implements the interface.

Use Case: Verify a trace-hit actor supports an interface before sending it an Interface Message.

5.2 Call Interface Function (Generic)

Provides 5.2 Call Interface Function (Generic) functionality.

Input	Type	Description
Actor	Actor Reference	The target actor.
Function Name	Name	Exact, case-sensitive Blueprint function name.
Params	FFoxFunctionParams	Generic key/value parameter pack.
Output	**Type**	**Description**
Return Value	Boolean	True if the named function existed and was called.

Use Case: Call Interact by name on whatever the player is looking at without knowing its class.

5.3 Bind To Interface Delegate

Provides 5.3 Bind To Interface Delegate functionality.

Input	Type	Description
Actor	Actor Reference	The actor with the delegate property.
Delegate Name	Name	Name of the multicast delegate UPROPERTY.
Callback	FFoxGenericDelegate	Function to call when the delegate fires.
Output	**Type**	**Description**
Return Value	GUID	Handle for later unbinding - store this.

Use Case: Listen for an OnDoorOpened event on an actor whose specific class you don't know.

5.4 Unbind Interface Delegate

Provides 5.4 Unbind Interface Delegate functionality.

Input	Type	Description
Handle	GUID	The handle returned from Bind To Interface Delegate.

Use Case: Detach a reflection-bound delegate in EndPlay to avoid a dangling reference.

6. Message Bus (7 Nodes)

6.1 Subscribe To Message

Provides 6.1 Subscribe To Message functionality.

Input	Type	Description
World Context	Object Reference	Auto-filled world context.
Message Tag	GameplayTag	GameplayTag to listen for.
Callback	FFoxMessageDelegate	Function called when a matching message arrives.
Subscriber	Actor Reference	The actor that owns this subscription.
Output	**Type**	**Description**
Return Value	GUID	Handle for later unsubscribe - store this.

Use Case: Listen for game-wide events like Game.PlayerDied from script without a component.

6.2 Unsubscribe From Message

Provides 6.2 Unsubscribe From Message functionality.

Input	Type	Description
World Context	Object Reference	Auto-filled world context.
Subscription Handle	GUID	The handle returned from Subscribe To Message.

Use Case: Stop listening for quest-progress messages once the quest is completed.

6.3 Unsubscribe All For Actor

Provides 6.3 Unsubscribe All For Actor functionality.

Input	Type	Description
World Context	Object Reference	Auto-filled world context.
Subscriber	Actor Reference	The actor whose subscriptions to remove.

Use Case: Clear every manual subscription an actor created in one call from its EndPlay.

6.4 Broadcast Message

Provides 6.4 Broadcast Message functionality.

Input	Type	Description
World Context	Object Reference	Auto-filled world context.
Message Tag	GameplayTag	GameplayTag identifying the message.
Sender	Actor Reference	Optional actor sending the message; can be null.
Payload	FInstancedStruct	Optional generic struct of extra data.

Use Case: Fire an alarm or world-state change to every subscriber of a tag at once.

6.5 Send Message To Actor

Provides 6.5 Send Message To Actor functionality.

Input	Type	Description
Target	Actor Reference	The actor to message.
Message Tag	GameplayTag	The tag identifying the message.
Sender	Actor Reference	Optional sender; can be null.
Payload	FInstancedStruct	Optional generic struct of extra data.

Use Case: Send a TakeDamage message to one specific enemy that may have several handlers for it.

6.6 Send Message To Actors With Tag

Provides 6.6 Send Message To Actors With Tag functionality.

Input	Type	Description
World Context	Object Reference	Auto-filled world context.
Actor Tag	GameplayTag	Registry tag - only actors with this tag receive the send.
Message Tag	GameplayTag	The message tag delivered.
Sender	Actor Reference	Optional sender; can be null.
Payload	FInstancedStruct	Optional generic struct of extra data.

Use Case: Send a Retreat command only to actors registered under Team.Player in one combined call.

6.7 Get Message Subscriber Count

Provides 6.7 Get Message Subscriber Count functionality.

Input	Type	Description
World Context	Object Reference	Auto-filled world context.
Message Tag	GameplayTag	The tag to query.
Output	**Type**	**Description**
Return Value	Integer	Number of active subscribers for the tag.

Use Case: Diagnose an unreceived broadcast by confirming whether anyone is subscribed to the tag.

Links & Support

- Documentation: alchemy-fox.de/FoxEasySoftRef
- 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