

RPG Action Bar & Inventory

Complete Buyer Guide

Version 1.1 | Unreal Engine 5.8 | Windows

Contents

Start here: choose one setup

Set up the healing lesson

Your first successful use

Connect and disconnect the owner

Assign an existing item to the bar

Connect one button

Request the item effect once

Understand the two potion examples

Everyday inventory and bar use

Appearance and window behavior

Make your own item

Connect your own healing effect

Add a custom right-click action

Bags add storage space

Listen for bag and capacity changes

Optional: display equipped items

Optional: equipment feedback

Optional: put an ability on the bar

Optional: accept the ability once

Optional: read the ability result

Punches and your character animation

Save the action bar

Save inventory stock

Load without replaying gameplay

Advanced: who owns the bar?

If something does not work

Before shipping your game

Quick glossary and support

Start here: choose one setup

Use one inventory and one action bar for each player. Choose the route that fits what you are doing.

1. Install the combined ActionBarInventoryKit plugin in your project Plugins folder. Enable it and its required ACL plugin, then restart the editor when prompted. Do not also install the two standalone products: this package already contains their modules.
2. To see the ordinary interface, use the Inventory Kit and Action Bar project settings for automatic local-player startup. Open the inventory with I or B. An empty inventory is normal; preview items are an explicit demonstration option.
3. To learn the combined Blueprint hookup, use the placed-actor lesson on the next page. Turn off automatic startup for both systems first, so you do not create a second inventory behind the scenes.
4. In the Content Browser, enable Show Plugin Content. Search the exact Blueprint names printed in this guide. Make your own child Blueprint or working copy for game-specific changes.

Expected result: The interface route uses the player/controller. The teaching route uses one placed actor. Keep those routes separate while learning.

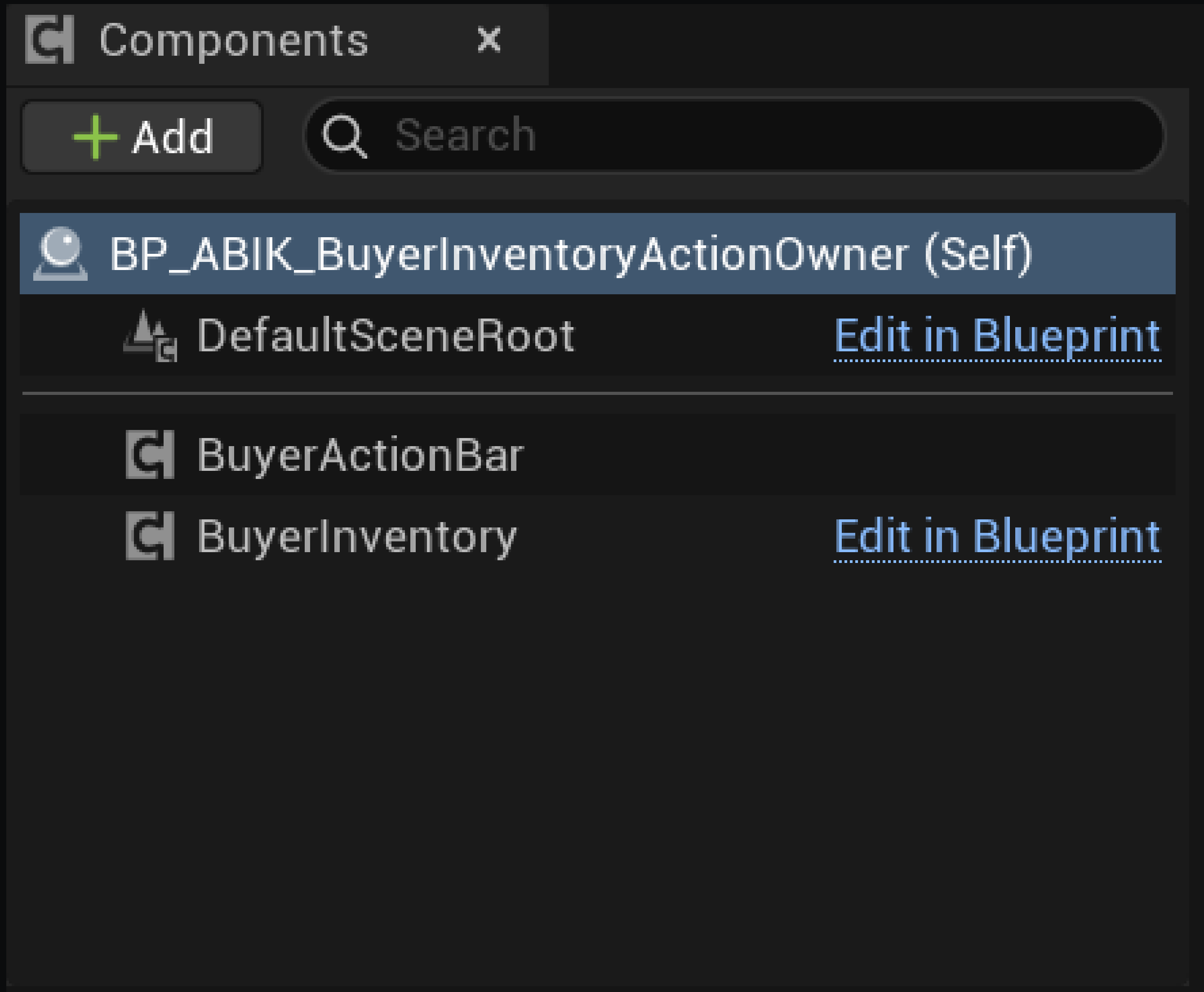
Set up the healing lesson

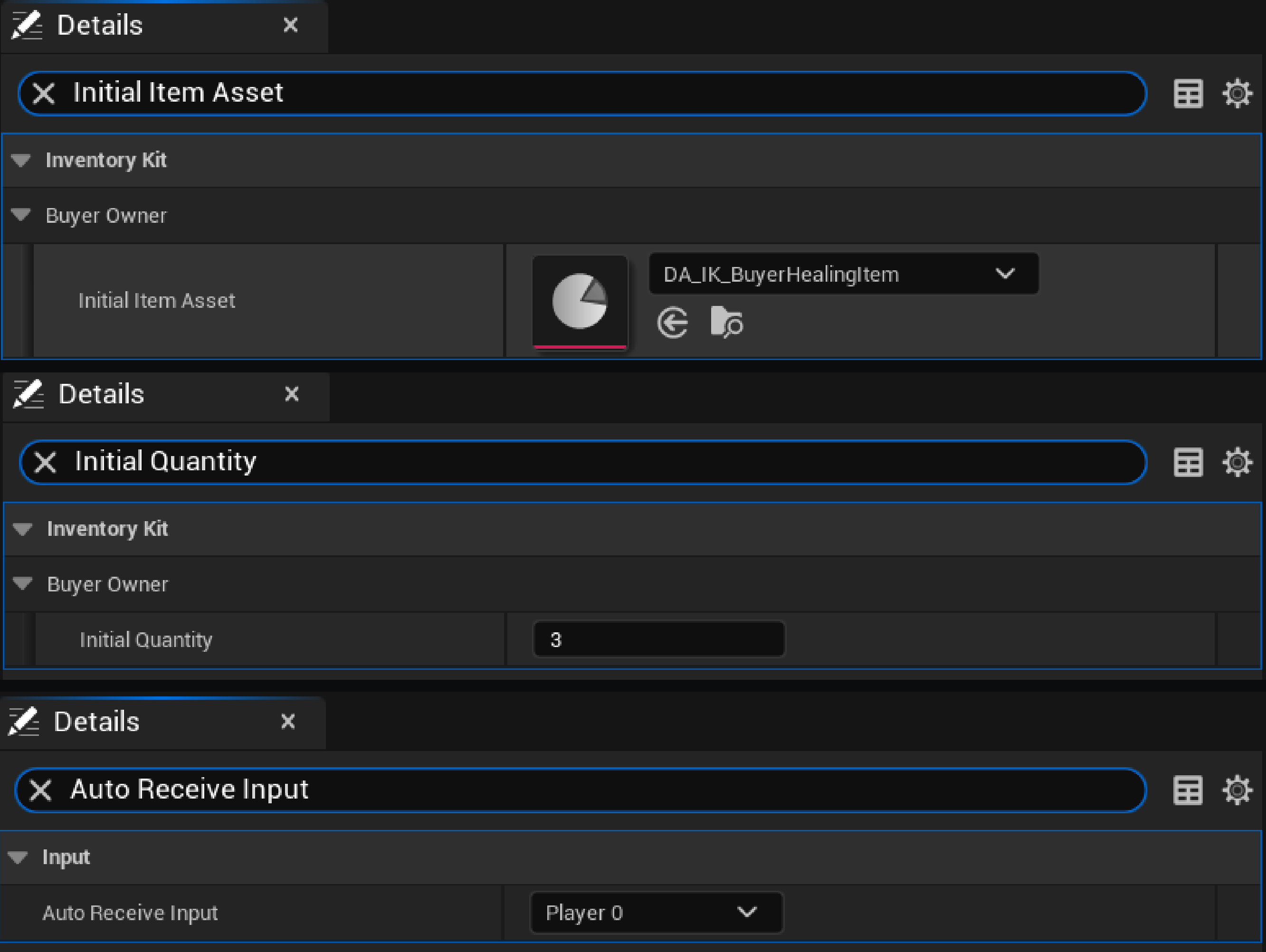
Open BP_ABIK_BuyerInventoryActionOwner. These four panels are the complete owner setup for this lesson.

1. Place one instance in your test level. Keep its BuyerInventory and BuyerActionBar components together.
2. Click Class Defaults. In Details, search Initial Item Asset and choose DA_IK_BuyerHealingItem. Search Initial Quantity and set 3.
3. Search Auto Receive Input and choose Player 0. Compile and save. Close other input menus before testing J.
4. For a fresh lesson, use a separate demo save slot so old stock does not hide the starting values. Do not overwrite your player save.

Expected result: The lesson starts with three healing items. The retained healing example starts at 50 health, with a maximum of 100.

[/ActionBarInventoryKit/ActionBarInventoryKit/Examples/BP_ABIK_BuyerInventoryActionOwner](#)





Your first successful use

Run your lesson level and press J once for each test. This is a check you can perform in your own project.

1. First press: the healing example goes from 50 to 75 health; inventory stock goes from 3 to 2.
2. Second press: health reaches 100; stock goes from 2 to 1.
3. Third press: full health prevents another use. The last item stays in inventory.
4. H is the inherited direct-inventory lesson. J is the combined action-bar lesson. In your game, give each button one route to the effect.
5. If nothing happens, check Player 0 input, both component references, the Initial Item Asset, and that automatic startup is off for the placed-actor lesson.

Expected result: A shortcut does not create another item. Using the shortcut spends stock from the same inventory.

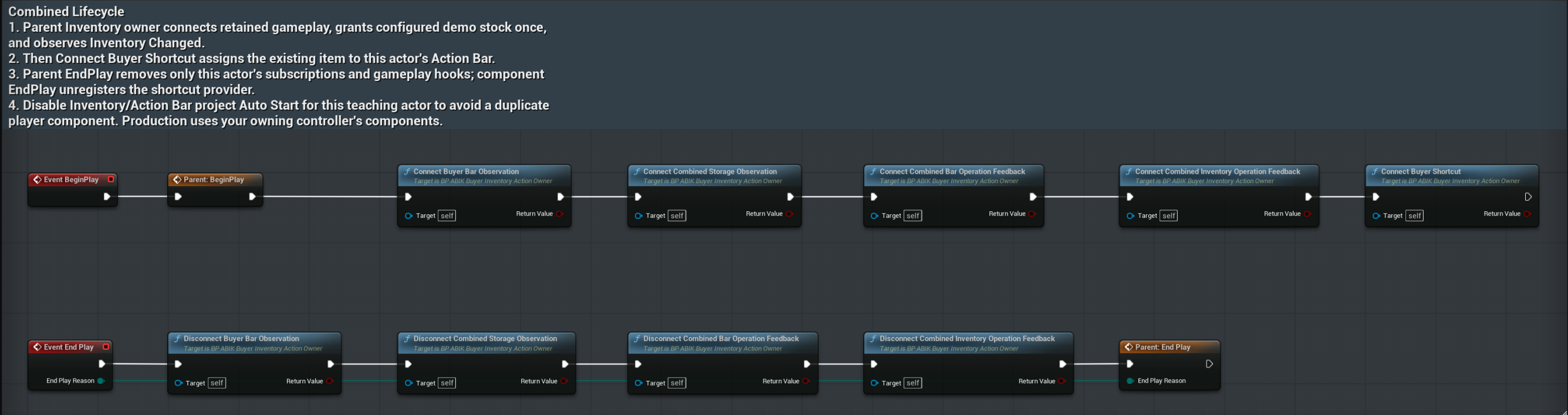
Connect and disconnect the owner

BeginPlay means the actor has started. EndPlay means it is leaving. The parent call runs the inherited inventory setup.

1. Let Parent BeginPlay connect the gameplay example and grant the configured starting stock once.
2. Connect the bar, bag and operation feedback. Then call Connect Buyer Shortcut.
3. When the actor ends, disconnect its named callbacks and run Parent EndPlay. A callback is simply a function that listens for a change.
4. Keep the gameplay object alive for as long as this inventory needs it. Do not reconnect or grant starting stock every frame.

Expected result: One owner connects once and removes its own listeners when it leaves.

/ActionBarInventoryKit/ActionBarInventoryKit/Examples/BP_ABIK_BuyerInventoryActionOwner / EventGraph



Connect and disconnect the owner | [Open full-size original](#)

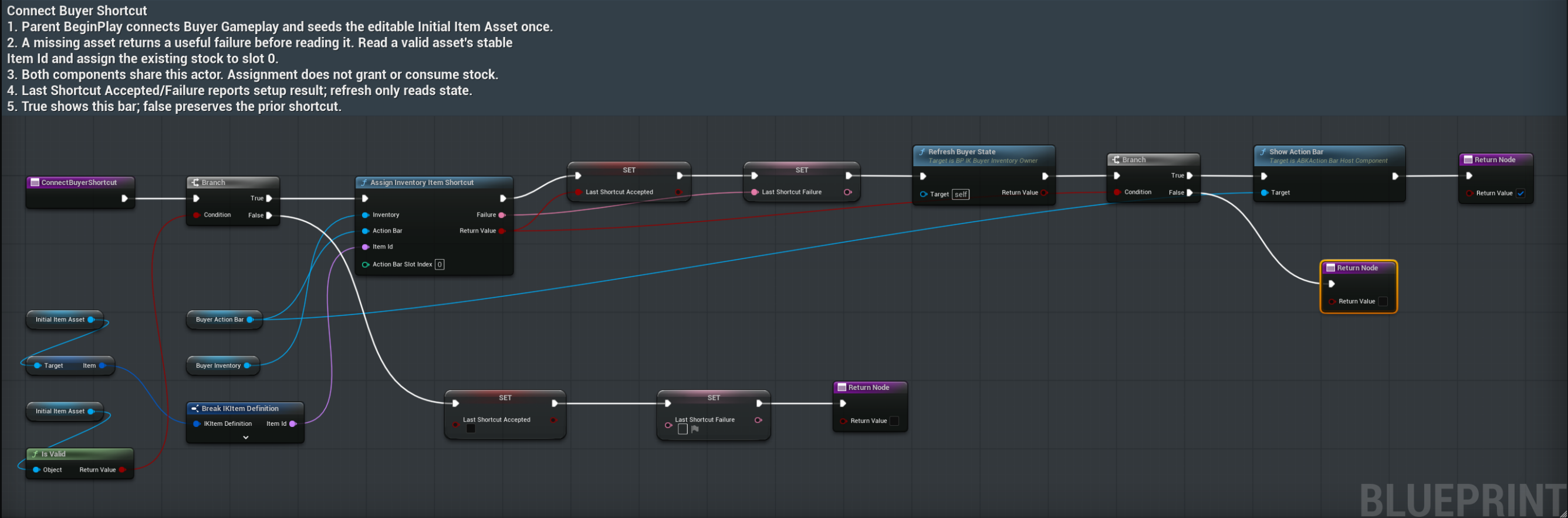
Assign an existing item to the bar

Connect Buyer Shortcut takes the stable Item Id from Initial Item Asset and assigns it to slot 0.

1. Read the item definition only after checking that the asset is valid.
2. Pass BuyerInventory, BuyerActionBar and Item Id to Assign Inventory Item Shortcut. Both components belong to this actor.
3. Read Last Shortcut Accepted and Last Shortcut Failure. On success, refresh the displayed state and show the bar.
4. On failure, show the reason and keep the previous shortcut. Assignment does not grant stock, consume stock or run the item effect.

Expected result: The bar points to an item already owned by the inventory.

/ActionBarInventoryKit/ActionBarInventoryKit/Examples/BP_ABIK_BuyerInventoryActionOwner / ConnectBuyerShortcut



Assign an existing item to the bar | [Open full-size original](#)

Connect one button

The example J key calls Trigger Buyer Shortcut once.

- 1. Use Pressed for a single request. Replace this key event with your own controller or Enhanced Input action when integrating.
- 2. Do not also call Inventory Use Item from the same button or from a result/state observer.
- 3. If you bind a bar shortcut key for the same action, remove the duplicate actor key event.

Expected result: One button press produces one request. The full instruction above also supplies the last comment line obscured in the original capture.

/ActionBarInventoryKit/ActionBarInventoryKit/Examples/BP_ABIK_BuyerInventoryActionOwner / CombinedInput

Combined Input

1. The inherited actor receives Player 0 input in this single-player teaching level. Press J for one action-bar-to-inventory request.

2. The parent H input teaches direct inventory use; use one route per button in your game.

3. Replace this key event with your own input owner. Never call Inventory Use Item again from a result/state observer.



Connect one button | [Open full-size original](#)

Request the item effect once

Trigger Buyer Shortcut asks BuyerActionBar to trigger slot 0. Inventory then checks whether its item can be used.

- 1. Call Request Trigger Slot on the retained bar. Store its actual accepted/refused result.
- 2. Refresh Buyer State reads health and stock for display. It does not repeat the effect.
- 3. Full health, no stock, an active cooldown or a disconnected owner can refuse the request. Treat a refusal as useful feedback.

Expected result: The accepted path applies the gameplay effect once; inventory owns any consumption.

/ActionBarInventoryKit/ActionBarInventoryKit/Examples/BP_ABIK_BuyerInventoryActionOwner / TriggerBuyerShortcut

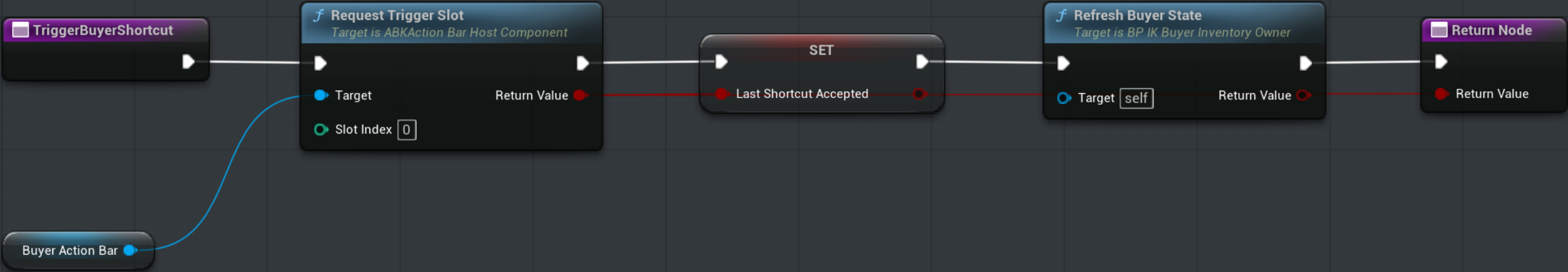
Trigger Buyer Shortcut

1. J requests this bar's slot 0 exactly once. Inventory owns gameplay dispatch and stock.

2. Last Shortcut Accepted reports success/refusal. Refresh Buyer State reads health and stock; it never repeats the effect.

3. Full health, no stock, a disconnected owner or an active cooldown rejects activation.

4. Change this input to your own controller/Enhanced Input flow in production.



Request the item effect once | [Open full-size original](#)

Understand the two potion examples

The main interface health potion and the small teaching item have different purposes.

1. The main UI health-potion entries use the same HealthPotion inventory item and original Inventory Kit health-potion icon. Their shortcut identity is IK.HealthPotion. Both entries draw from the same stock and use the same inventory activation route.
2. A shortcut is not a second stack. Assigning or migrating a potion shortcut must not grant extra potions.
3. DA_IK_BuyerHealingItem is the separate editable lesson asset. It supplies the three-item healing exercise in this guide. Do not rename it to the main potion just to match the icon.
4. For your game, connect the health effect to your own health system. The UI cannot infer which character should be healed or how your health is saved.

Expected result: Use a stable Item Id for each real item, and have every shortcut to that item use that same identity.

Everyday inventory and bar use

The inventory holds stock. The bar lets the player reach actions quickly.

1. Open inventory with I or B in the ordinary setup. Drag stacks to move, merge or swap them. Shift-drag splits a stack; the context menu also offers a quantity-based split.
2. Search and category filters change what you see, not the underlying slot identity. Sorting can include hidden items. Re-read the current slot before a delayed action.
3. Assign inventory items through Assign Inventory Item Shortcut in your game hookup. Use the bar slot or its configured binding to request the action.
4. Keyboard/controller navigation uses the UI focus: arrows or D-pad move, Enter/Space or the bottom face button opens actions, and Escape or the right face button cancels. Check conflicts with your own input mapping.
5. A cooldown means the action is temporarily unavailable. Start or display a cooldown after the game accepts the effect, not simply because the player clicked.

Expected result: Your own input setup can change these controls. Keep the displayed binding and the actual input route in agreement.

Appearance and window behavior

Use the built-in settings before changing Blueprint or C++ code.

1. Inventory appearance settings adjust scale and columns. Save Settings keeps the chosen appearance. Reset Appearance is for presentation; it is not an inventory wipe.
2. Shift-drag the inventory header to move its window. Where a 3D item preview is supplied, drag to rotate and use the wheel to zoom.
3. Use Action Bar settings for its layout, bindings and presentation. A layout change should preserve the gameplay owner and inventory stock.
4. The combined launcher has project settings for button size, corner inset and background opacity. A registered BIS dock keeps its own presentation.
5. If your game manages cursor and input focus itself, configure inventory input-mode management before opening it. Your controller remains responsible for restoring your game-specific focus behavior.

Expected result: Appearance settings change how the interface looks; they should not replay gameplay effects.

Make your own item

Duplicate an appropriate item definition, then give it a stable identity and the presentation your players need.

1. Set Item Id, display name, category, icon and stack limit. Keep the Item Id stable after release so saves and shortcuts can still find the item.
2. Choose the intended Primary Action and whether accepted use consumes stock. For the healing lesson, the Primary Action is Use and Consume On Use is enabled.
3. The lesson healing tag is an ordinary Name value used by its example. Connect the appropriate activation owner before testing use.
4. Grant stock through your game reward, pickup or load path. Check how many items were actually added when capacity is limited; handle the remainder.
5. Assign the owned Item Id to a bar slot. Changing its icon does not give it a gameplay effect.

Expected result: The standalone Inventory screenshot below describes the shared healing definition. The combined owner adds the action-bar connection.

/InventoryKit/InventoryKit/Examples/DA_IK_BuyerHealingItem

[Make your own item \(shared Inventory guide reference\)](#) | [Open full-size original](#)

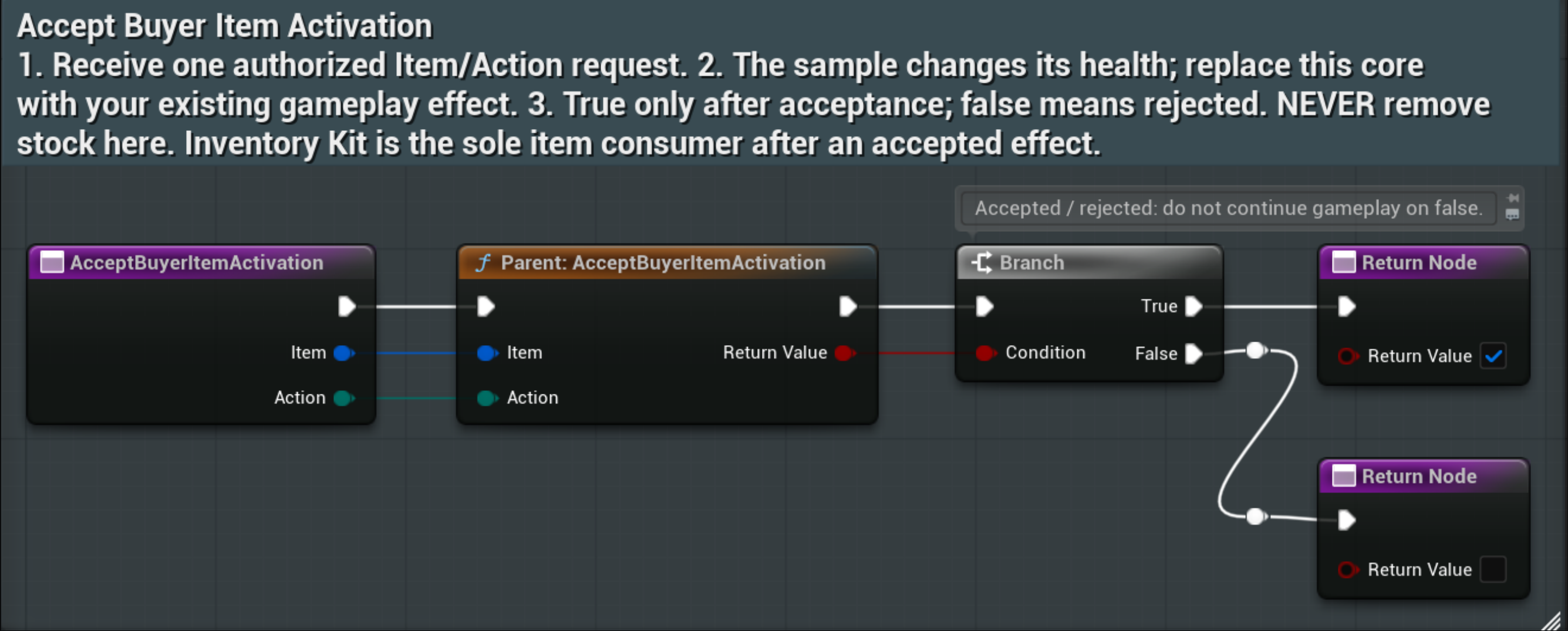
Connect your own healing effect

An activation owner is the object that decides whether an item can be used and performs its game effect.

1. Retain your subclass of BP_IK_BuyerActivation and connect it to the intended inventory.
2. Use Get Buyer Use Unavailable Reason for a read-only check, such as whether health is already full.
3. At Accept Buyer Item Activation, check readiness again, apply your intended effect once, and return the real accepted result.
4. The shown parent call already applies the example healing. Replace the effect boundary for your health system; do not call that parent and then heal a second time.
5. Let inventory handle accepted consumption. Result listeners may update text or sound, but must not heal or remove items again.

Expected result: The game owns health. Inventory owns stock. The return value tells inventory whether the game accepted the use.

/InventoryKit/InventoryKit/Examples/BP_IK_BuyerActivation / AcceptBuyerItemActivation



Connect your own healing effect (shared Inventory guide reference) | [Open full-size original](#)

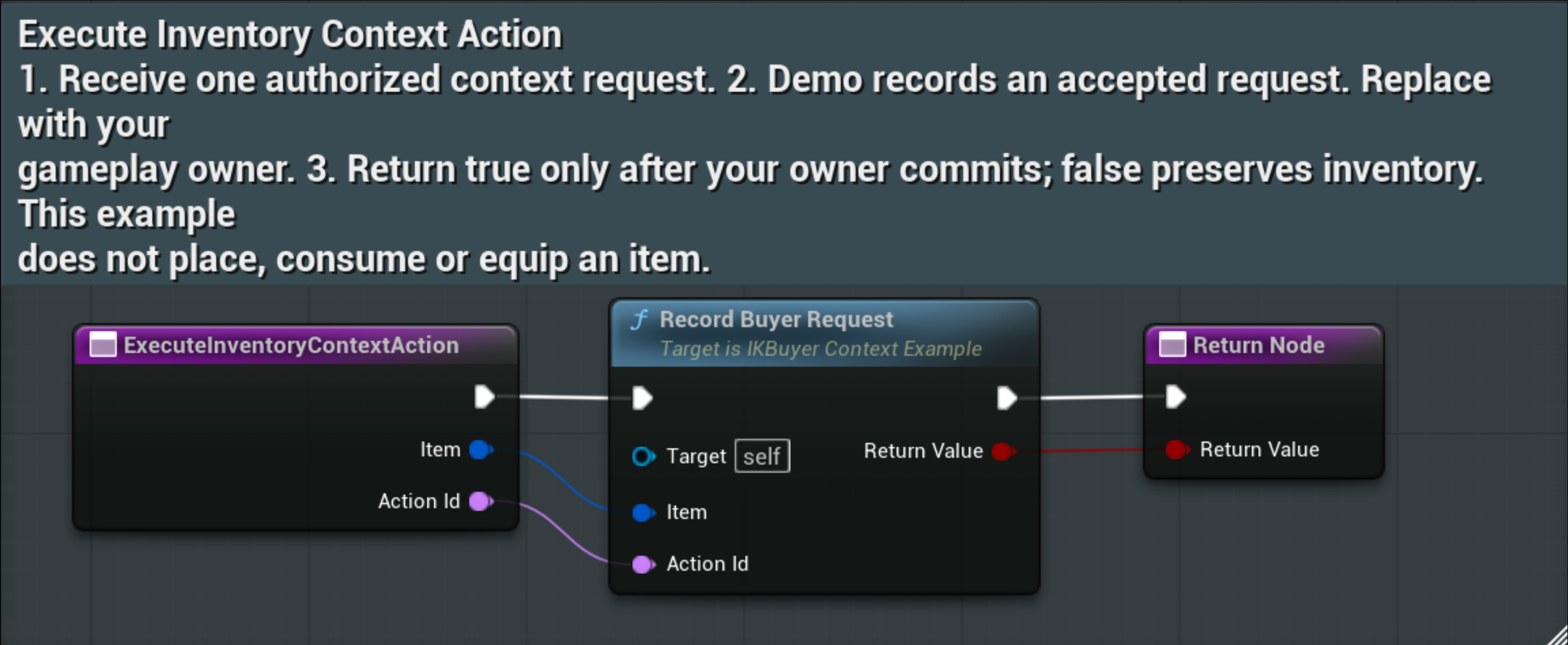
Add a custom right-click action

A context action is an extra item-menu command, such as Study Item.

1. Retain your subclass of BP_IK_BuyerContext and connect it to the same inventory.
2. Give each menu action a stable Action Id and a readable label. Check availability without changing gameplay state.
3. Execute Inventory Context Action is the place to run the accepted custom action. Return its actual result.
4. The shared example records the request. Replace or extend that behavior deliberately for your game. A menu command does not automatically consume stock.
5. Disconnect your own provider and listeners when their owner leaves.

Expected result: Menu building and availability checks read state; execution is the only place that commits the action.

/InventoryKit/InventoryKit/Examples/BP_IK_BuyerContext / ExecuteInventoryContextAction



Add a custom right-click action (shared Inventory guide reference) | [Open full-size original](#)

Bags add storage space

A storage bag is an owned item that adds capacity while equipped in a bag socket.

1. Use a bag definition with the intended storage bonus. The Buyer Storage Bag lesson adds eight slots.
2. Grant the bag through your game, then equip that owned bag into an available socket. Choosing an asset in defaults is not itself a stock grant.
3. Before removing a bag, make room for all remaining items within the smaller capacity. Refusal protects the items that would no longer fit.
4. Keep the destination for a removed bag inside the reduced capacity. Show the returned failure instead of silently discarding stock.

Expected result: Equip and unequip change capacity through the inventory owner.

/InventoryKit/InventoryKit/Examples/DA_IK_BuyerStorageBag

File

Edit

Asset

Window

Tools

Help

DA_IK_BuyerStorageBag x

Details

Search

Inventory Kit

Item

Item Id	BuyerStorageBag
Display Name	Buyer storage bag
Description	
Category	Bag
Tags	0 Array elements
Metadata	0 Map elements
Rarity	Common
Max Stack Size	1
Unit Weight	0.0
Base Value	0
Usable	
Consume on Use	
Primary Action	Equip
Activation Ability Id	None
Action Bar Cooldown Seconds	0.0
Icon	None None
External Icon Plugin	None
External Icon Resource	
Inspection Mesh	None None
Preview Subtitle	
Preview Stats	0 Array elements
Default Preview Quantity	1
Inspection Framing Scale	1.0
Inspection Initial Yaw	25.0
Inspection Initial Pitch	-8.0
Inspection Initial Zoom	1.0
Storage Slot Bonus	8

Bags add storage space (shared Inventory guide reference) | Open full-size original

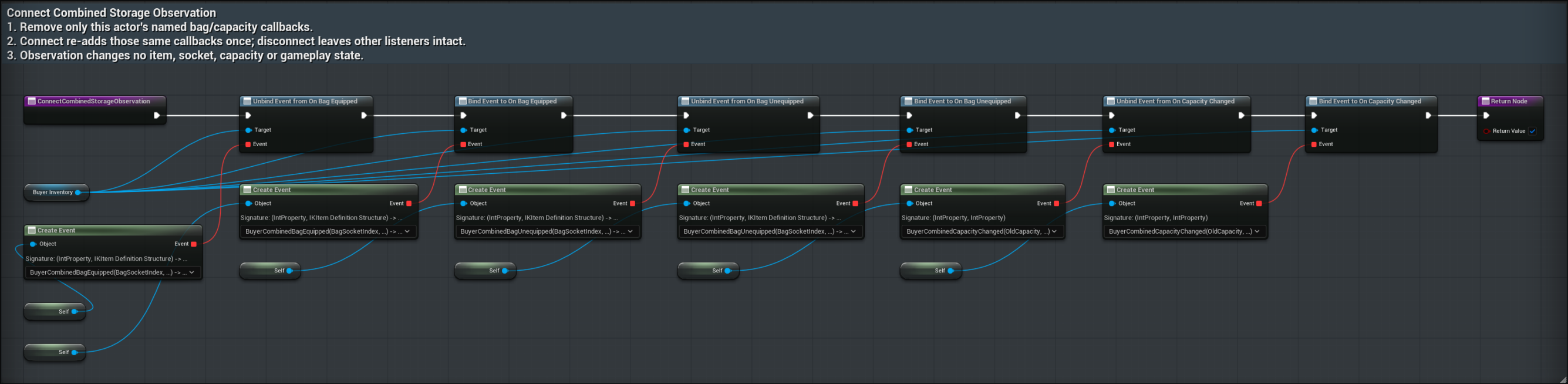
Listen for bag and capacity changes

Connect Combined Storage Observation keeps this actor informed when bags or capacity change.

1. Unbind this actor's named callback, then bind the same callback once. This prevents duplicate notifications when setup is repeated.
2. Read the committed bag and capacity state to update your HUD, counters or help text.
3. Do not equip bags, grant items or alter stock from the notification itself.
4. Disconnect only this actor's callbacks during cleanup; leave other listeners intact.

Expected result: Observing a change does not perform the change again.

/ActionBarInventoryKit/ActionBarInventoryKit/Examples/BP_ABIK_BuyerInventoryActionOwner / ConnectCombinedStorageObservation



Listen for bag and capacity changes | Open full-size original

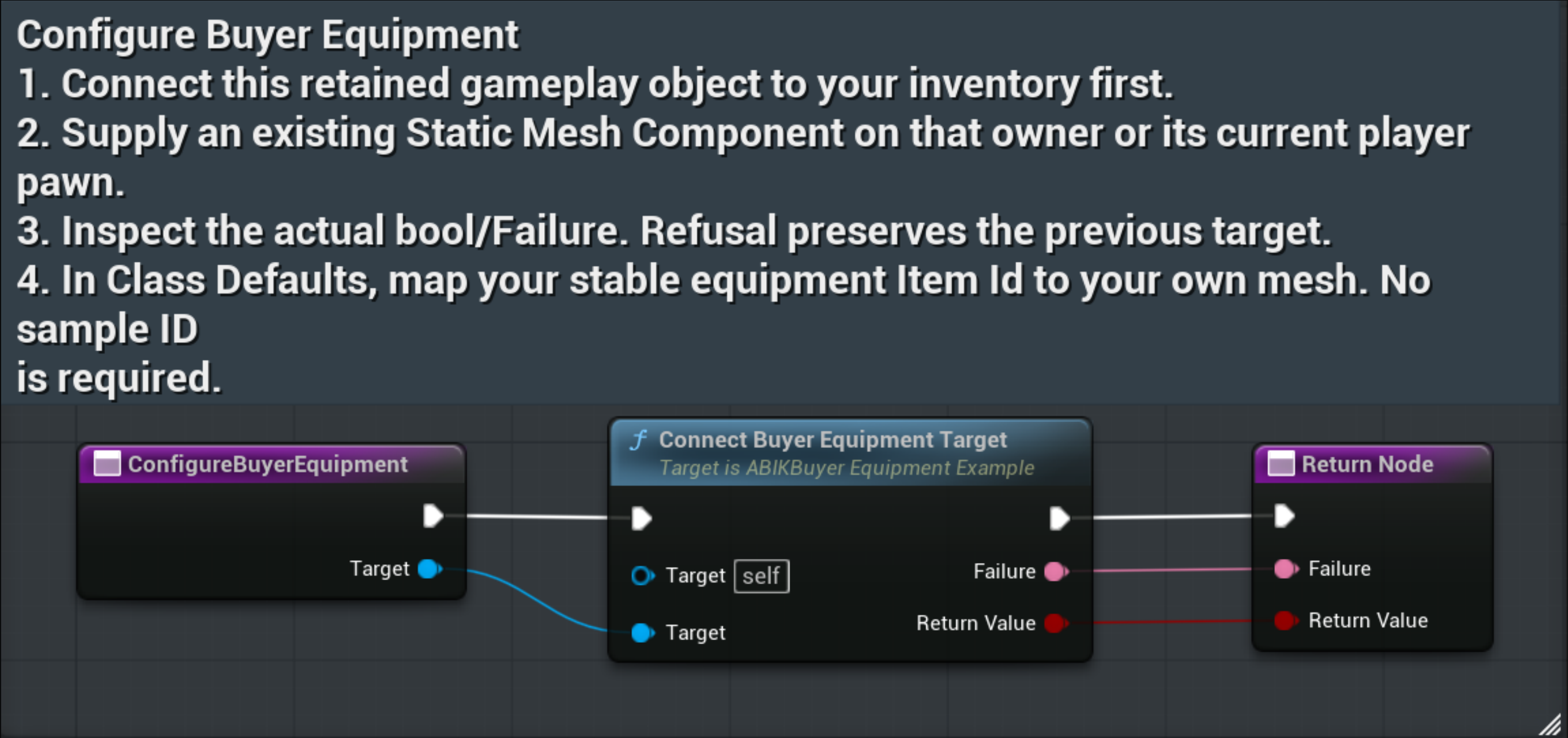
Optional: display equipped items

The equipment example changes an existing Static Mesh Component. It is a starting point for visible equipment, not a complete gear-stat system.

1. Create and retain BP_ABIK_BuyerEquipmentGameplay. Connect its inherited inventory owner first.
2. In Class Defaults, search Equipment Meshes. Add a map entry from your stable equipment Item Id to the mesh you want displayed.
3. Pass an existing Static Mesh Component on that inventory owner or its current player pawn into Configure Buyer Equipment. The target is supplied at runtime.
4. Check Return Value and Failure. Refusal keeps the previous target. Use an owned equipment item and configure consumption appropriately.
5. For weapon icons, map Item Ids in Equipment Weapon Categories and connect the intended weapon-category source. Your game still owns combat behavior and equipment statistics.

Expected result: An accepted equipment action changes the configured mesh target.

/ActionBarInventoryKit/ActionBarInventoryKit/Examples/BP_ABIK_BuyerEquipmentGameplay / ConfigureBuyerEquipment



Optional: display equipped items | Open full-size original

Optional: equipment feedback

Connect Buyer Equipment Feedback listens after the equipment change has committed.

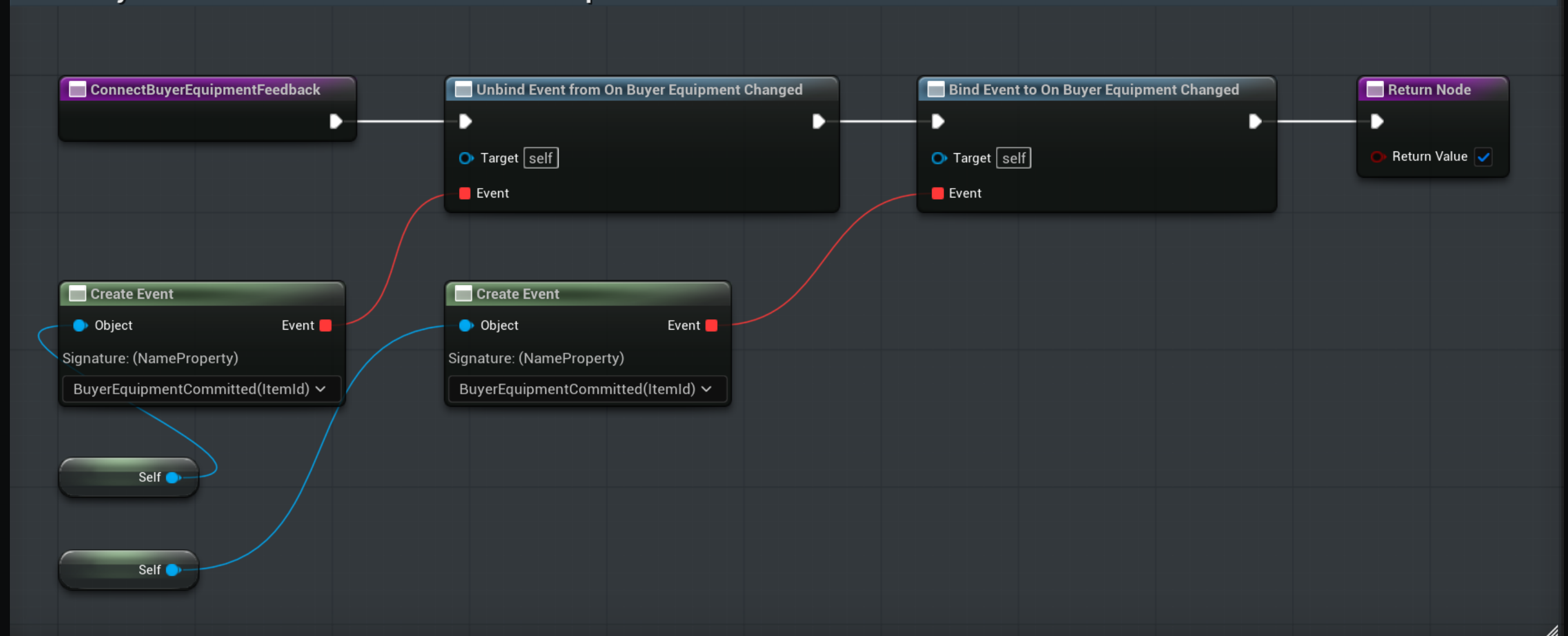
1. Call it after inventory and equipment-target setup succeed, before accepting input.
2. Use Buyer Equipment Committed to update UI or play feedback. Do not equip the item a second time from this event.
3. The function removes and re-adds only this object's own named callback. Repeated setup should not multiply notifications.
4. Disconnect feedback and disconnect the retained inventory owner during cleanup. Disconnecting listeners does not undo an already committed mesh change.

Expected result: Use the dedicated unequip operation when you intend to restore the earlier mesh; cleanup and unequip have different purposes.

[/ActionBarInventoryKit/ActionBarInventoryKit/Examples/BP_ABIK_BuyerEquipmentGameplay / ConnectBuyerEquipmentFeedback](#)

Connect Buyer Equipment Feedback

1. Remove only this retained object's named feedback callback.
2. Connect re-adds the same callback once; repeated setup is idempotent.
3. Disconnect preserves other subscribers and committed mesh/stock.
4. Call Connect after successful gameplay/target setup, before input. Own Disconnect Buyer Inventory removes feedback before native cleanup.



Optional: equipment feedback | [Open full-size original](#)

Optional: put an ability on the bar

Use the ability example when an owned inventory item should request the existing local melee router.

1. Use the player/controller-owned inventory and the existing ability router on its current pawn. Keep the gameplay object alive.
2. Create an owned item with a stable Item Id and Primary Action Activate. Configure whether this item should be consumed for your design.
3. Call Connect Buyer Ability with Inventory, Router and Item Id. These are runtime inputs, not extra Class Defaults fields.
4. Check the returned accepted value and Failure. This connection removes only that router's automatic listeners and keeps the combined bar provider.
5. Assign the owned item through Assign Inventory Item Shortcut. Give it one incoming input route.

Expected result: This example connects an existing local ability; it does not supply a multiplayer ability framework.

/ActionBarInventoryKit/ActionBarInventoryKit/Examples/BP_ABIK_BuyerAbilityGameplay / ConnectBuyerAbility

Connect Buyer Ability

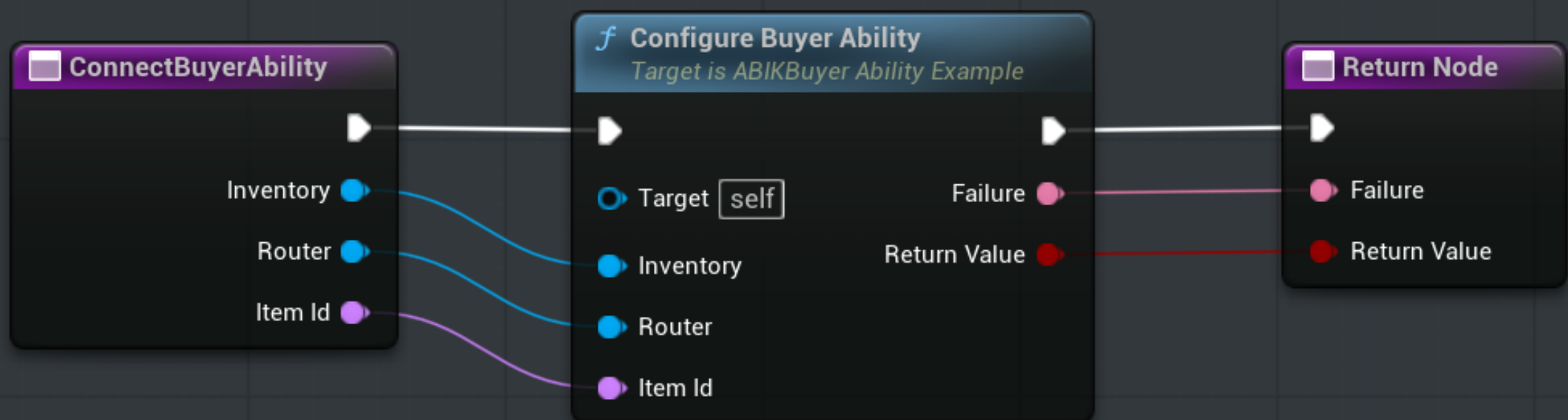
1. Retain this object in your player owner. Supply its Inventory and an existing beginner router on the current pawn.

2. Set Item Id to your own stable Activate item. Configure returns the real accepted/Failure pair.

3. Explicit connection removes only this router's automatic listeners. The existing combined bar provider keeps ownership.

4. Assign that owned inventory item to a bar slot using Assign Inventory Item Shortcut.

No automatic network ability framework is supplied.



Optional: accept the ability once

The parent of Accept Buyer Item Activation rechecks readiness and calls the current pawn router once.

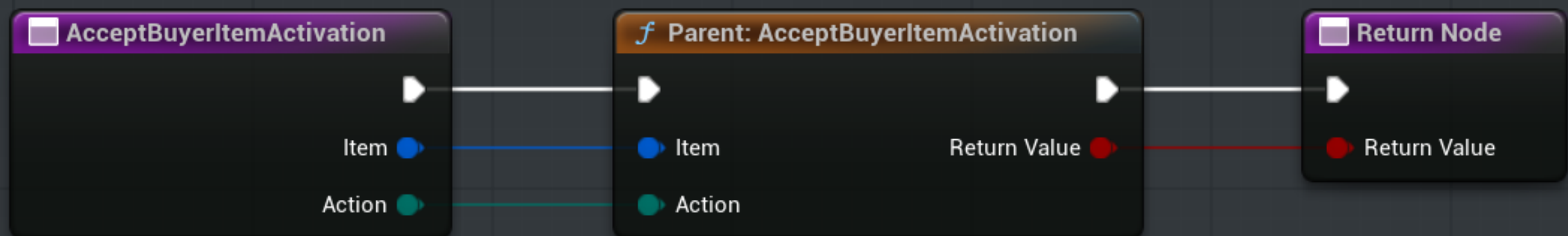
1. Return the actual accepted value from the parent. Do not report success just because the object reference exists.
2. The melee sweep applies point damage to a hit actor. A valid attack that misses can still be an accepted activation.
3. Use Last Ability Result to read hit, target, damage, refusal reason and punch index.
4. Do not apply damage or consume inventory again in this wrapper.

Expected result: Your game owns the target health and its save data.

/ActionBarInventoryKit/ActionBarInventoryKit/Examples/BP_ABIK_BuyerAbilityGameplay / AcceptBuyerItemActivation

Accept Buyer Item Activation

1. Parent rechecks readiness and calls the existing pawn router exactly once.
2. Its actual melee sweep applies Unreal point damage to the hit buyer actor. An accepted miss remains an accepted activation.
3. Return the actual bAccepted; Last Ability Result includes hit, target, damage, reason and punch index. Never consume inventory here.
4. Observe Buyer Ability Result is feedback after the effect. Do not apply damage again.



Optional: accept the ability once | [Open full-size original](#)

Optional: read the ability result

Observe Buyer Ability Result runs after an accepted effect and stores the result for feedback.

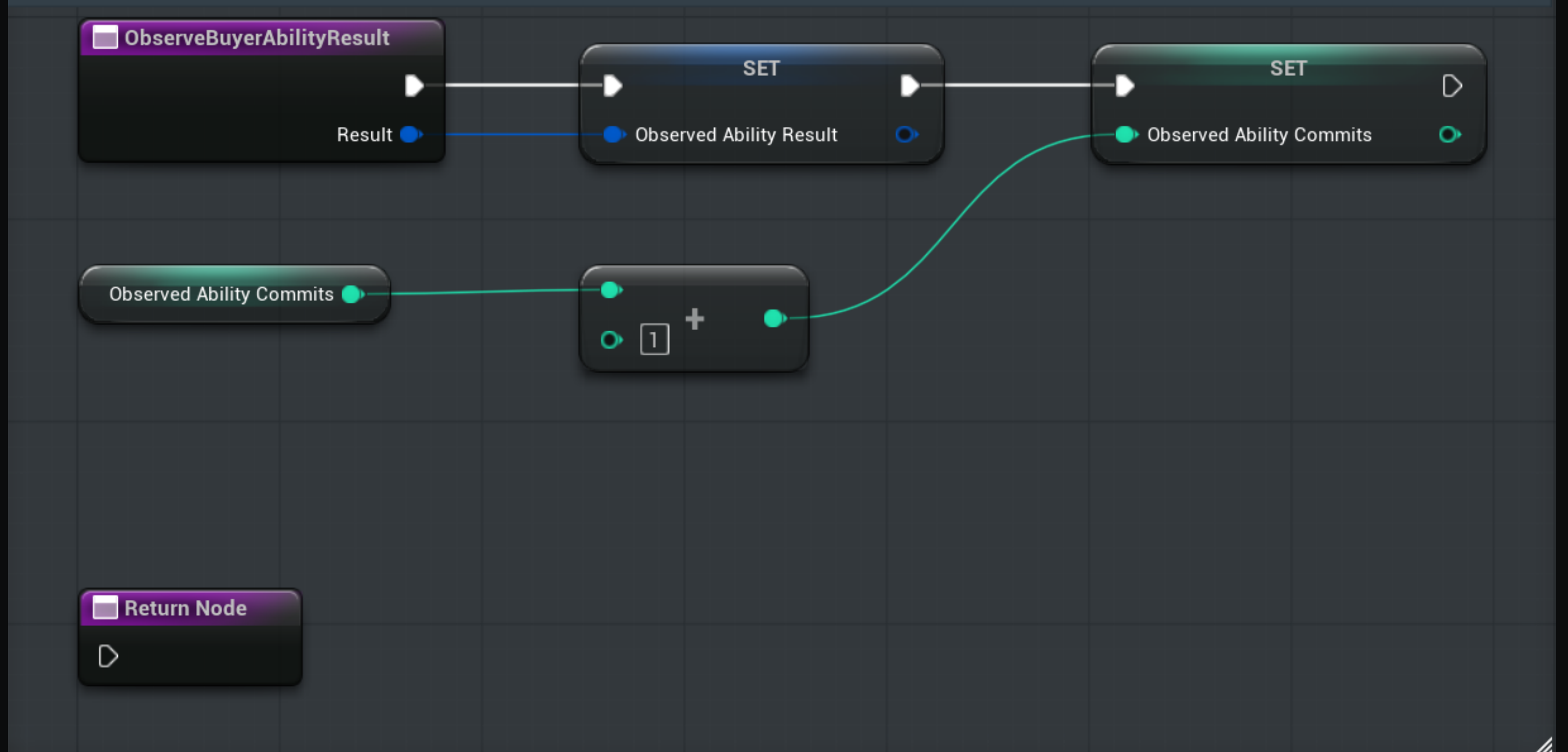
1. Keep the complete Result for HUD text, quest logic or analytics.
2. The observation counter counts committed notifications. A cooldown refusal does not create a committed notification.
3. Keep this graph observational: never request another attack, apply damage or remove stock here.
4. Disconnect the retained inventory owner when it ends. Reconnect explicitly to the correct current pawn when ownership changes.

Expected result: A result reports what happened. It is not a second command to do it.

/ActionBarInventoryKit/ActionBarInventoryKit/Examples/BP_ABIK_BuyerAbilityGameplay / ObserveBuyerAbilityResult

Observe Buyer Ability Result

1. One accepted ability sends its actual outgoing result after the melee effect.
2. Save the complete result and increment an observation counter for HUD, quest or analytics feedback.
3. Cooldown/refusal produces no committed notification. This graph never reapplies damage or removes stock.



Optional: read the ability result | [Open full-size original](#)

Punches and your character animation

The router and the character animation must both be connected for the intended motion.

1. Use the configured BP_ABK_SimpleAbilityRouter example for its saved punch clips. Configure damage, reach, hit radius and cooldown for your game.
2. The current motion route supports full-body punches while standing and a blend that preserves lower-body locomotion while moving.
3. Your character needs a compatible skeleton and Animation Blueprint using the punch animation integration. A custom mesh does not become compatible simply by assigning an action-bar slot.
4. The combined demonstration host has its own mannequin Animation Blueprint hookup. Treat that as host setup; do not assume your unrelated pawn automatically receives the same Anim Class.
5. Check stationary punches, walking punches and return to locomotion on your character. Your game also owns authority, replication and target health.

Expected result: Start from your character's existing locomotion and integrate the punch route deliberately. Keep only one automatic or inventory-owned attack route active.

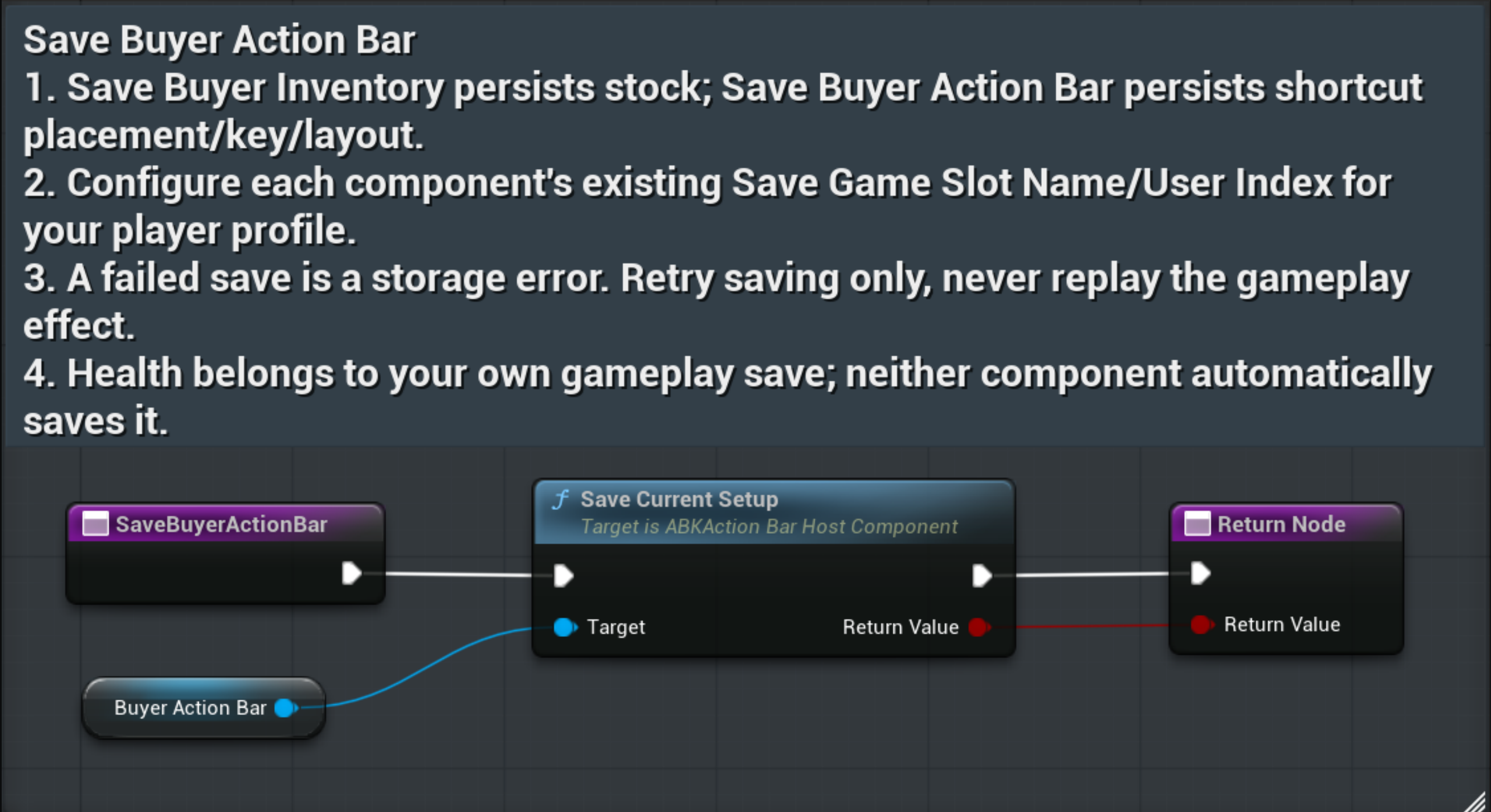
Save the action bar

Inventory and action-bar saves store different things. Give each its own Save Game Slot Name and the intended user index.

- 1. Save Buyer Action Bar calls Save Current Setup on BuyerActionBar. It stores bar setup such as shortcut placement, keys and layout.
- 2. Read the actual save result. If storage fails, show the error and retry saving only.
- 3. Do not retry the original healing, attack or equipment action because its later save failed.
- 4. Save character health, quests and other gameplay state in your own game save.

Expected result: Saving UI setup does not automatically save your character.

/ActionBarInventoryKit/ActionBarInventoryKit/Examples/BP_ABIK_BuyerInventoryActionOwner / SaveBuyerActionBar



Save the action bar | [Open full-size original](#)

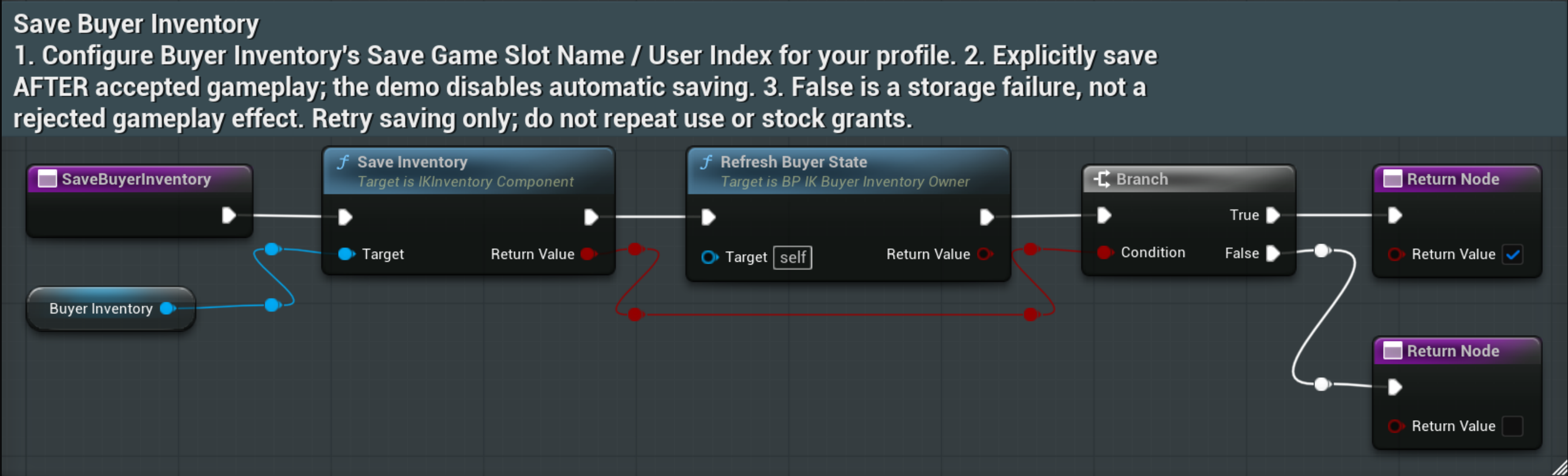
Save inventory stock

The inherited Save Buyer Inventory function uses the inventory persistence route.

- 1. Configure the inventory save slot separately from the action bar slot.
- 2. Call Save Buyer Inventory on the retained owner and check the returned result.
- 3. Keep stock, capacity and equipped-bag state on the inventory route. Coordinate your own gameplay save at a deliberate checkpoint.
- 4. Use separate demo/profile names during testing so a lesson cannot overwrite an existing player save.

Expected result: The shared Inventory screenshot applies to the parent of the combined owner.

/InventoryKit/InventoryKit/Examples/BP_IK_BuyerInventoryOwner / SaveBuyerInventory



Save inventory stock (shared Inventory guide reference) | [Open full-size original](#)

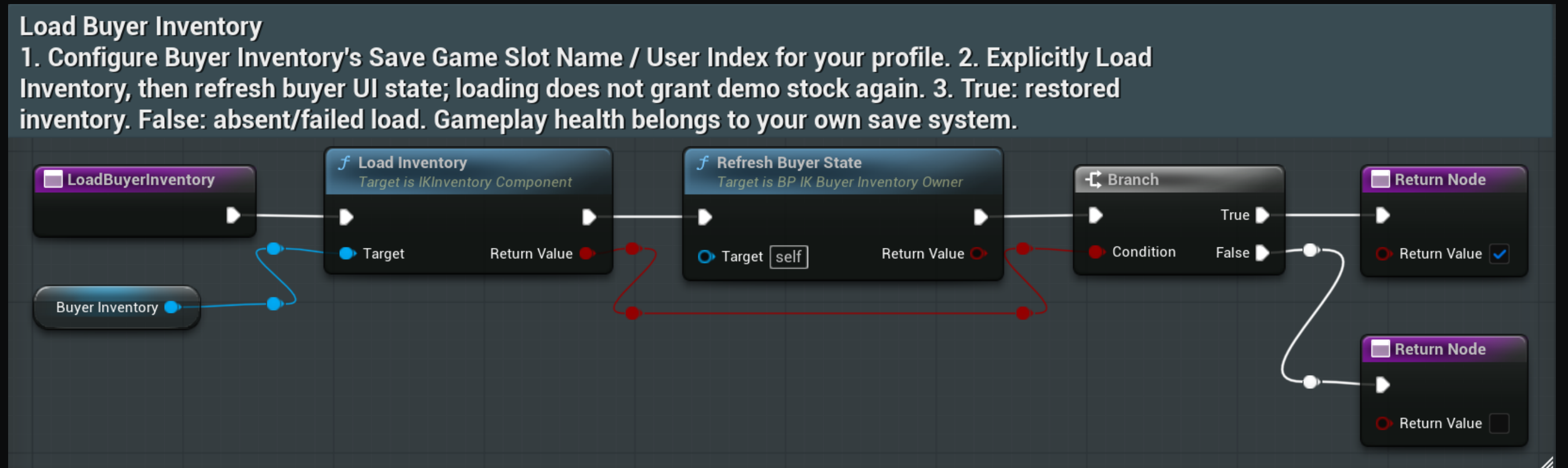
Load without replaying gameplay

Use the same slot name and user index used when saving.

1. Call Load Buyer Inventory and check its actual result before assuming saved stock is available.
2. Restore bar setup through its separate persistence route. Stable Item Ids let shortcuts identify restored inventory.
3. Load health and other gameplay state through your own game save.
4. Refresh displayed state after loading. Do not heal, damage, grant starting stock again or consume an item because a load notification arrived.

Expected result: Loading restores state. It does not replay the actions that originally created that state.

[//InventoryKit/InventoryKit/Examples/BP_IK_BuyerInventoryOwner / LoadBuyerInventory](#)



Load without replaying gameplay (shared Inventory guide reference) | [Open full-size original](#)

Advanced: who owns the bar?

Most combined integrations only need inventory shortcuts. A full action-bar provider is for a game that wants to supply the whole bar.

1. Keep the combined provider when using Assign Inventory Item Shortcut. Do not install a full replacement provider just to add one inventory item.
2. For a deliberate custom provider, retain it for the bar's lifetime. Get Action Bar Slots returns the presentation snapshot; Can Trigger Action Slot checks readiness without side effects.
3. Trigger Action Slot runs the accepted action once. Recheck readiness there and return its actual result.
4. Disconnect only your own provider and callbacks. Do not take over a different active owner silently.
5. For native integration, add the modules you use to Build.cs: ActionBarInventoryKit, ActionBarKit and InventoryKit. Read the public headers for the current signatures. The normal editable examples do not require changing plugin C++.

Expected result: A provider is the object that answers the bar's questions and handles requests. Decide which object owns that responsibility before connecting it.

If something does not work

Start with the first matching symptom.

1. Two inventories or inconsistent counts: automatic player startup and the placed teaching actor may both be active. Choose one owner and reconnect the shortcut to that inventory.
2. J does nothing: check Auto Receive Input = Player 0, input focus, Initial Item Asset, and the actual shortcut failure. Close a UI that is consuming the key.
3. Item is visible but cannot be used: check stock, cooldown, gameplay connection, correct Primary Action, and the unavailable reason. Full health is an expected refusal in the lesson.
4. An effect happens twice: look for both an actor key event and a bar binding, or an effect replayed from a result listener.
5. A bag will not come off: move items into the capacity that will remain after removal.
6. Old quantities return: check the save slot/profile being loaded. Use a separate fresh demo slot instead of deleting a player save.
7. Punch motion is wrong: check your pawn skeleton, Animation Blueprint and punch blend hookup. A functioning damage request does not prove animation integration.

Expected result: Read the real failure message before changing the graph. Fix ownership or input first; do not add another effect call as a workaround.

Before shipping your game

Use these checks on your own final game and content.

1. Verify a fresh player and a returning saved player. Test accepted use, refusal, empty stock, cooldown, disconnect and reconnect.
2. Check bags at the capacity limit, save/load, and the same potion used from both shortcut positions. Confirm stock cannot duplicate or disappear.
3. Include your item definitions, icons, meshes, animation assets and dependencies in the packaged game. Verify soft-referenced content is cooked through your project's asset/cook setup.
4. Keep the plugin's content layout intact. Its shared /ActionBarKit/ and /InventoryKit/ asset paths are part of the combined installation.
5. Test keyboard, mouse and your intended controller mappings in the packaged game. Confirm custom cursor/focus restoration.
6. Verify multiplayer authority and replication in your own implementation if your game needs them. The local teaching examples are not a complete multiplayer inventory or ability architecture.

Expected result: Editor success alone is not a packaged-game or release approval.

Quick glossary and support

The terms used in the graphs are simpler than they sound.

1. Owner: the actor or controller whose inventory and action bar you are using.
2. Item Id: the stable name used to identify an item across stock, shortcuts and saves.
3. Provider / gameplay object: the retained object that supplies actions or performs the game effect.
4. Accepted: the actual operation succeeded. A request alone is not success.
5. Observer / callback: a listener that reads a change after it happens. It should not repeat the effect.
6. Support: support.blackironspine@proton.me. Include product version, Unreal version, steps to reproduce, the actual failure text and relevant logs.
7. Our initial-response target is 24 hours. This is not a guaranteed resolution time.

Expected result: Black Iron Spine Studios