ASHENFORGE

UIForge

Build game interfaces in Blueprint

Menus, HUD composition, settings, controls and presentation on Unreal CommonUI.



Emberfall MMO example - actual Unreal Engine capture

Version 0.1.0 - Development documentation

UE 5.8.3 / Win64 reference implementation. Read Current product status for tested scope and remaining limits. Engine images match the web guides; use the web viewer at actual size for small Blueprint pins. Mermaid workflows are included as text source in this print companion.

Contents

Start here

Overview
Build a game UI · pictures
Build settings · pictures
Customize · pictures
5.8.3 product status
Current example gallery
Implement in your game
Product Blueprint authoring
Code and Blueprint scheme
Installation
Blueprint quick start
Recipe index
Playable interfaces
Customization

Menus and navigation

Screens and HUD layers
Menu composition
Menu data
Focus and back navigation
Navigation hints
Dialogs
Loading
World travel

Controls and settings

Automatic input switching
Button glyphs and prompts
Build a settings menu
Settings and audio
Localization

Presentation

Themes and transitions
Button and resource artwork
Responsive layouts
Safe areas
Scrolling
Readout layouts
Tooltips
Notifications
Progress variants
Center-origin cooldown tint

Data and reusable entries

Presentation sources
Journal and list entries
Field cards and tile entries
List, tile and tree boundaries

World-space UI

Game interfaces and adapters

RPG HUD and rebinding
Build an ability browser
RPG research and decisions
Army and RPG HUD authoring
Example world presentation
Example character animation
Historical scout recipe
Historical RPG recipe
Optional MassForge adapter
Migration

Version and verification

Historical visual catalog
Competitive quality review
Current 5.8.3 verification
Artwork and dependencies
Release plan and feature matrix
Engine compatibility
Historical shipping verification
Historical feature matrix
Acceptance and limitations

API reference

API and result handling

START HERE

UIForge overview

Learn with real editor images

Follow Build your first game interface, Build a tabbed settings menu, and Blueprint customization. Each route shows actual asset locations, Designer selections, fields or graph nodes, the action to take and the expected result. Images enlarge with a click or keyboard Enter.

Start with a working flow

Open **ForgeGateway** in the standalone UE5.8.3 project. Choose Emberfall or Ironfront, then use **Continue / New Game / Load Game → playable encounter → pause → settings / save / load → main menu**. The current gallery contains real Unreal captures of these saved Blueprint interfaces.

The UI is built through public features: CommonUI screen stacks, named HUD layout profiles, reusable entries, commands, themes, input-aware prompts and adapters. Encounter gameplay and local saves belong to the examples. The core has no MassForge or inventory dependency.

Installation and playable routes gives the project paths and setup. Blueprint recipes starts your own interface with **Get Local Player Subsystem → Create Root → Push Screen**.

Make it your own

Start with product Blueprint authoring: replace a HUD region, a compact frame, an ability button, a layout profile or a theme. The separate Blueprint tutorial demonstrates custom screens, entries and data sources; Emberfall uses its live quest adapter for the playable journal. Themes and animation covers live presentation and UMG timelines. Center-origin tint documents the independently reproduced Sakura cooldown behavior separately from other progress styles.



Emberfall running in Unreal Engine 5.8.3 after the environment cleanup

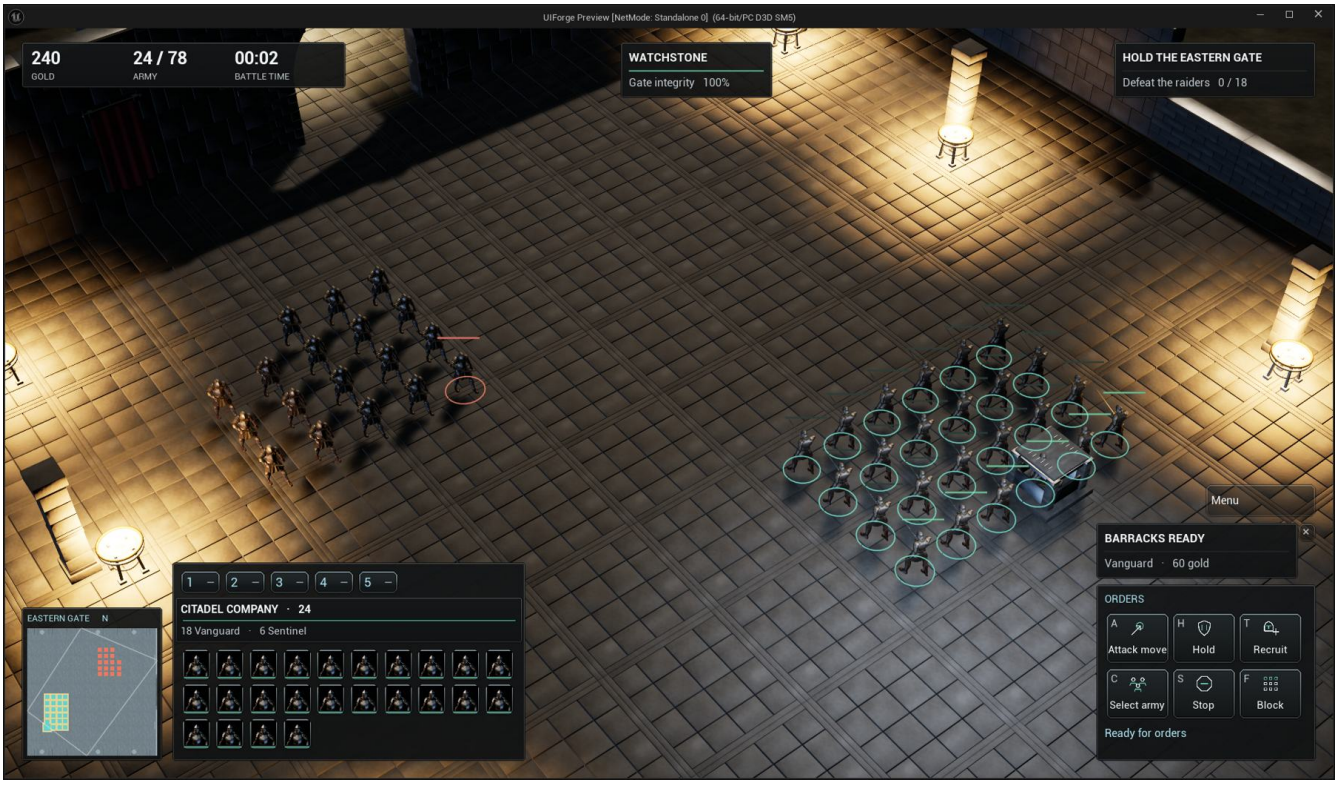
Development status

This is **0.1.0 development documentation**, maintained with implementation. The current product rebuild targets **UE5.8.3 CL58210709 / Win64**. Read current status for completed checks and limitations. Older gallery/three-engine records describe historical revisions and do not certify the rebuilt examples on other engines.

Physical controller testing is deferred by the user. Software input tests do not certify controller hardware, touchscreens or other platforms. Game Master's visual design is exclusive; these examples use original layouts and assets. Inventory mechanics are outside InterfaceForge.

Author the current visual system

Use named button skins, resource artwork and shared window surfaces, the complete multi-tab settings recipe, and the working ability-browser recipe. Both playable examples use these same public controls. Original editable artwork and independent core themes are included; reference-game and competitor designs are not copied.



Current army selection interface built with the same public toolkit

START HERE

Build your first game interface

Follow this visual route to add a main menu, HUD, custom Blueprint entry, pause screen and settings to your own game. Each editor image shows a real saved asset in Unreal Engine 5.8.3. Click or keyboard-focus an image and press Enter to enlarge it. Choose **Actual size · 100%** and scroll to read small property names and graph pins; Escape closes the viewer.

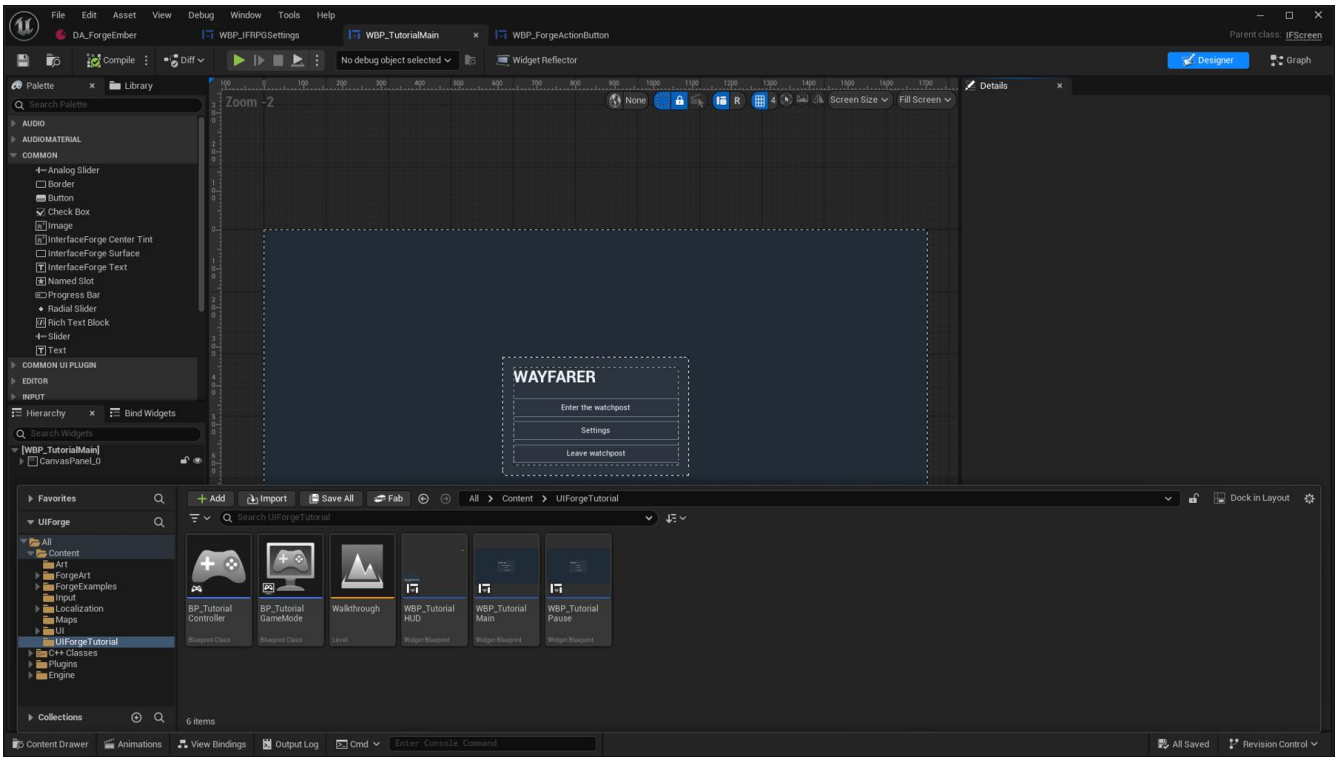
UIForge is the product name. **InterfaceForge** remains the plugin/module name and **IF** is the class prefix. Your game owns gameplay, possession and saves. The plugin needs CommonUI/CommonInput and Enhanced Input; it does not need MassForge or an inventory plugin.

1. Find the learning assets

Install `InterfaceForge` in `<YourGame>/Plugins/`, enable it with CommonUI and Enhanced Input, and reopen/build the project with the matching engine. These instructions target the verified UE 5.8.3 / Win64 source revision. See installation for toolchain and compatibility limits.

In the Content Browser, enable **Show Plugin Content** from its Settings menu. Core themes live at `/InterfaceForge/UIForge/Themes`. The example project adds `/Game/UIForgeTutorial` and `/Game/ForgeExamples`; these folders are not a dependency of the core plugin.

Open `/Game/UIForgeTutorial/Walkthrough` to inspect the small Blueprint integration. Keep your existing PlayerController/GameMode when adapting an established game. For a new learning map, assign `BP_TutorialGameMode` in **World Settings → GameMode Override**.



Content Drawer at Content → UIForgeTutorial showing the Walkthrough map, controller, GameMode and three screen Blueprints

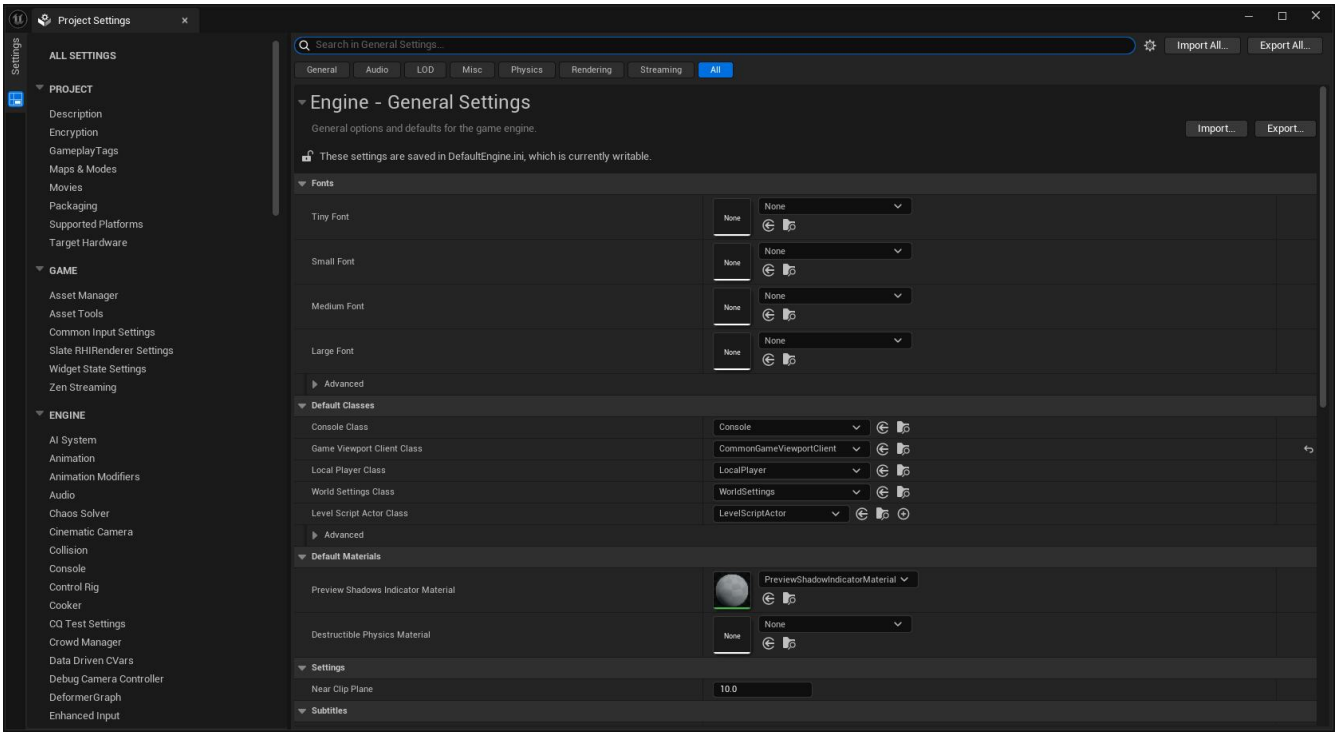
Press Ctrl+Space to open the Content Drawer. Select UIForgeTutorial in the folder tree, then double-click Walkthrough. These six assets form the small learning flow.

The full playable games are separate references: Emberfall and Ironfront. The learning map is intentionally small so its public Blueprint flow remains easy to follow.

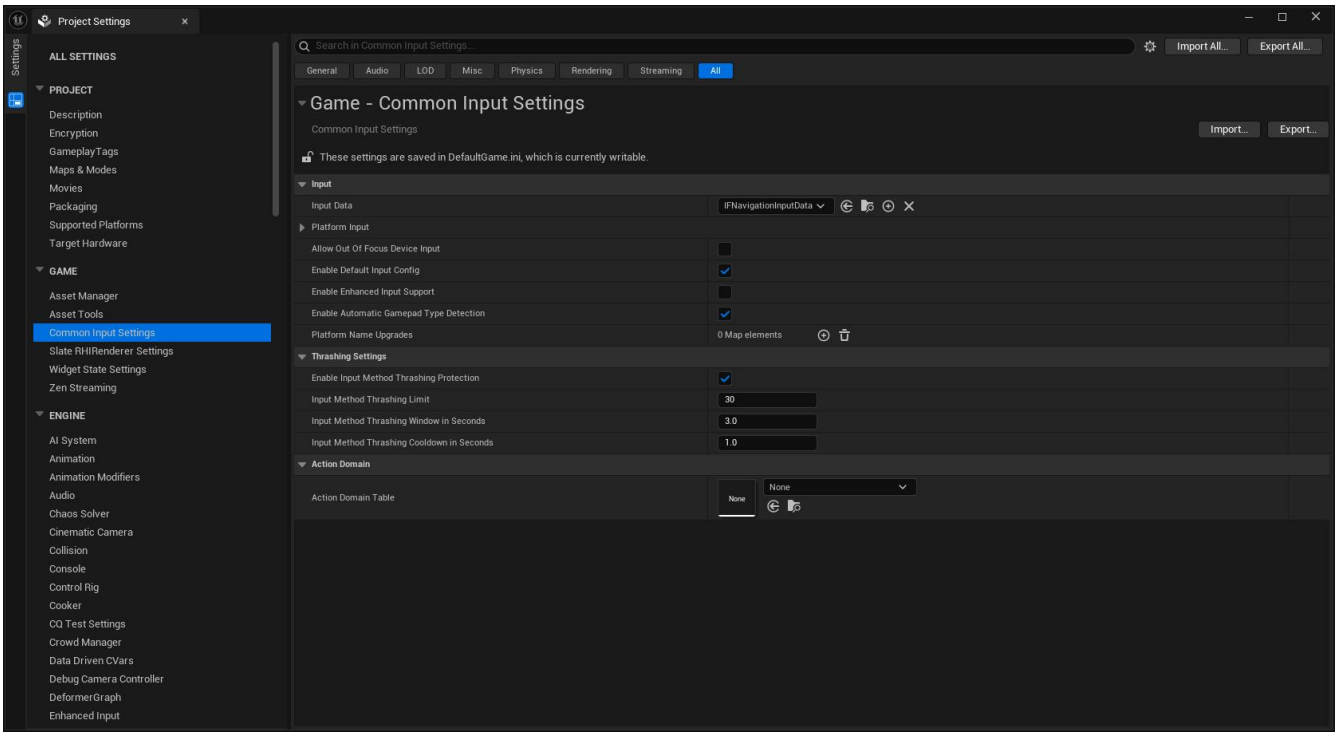
2. Set up CommonUI before building menus

In **Edit → Project Settings**, configure:

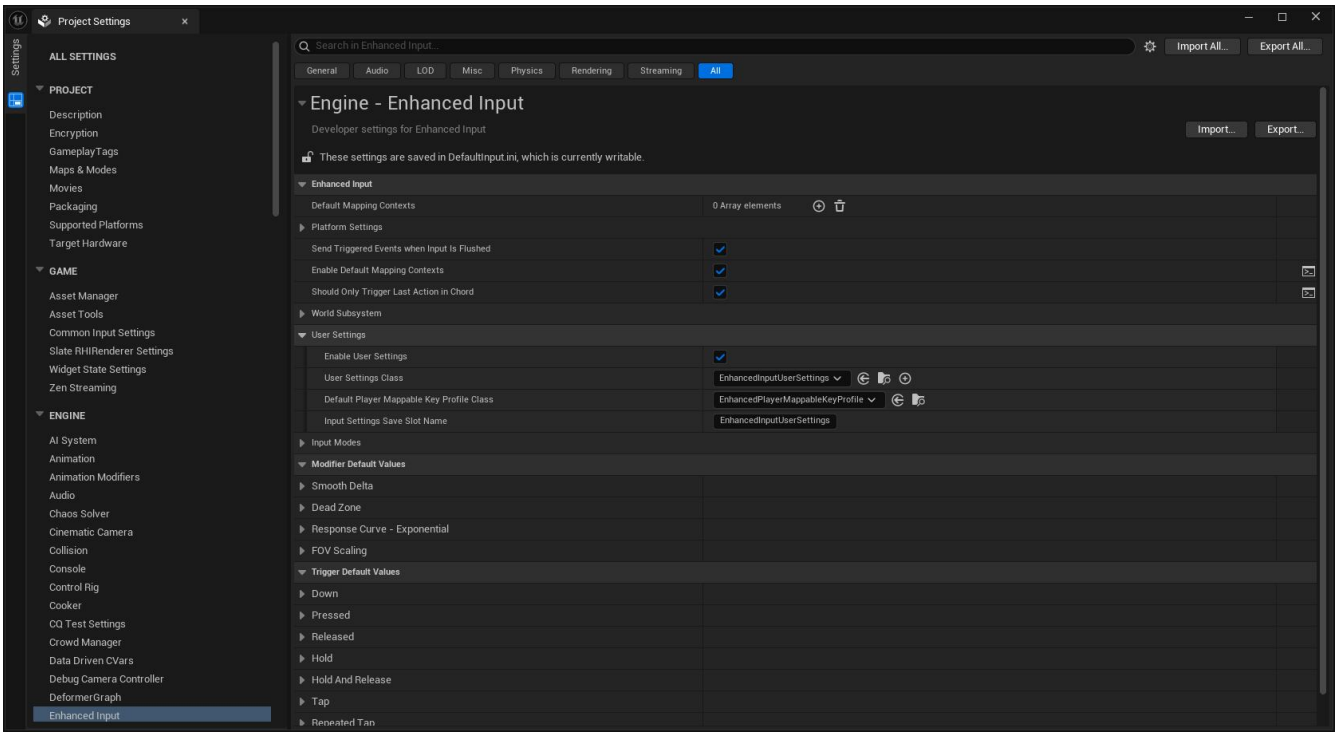
Find	Set
Engine → General Settings → Game Viewport Client Class	CommonGameViewportClient
Common Input Settings → Input Data	IFNavigationInputData or your compatible CommonUI input-data class
Plugins → Common UI Framework → Common Button Accept Key Handling	Trigger Click
Enhanced Input → Enable User Settings	Enabled for player-mappable bindings
Input → default player-input/component classes	EnhancedPlayerInput / EnhancedInputComponent when using Enhanced Input



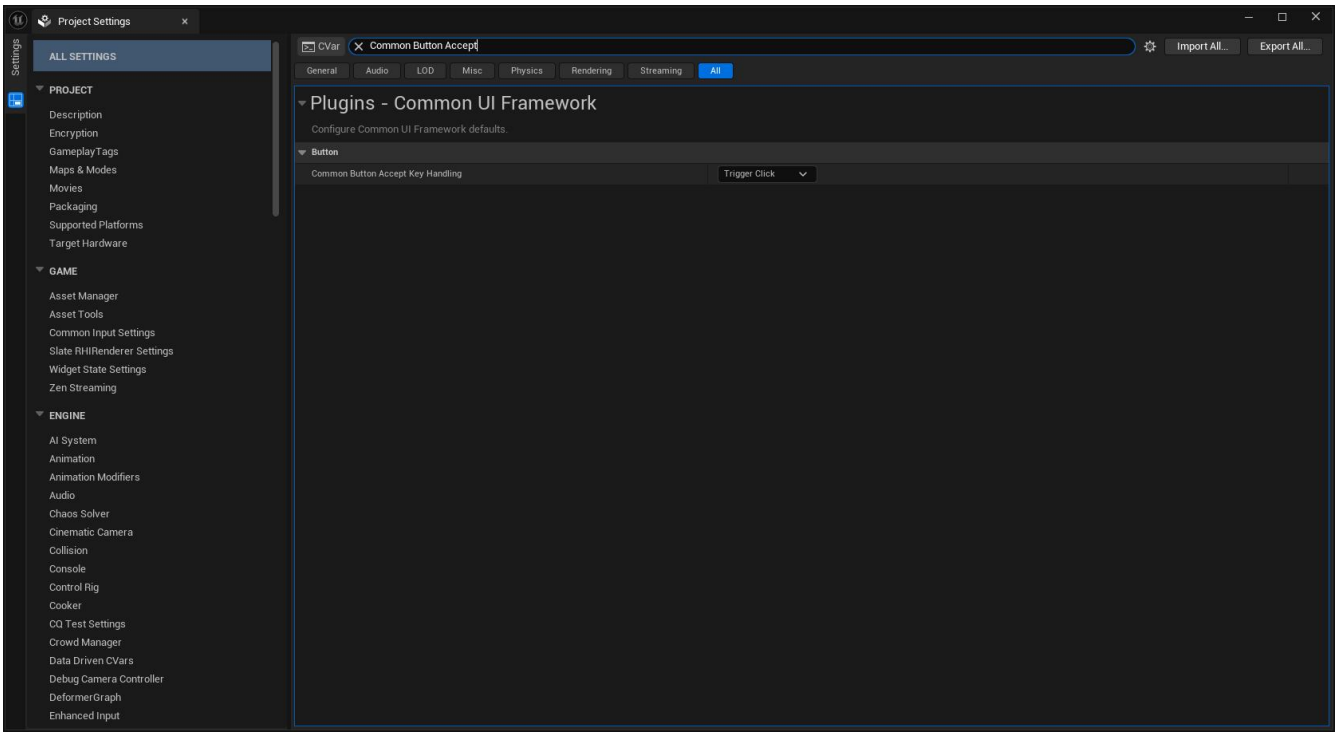
Game Viewport Client Class set to CommonGameViewportClient in the real Project Settings window



Common Input Settings showing the IFNavigationInputData selection



Enhanced Input with User Settings expanded and Enable User Settings checked



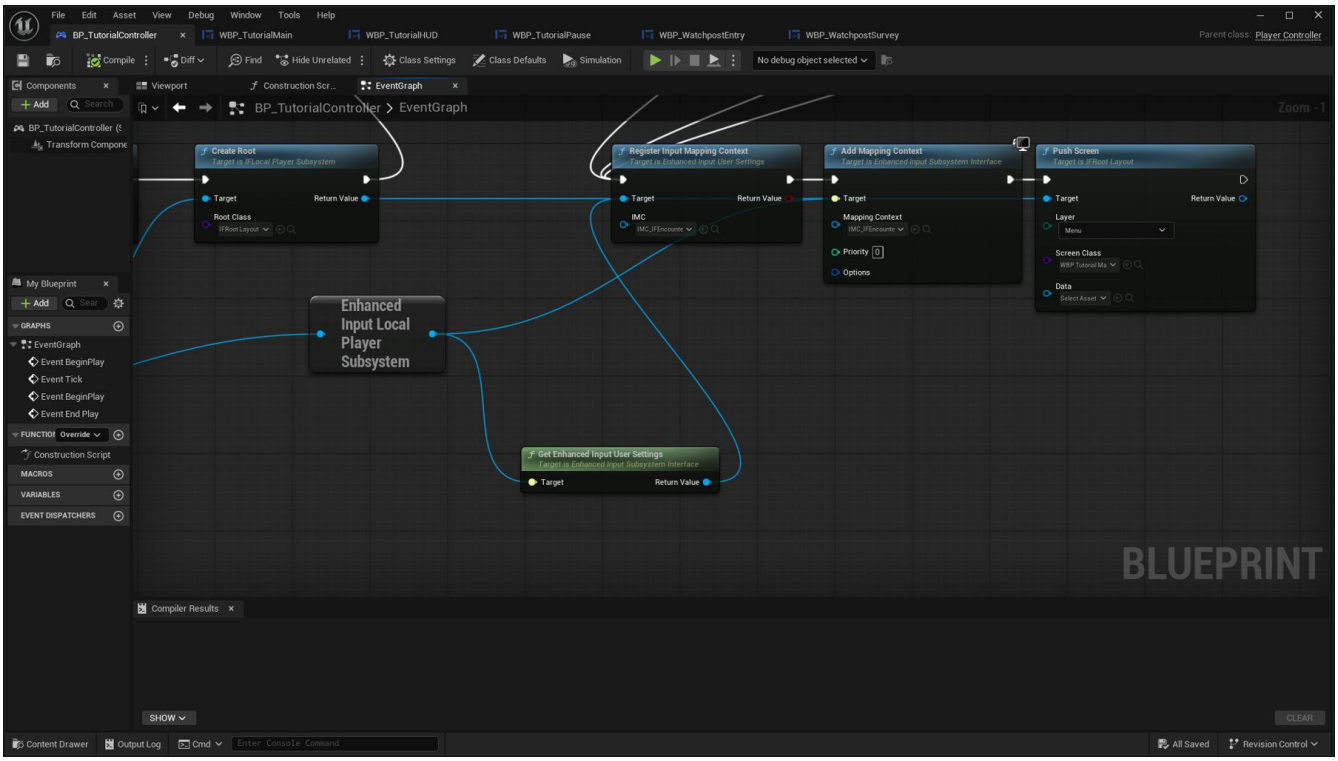
Common UI Framework showing Common Button Accept Key Handling set to Trigger Click

The full integration guide explains the existing-engine responsibilities and registration order. Keyboard/controller prompts should use real mapping names from your Enhanced Input context. They should not be separate strings that become stale after remapping.

3. Create one local-player root

Open: `/Game/UIForgeTutorial/BP_TutorialController`, **EventGraph**.

In the local controller's **Event BeginPlay**, branch on **Is Local Controller**. Get **IFLocal Player Subsystem**, call **Set Theme**, and call **Create Root** with **Root Class = IFRootLayout**. Use the returned root for **Push Screen**, **Layer = Menu**, **Screen Class = your IFScreen Blueprint**.



Actual tutorial controller graph with Create Root, Enhanced Input registration and Push Screen on the Menu layer

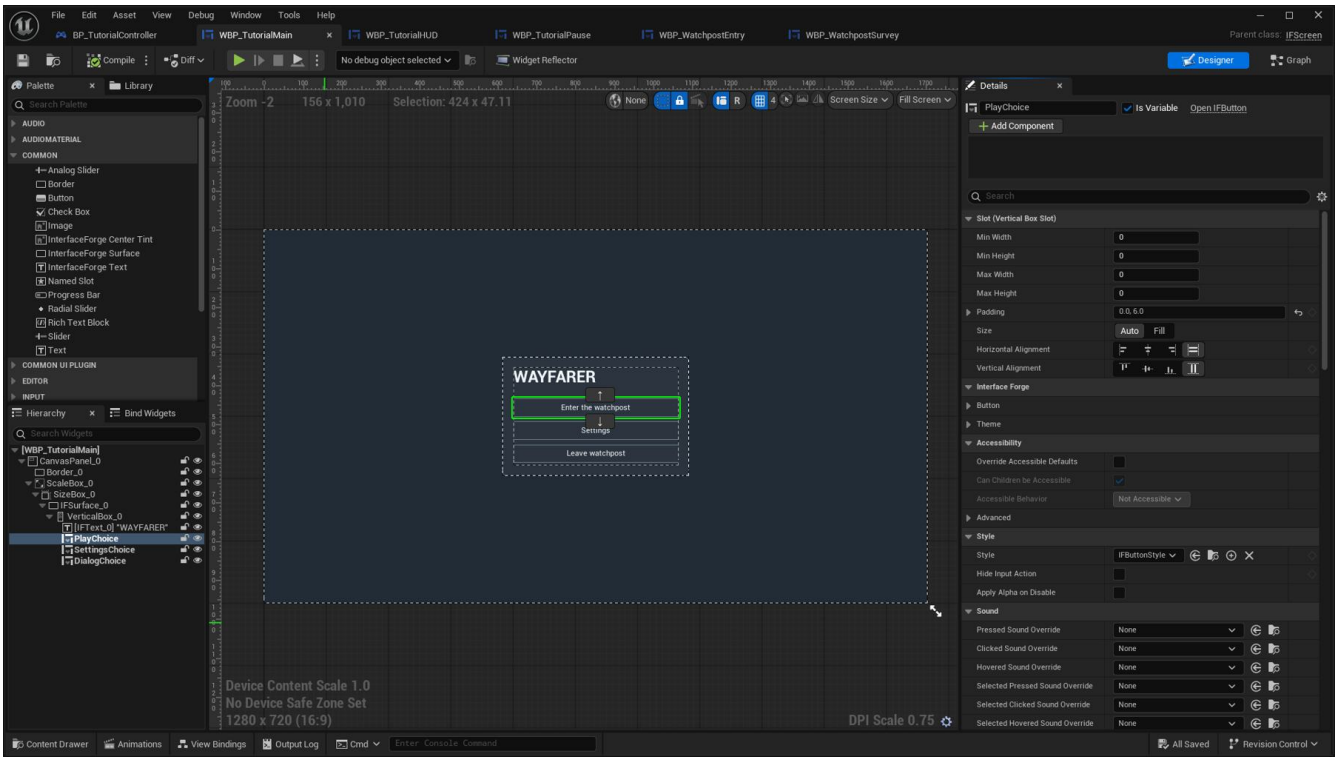
Notice the blue Return Value wire from Create Root to Push Screen's Target. Register Input Mapping Context belongs to Enhanced Input User Settings; Add Mapping Context belongs to the Enhanced Input local-player subsystem. Registration makes mappings available for rebinding; adding the context enables gameplay input.

Expected: one root for this local player and one menu. Keep your existing mapping priorities. Do not call Clear All Mappings or create a new viewport root every time a screen opens. On EndPlay or world travel, **Remove Root** and let the next local controller initialize its own interface.

4. Design the menu in Blueprint

Open: `/Game/UIForgeTutorial/WBP_TutorialMain`, **Designer**. Its parent is **IFScreen**. For your own screen, create a Widget Blueprint and choose IFScreen in the parent-class picker.

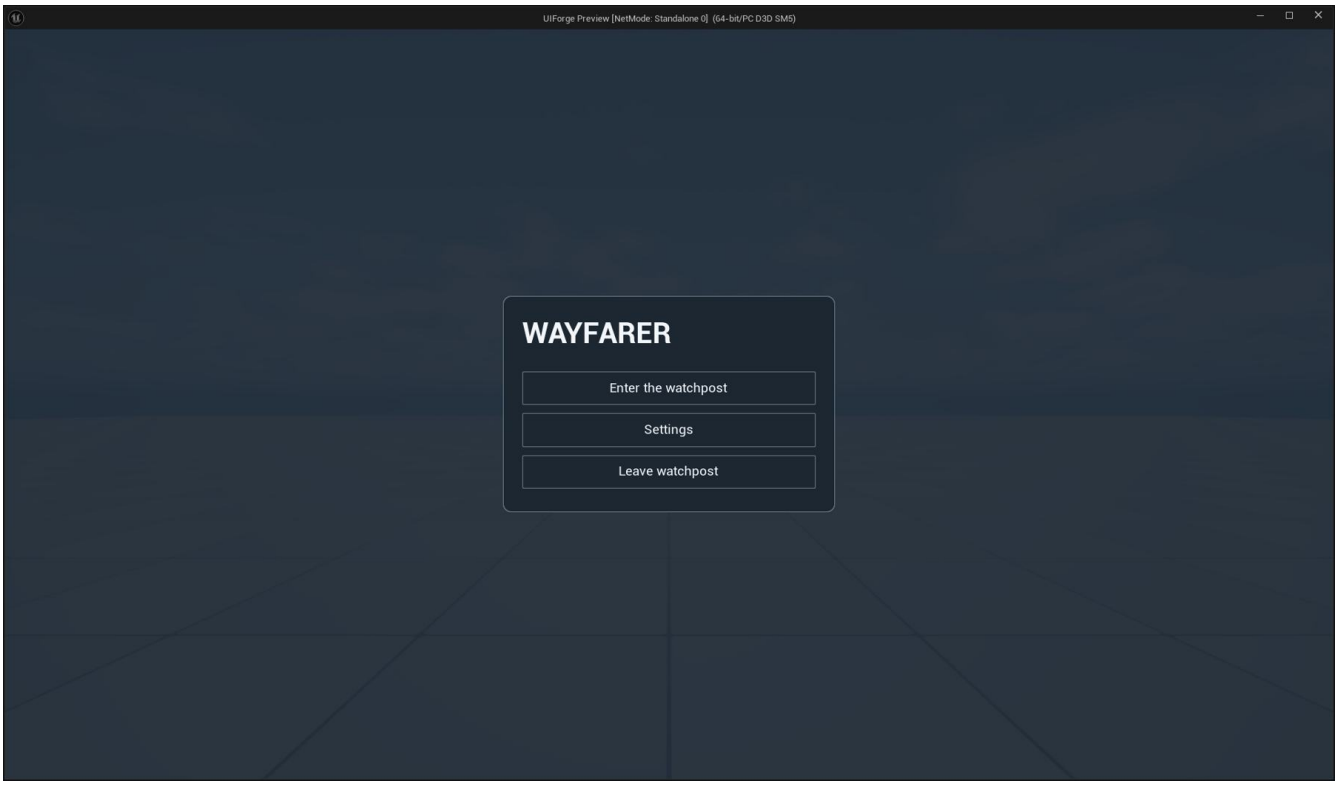
Build the background/layout with ordinary UMG panels. Add **IFButton** controls with useful Designer names, such as `PlayChoice` and `SettingsChoice`. Select a button and set **Interface Forge** → **Button** → **Label**. Native fallback labels are visible in Designer; a Blueprint-owned content tree can replace them.



Tutorial main menu Designer with PlayChoice selected in the hierarchy and its native label visible

The left Hierarchy selects the actual control. The right Details panel edits its label and presentation. The centered preview is this small learning screen; your game can use a different composition and theme.

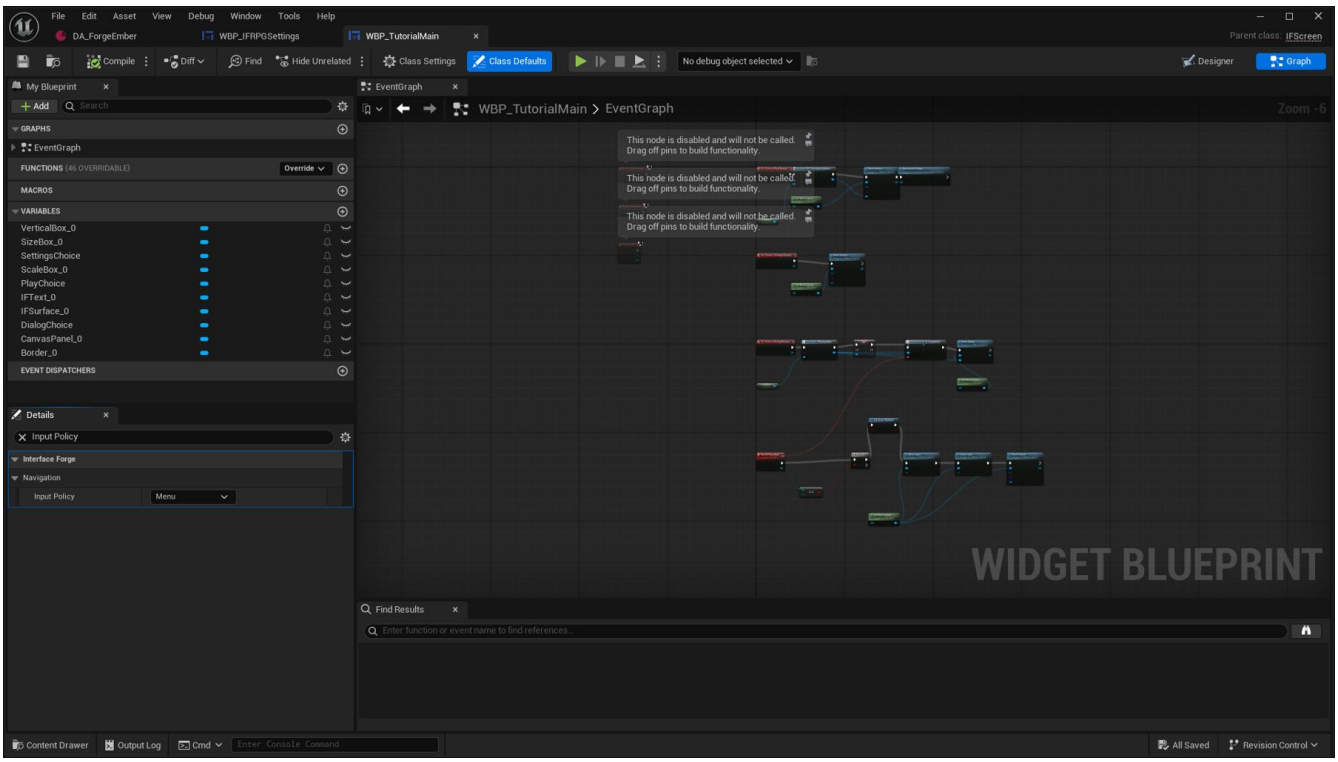
Expected: the same widget is editable in Designer and clickable in Play. Decorative images should be hit-test invisible so they do not cover buttons.



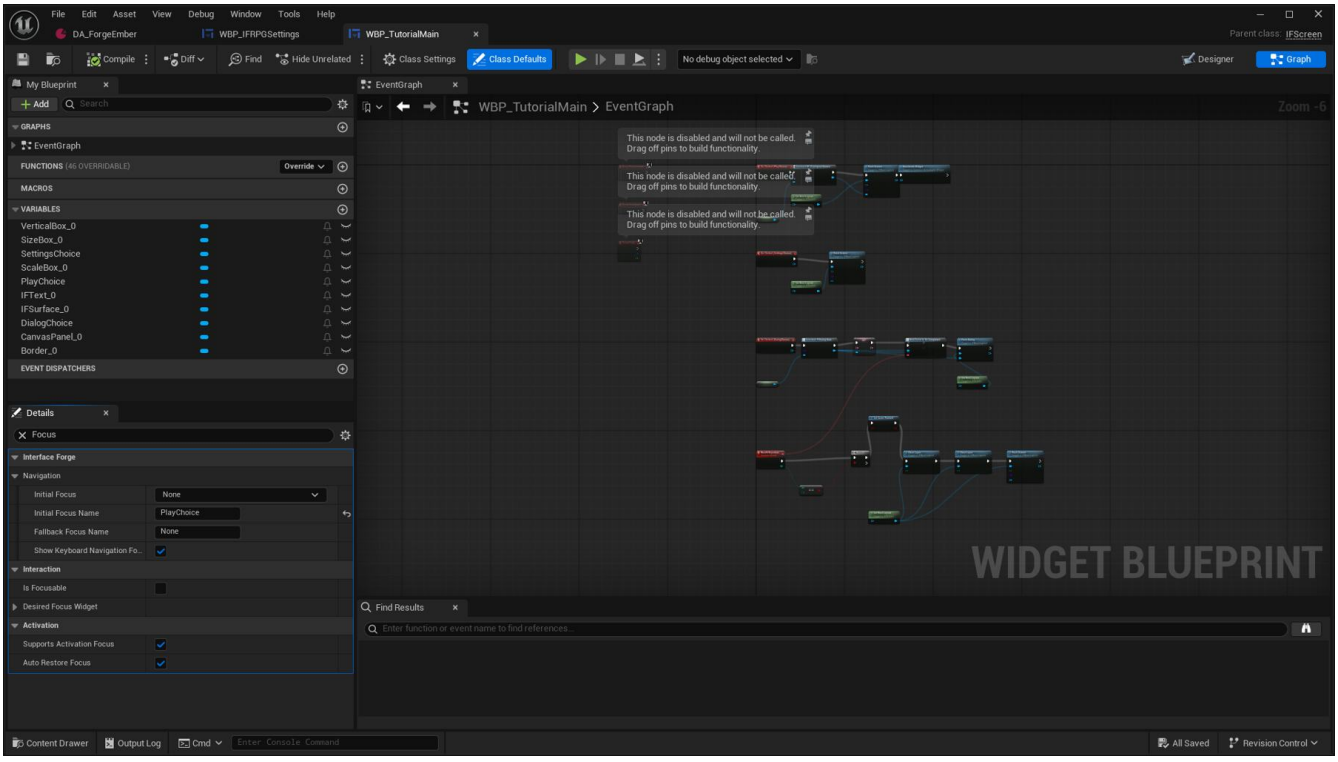
The same learning menu running in Unreal, with the native buttons using its assigned theme

5. Assign input ownership and initial focus

Choose **Class Defaults**. Set **Input Policy = Menu**. Set **Initial Focus Name = PlayChoice** and **Fallback Focus Name = SettingsChoice**; these are Designer widget names, not the label text.



Tutorial screen Class Defaults for its Menu input policy



Tutorial screen focus defaults with Initial Focus Name set to PlayChoice and the Fallback Focus Name field below it

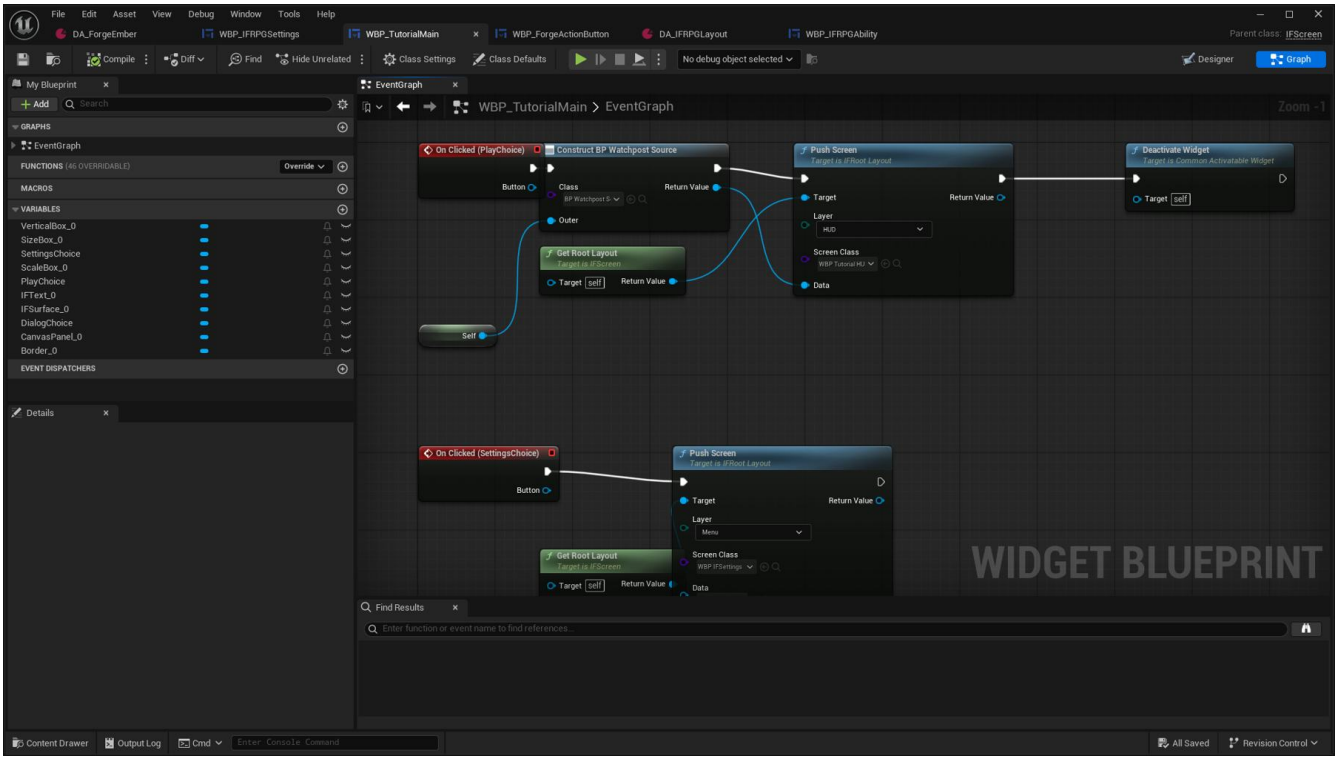
The saved asset uses PlayChoice for initial focus and leaves fallback unset. Assign SettingsChoice as a fallback in your own screen when PlayChoice may be hidden or disabled.

Expected: keyboard Accept activates a focused button, Back follows the top screen's policy, and returning from a popup restores useful focus. CommonUI owns routing. Do not add a competing Set Input Mode loop on every tick. See input

switching for the shared detector and focus/navigation for screen contracts. Physical controller/touch testing is still deferred.

6. Open a HUD through the same root

Select `PlayChoice`, add its **On Clicked** event (`On Clicked (PlayChoice)` in the graph), and open the Graph. Use **Get Root Layout → Push Screen**, **Layer = HUD**, **Screen Class = your HUD**. Pass a data source on **Data**. Deactivate the main screen after the HUD is pushed.

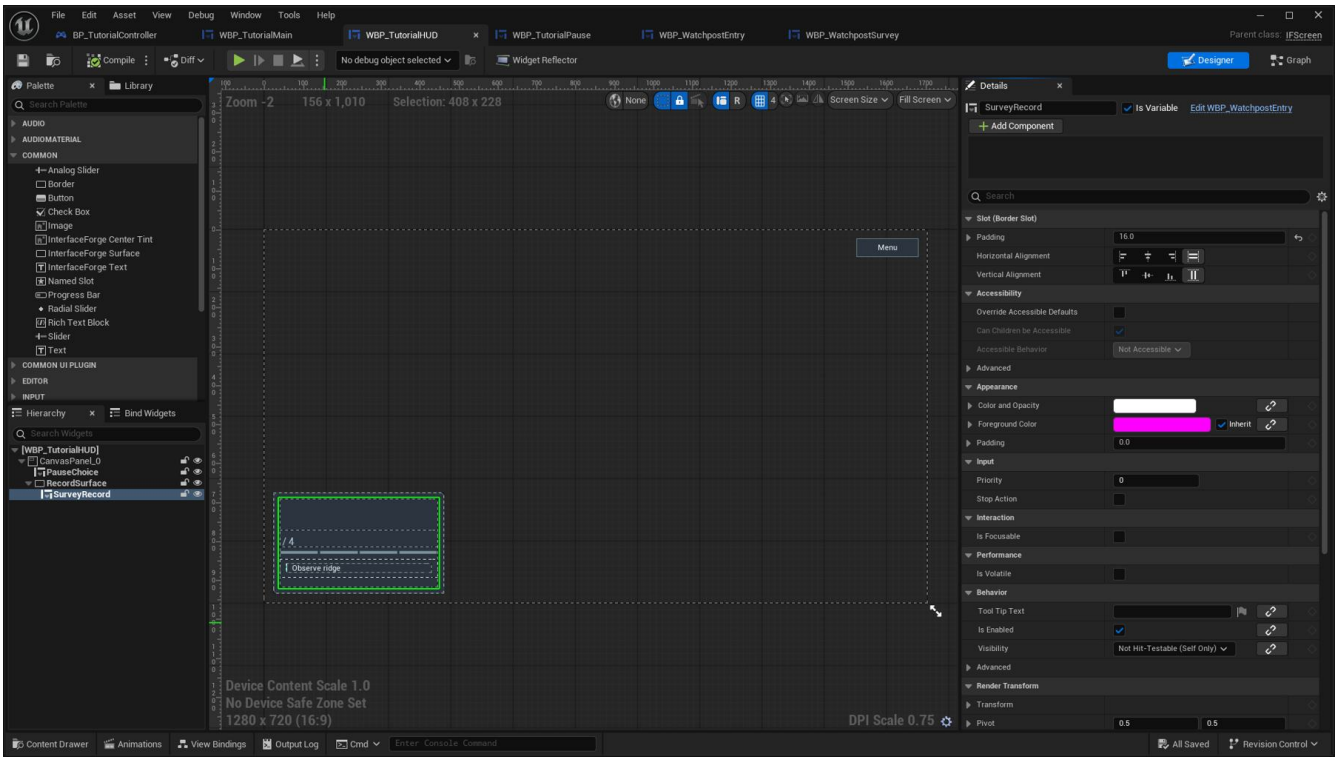


Tutorial PlayChoice graph pushing the HUD with a Blueprint source object as Data

Expected: your game continues underneath the HUD, while menus still use their own layer. The data object belongs to your game; it does not move gameplay rules into the UI plugin.

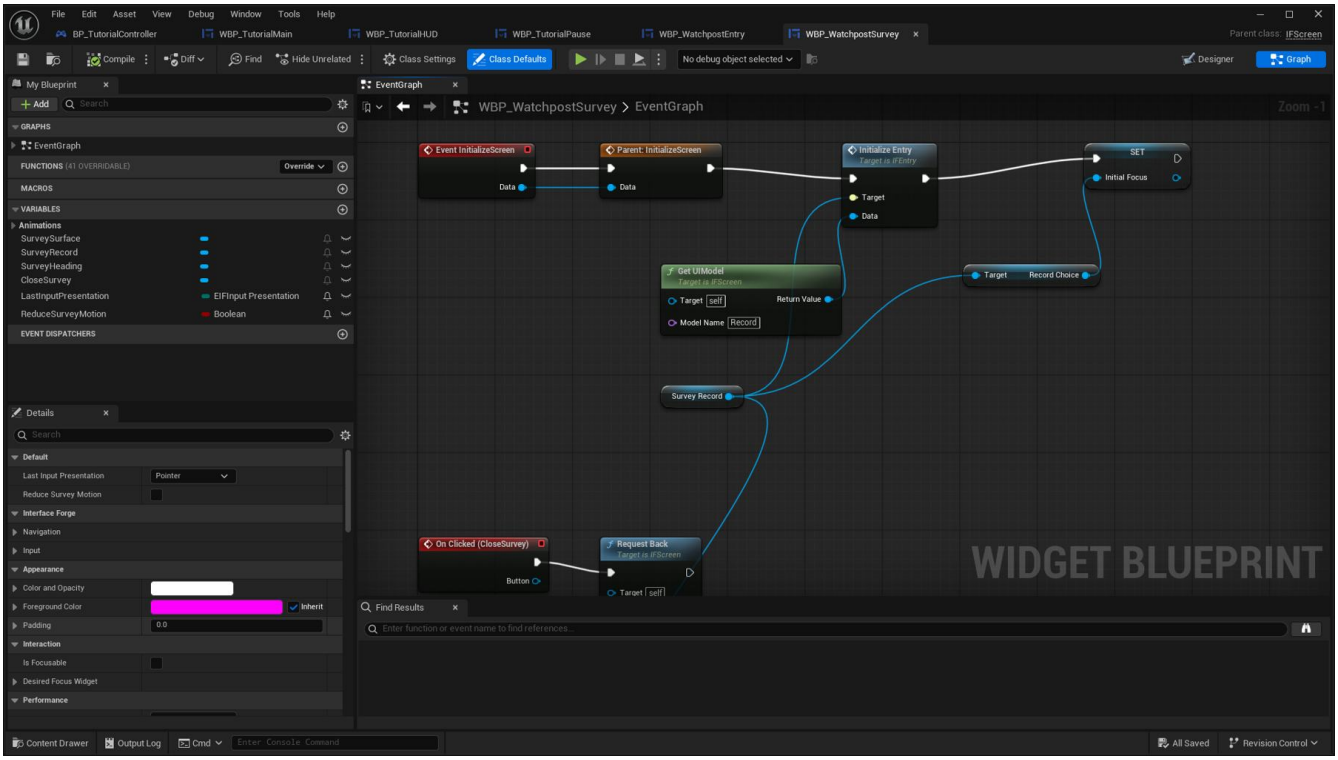
7. Create a custom entry and connect its data

Open: `/Game/UIForgeTutorial/WBP_TutorialHUD`, **Designer**. `SurveyRecord` is an instance of `/Game/UI/Blueprint/WBP_WatchpostEntry`, a Blueprint child of **IFEntryWidget**.



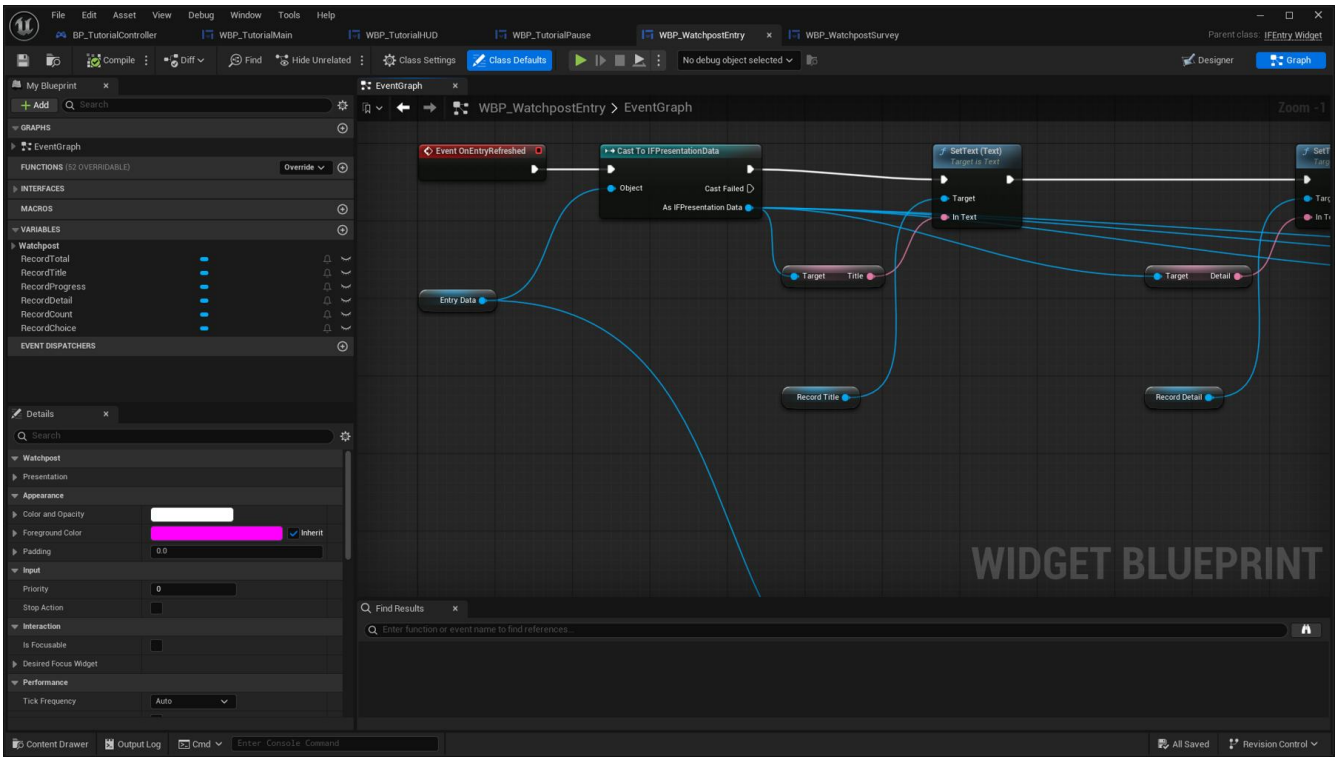
Tutorial HUD Designer showing its custom SurveyRecord entry

Your screen's **Initialize Screen** override must call parent first. Use **Get UIModel** for a stable model and **Initialize Entry** to provide its data/context to the custom entry.



Watchpost screen graph calling parent Initialize Screen, getting the model and initializing its custom entry

In the entry, implement **OnEntryRefreshed**. Read **Entry Data**, cast to the model your project supplied, and update text/progress. A model update should call **Notify Data Changed**; the entry should not scan the game's population or allocate new records every frame.



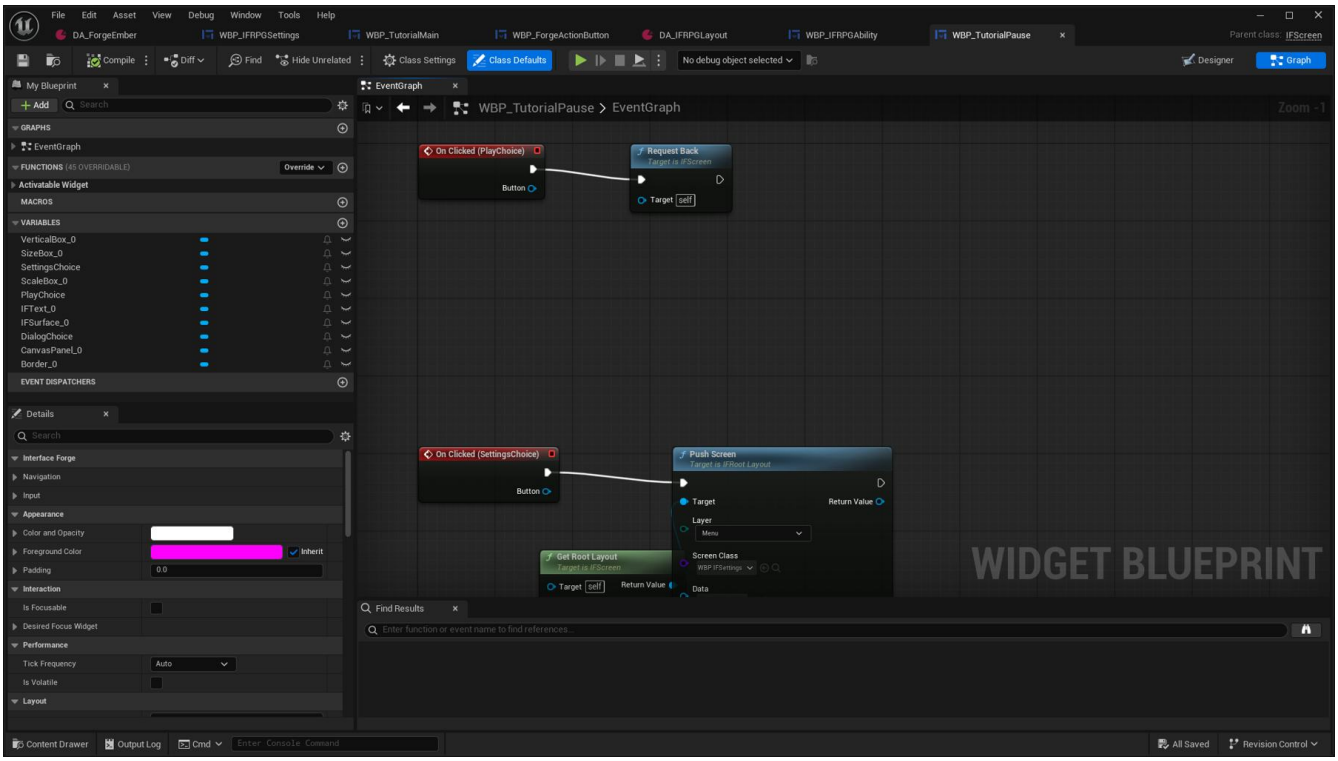
First part of the custom entry refresh graph: Entry Data is cast to IFPresentationData, whose Title feeds SetText on RecordTitle

This detail shows the Title connection. Continue along the saved graph for Detail, quantity and progress; use the same notification-driven refresh for your own fields.

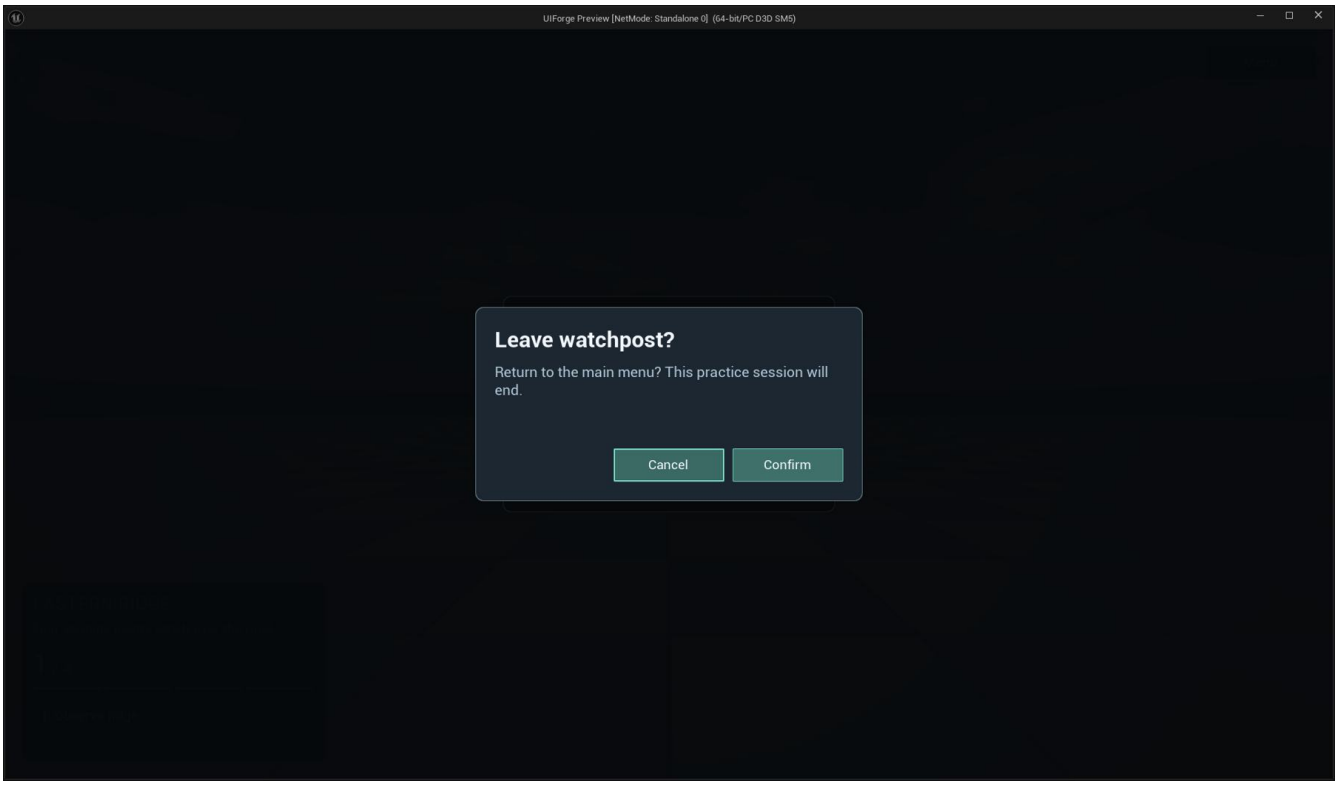
Expected: clicking the learning entry requests an action from the Blueprint source; the visible count changes through the model notification. The entry lifecycle recipe covers selection, recycling, subscription cleanup and action context.

8. Add pause and settings

Open: `/Game/UIForgeTutorial/WBP_TutorialPause`, **Graph**. Its buttons use the existing root to resume, push the settings screen, or ask for confirmation before returning. Your game owns whether simulation pauses; the menu owns presentation/focus.

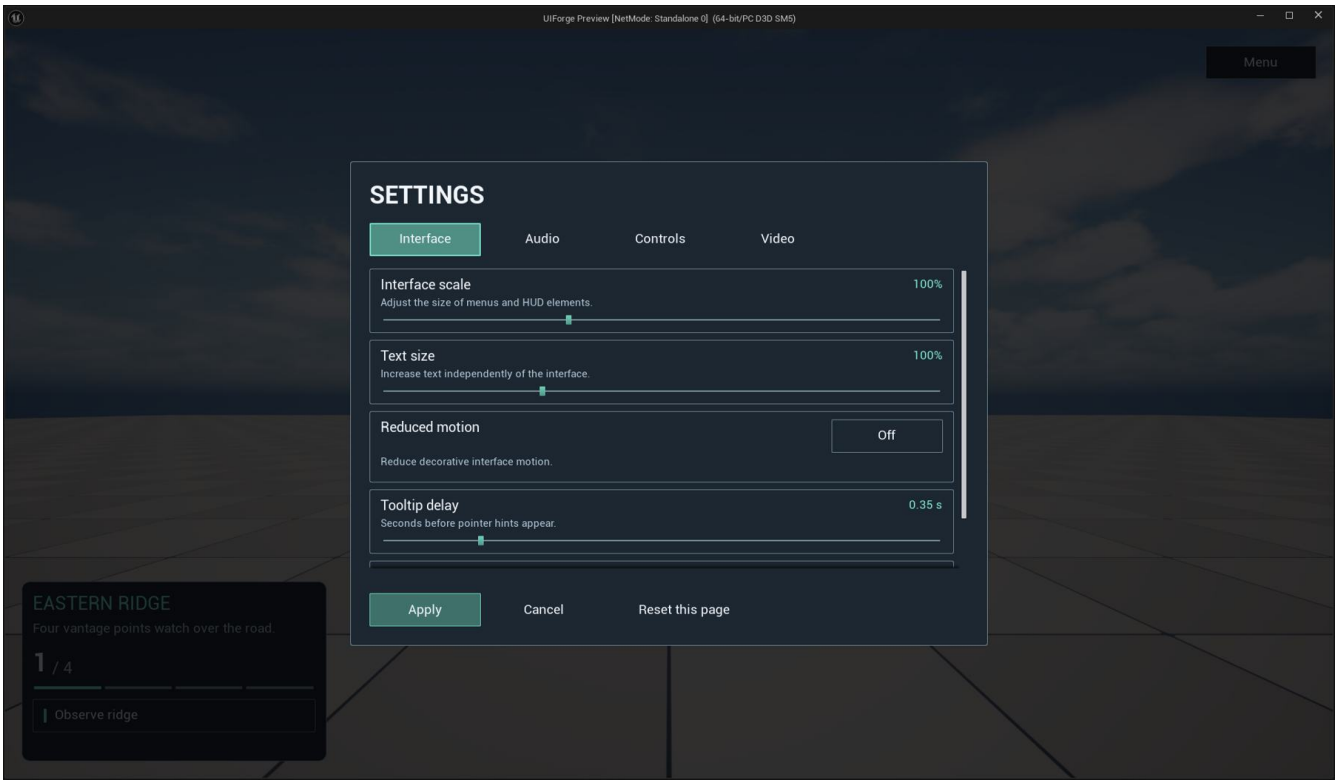


Tutorial pause Blueprint graph using Request Back for Resume and Push Screen for Settings



Actual return confirmation opened by the learning pause menu

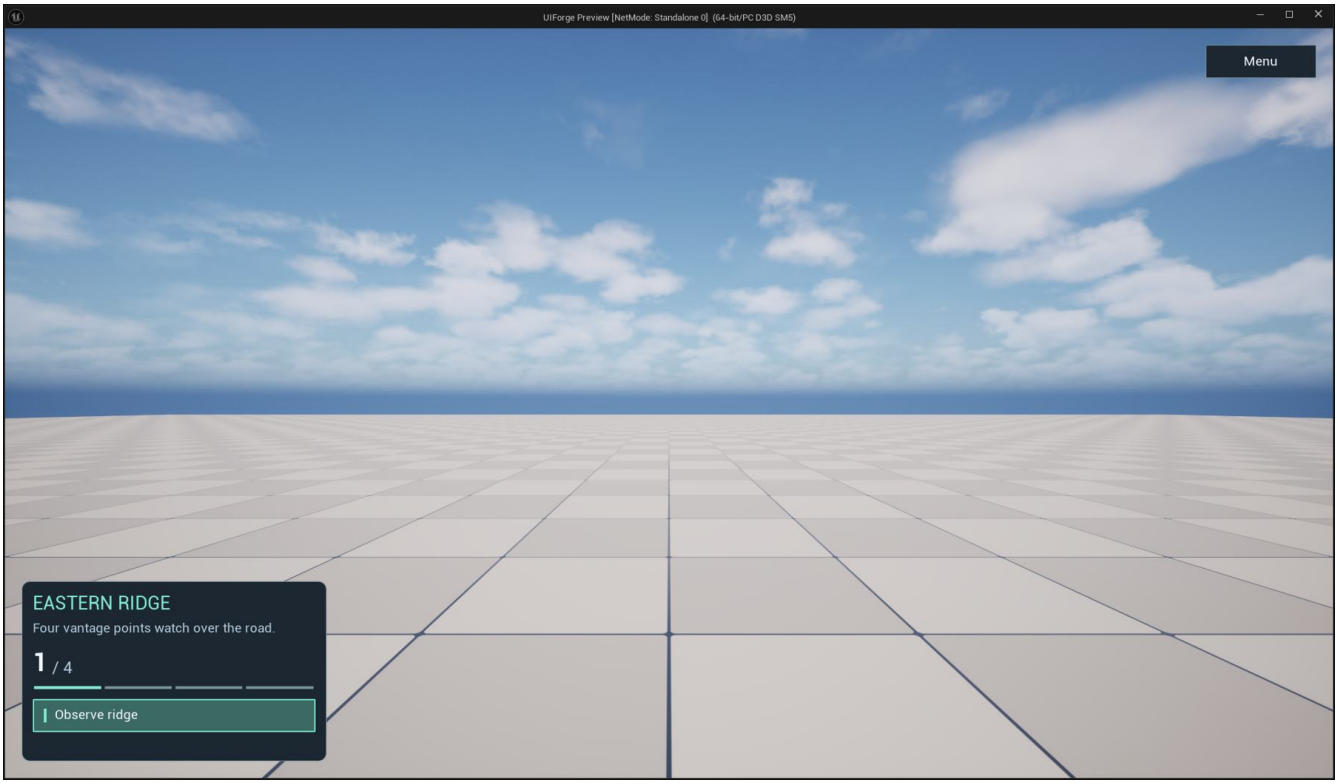
Open settings through **Push Screen** → **Menu**. Build it using the visual settings walkthrough, rather than a C++ class that constructs your game's whole menu. Dialogs use **IFDialogData** → **Push Dialog** and return a result; execute game actions only after handling that result.



The shared settings menu running over the learning HUD, using the learning flow's theme

9. Verify your integration

Play the learning map and follow **main → HUD/custom entry → pause → settings → Cancel → return confirmation → main**. Check that the entry responds, the screen stack restores focus, Cancel restores previews, and closing/reopening does not duplicate the root or its subscriptions.



The Blueprint learning flow running with a custom entry after one observation action

The visible count comes from the Blueprint source/model; the entry button requests the action and the model notification refreshes its presentation.

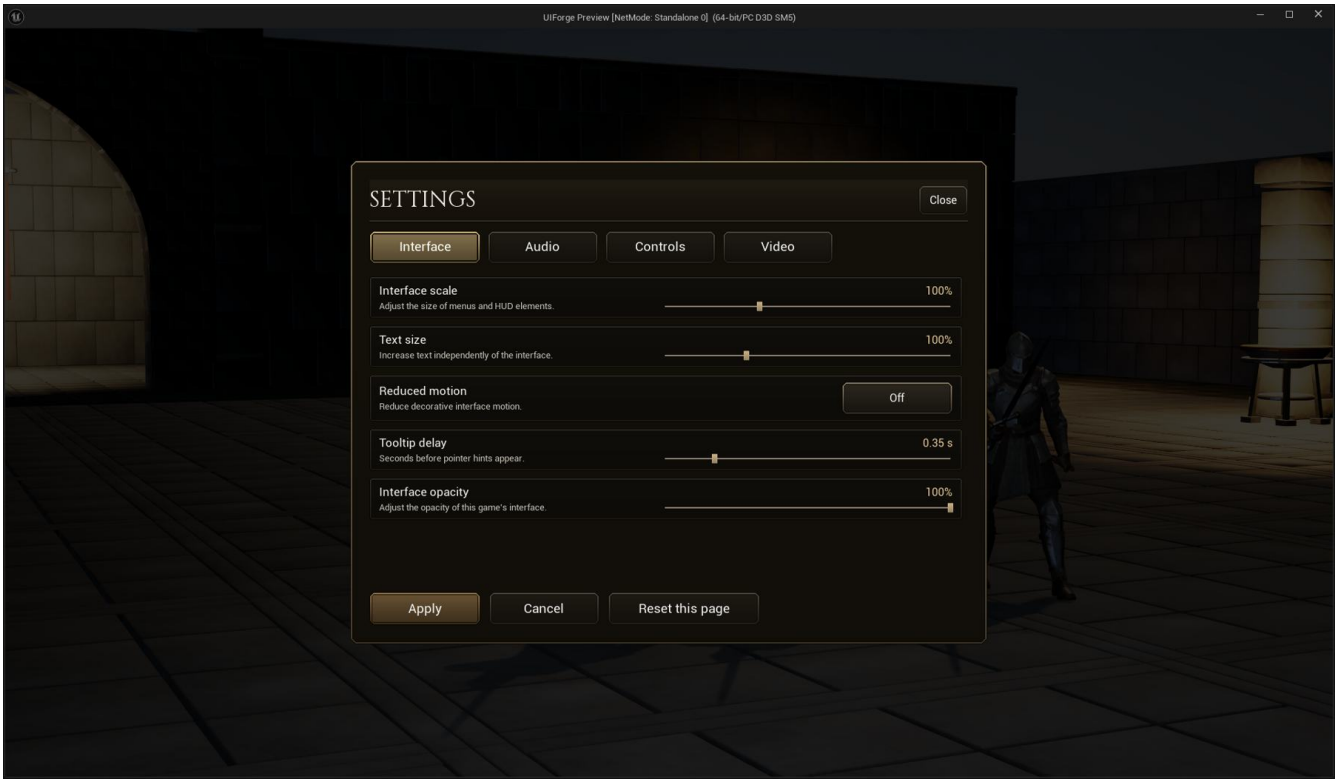
Then test your game's real data adapter, remapped prompts, supported resolutions and input devices. The fuller integration guide explains authority, travel, packaging references and troubleshooting. An attractive screenshot is not evidence that these runtime behaviors work in your project.

START HERE

Build a tabbed settings menu

This is a Blueprint authoring walkthrough, not just a tour of the example. You build the screen and row layout in UMG, define values with Data Assets, and let one **IFSettingsMenu** coordinate preview, Apply, Cancel and defaults. No game-specific C++ menu builder is required.

Click an image to enlarge it. Use **Actual size · 100%** and scroll for small editor fields or graph pins. Keyboard users can focus an image, press Enter, tab to the size control and press Escape to return.



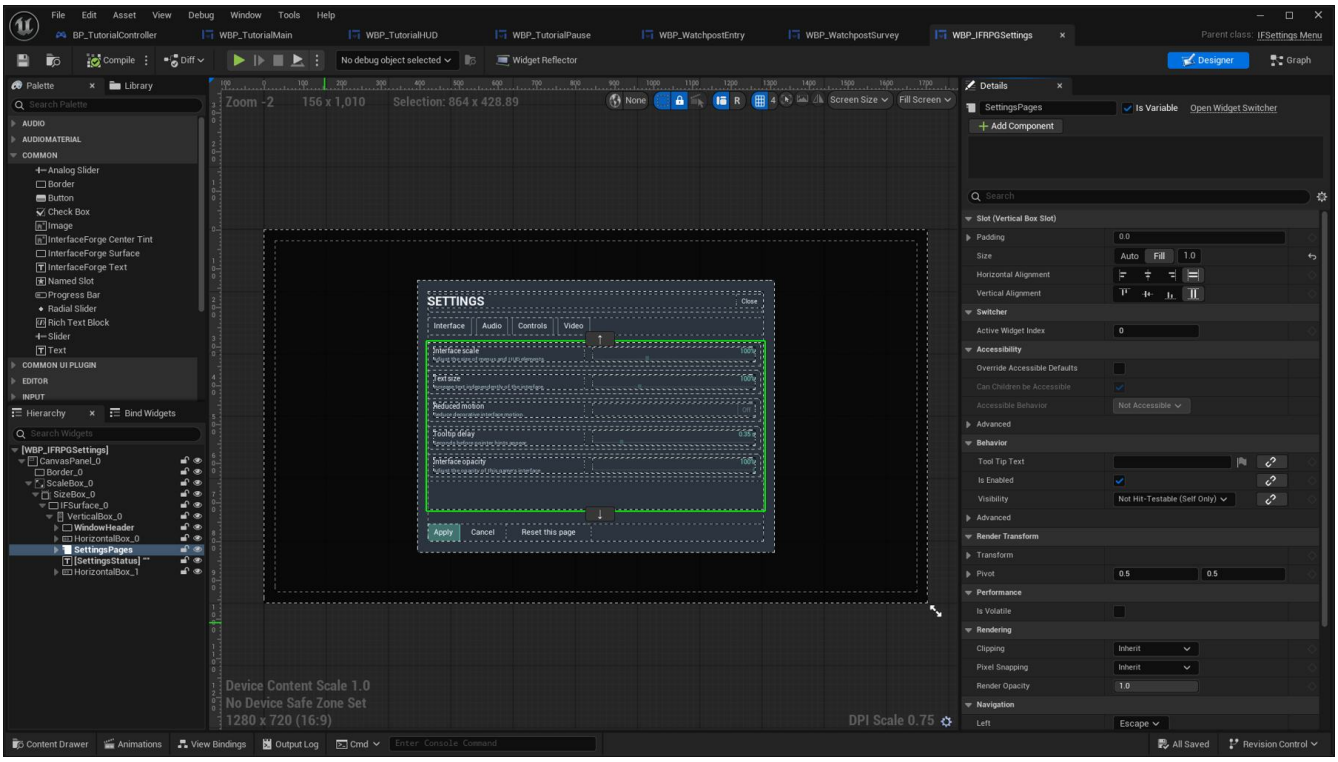
Current source RPG settings with four tabs, definition-driven controls and shared footer

This is the expected playable result. The pictures below show where its pages, tabs, definitions and controls are authored. All captures use the current UE 5.8.3 source project, not the older packaged candidate.

1. Create the screen and its pages

Create a Widget Blueprint with parent **IFSettingsMenu**. For a saved reference, open `/Game/ForgeExamples/UI/RPG/WBP_IFRPGSettings` in Designer. Build your background, heading, tabs, page body and footer from ordinary widgets.

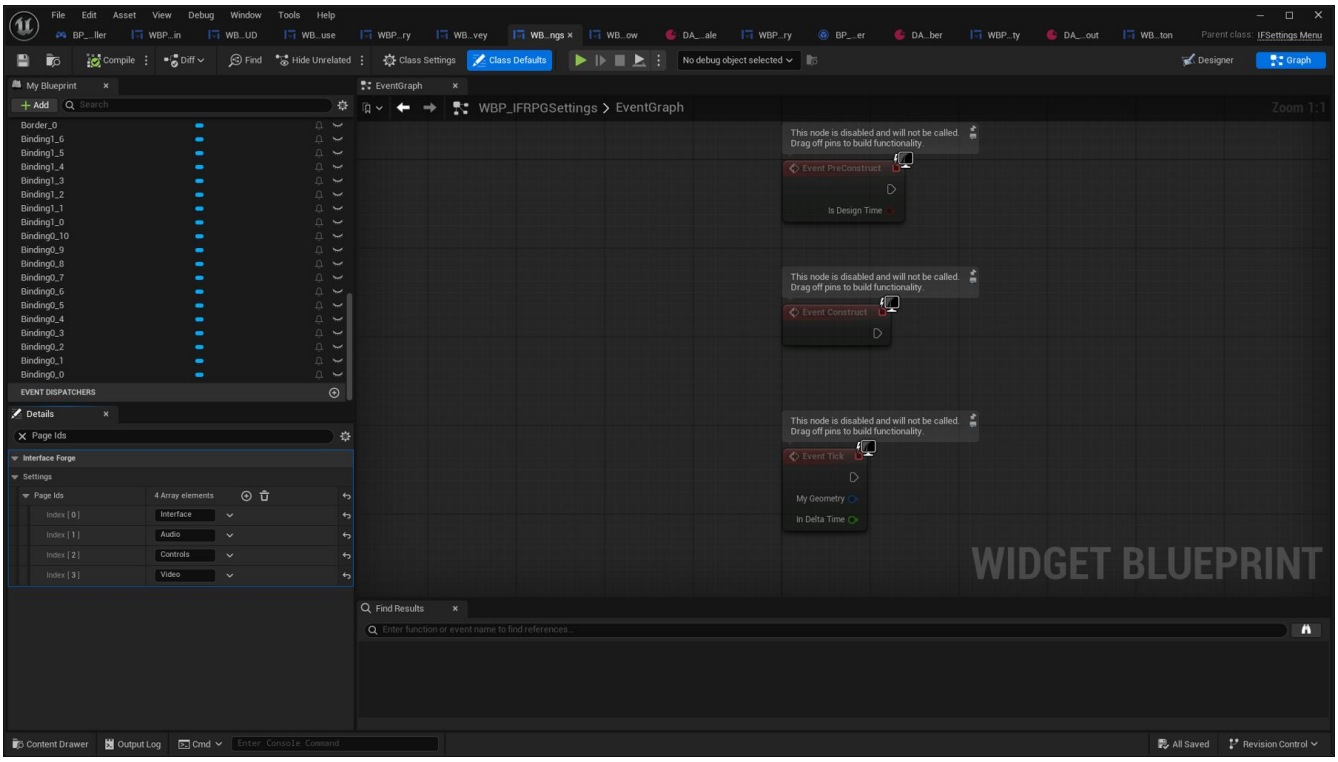
Add a **Widget Switcher** named exactly `SettingsPages`. Add one page/Scroll Box per category. The example order is **Interface, Audio, Controls, Video**.



SettingsPages selected in the actual settings Designer hierarchy

Select SettingsPages in the left Hierarchy. Its children are the category pages. The preview is a design-time view; gameplay preferences are not loaded or changed by opening this asset.

Choose **Class Defaults** → **Page Ids** and enter `Interface`, `Audio`, `Controls`, `Video` in the same order. If you reorder pages, reorder the IDs too.

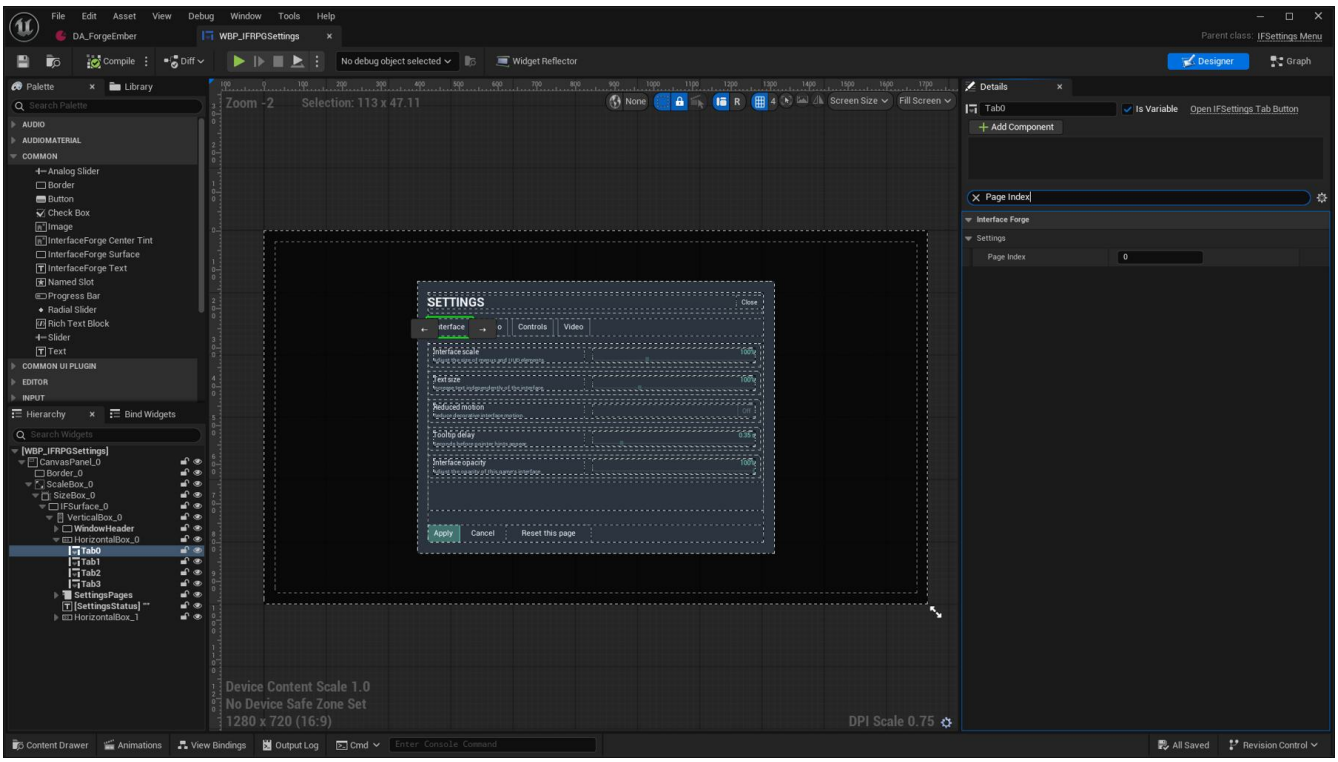


Settings menu class defaults showing the Page Ids authoring location

Expected: each page has one stable identity. A setting's Page Id connects it to defaults/category behavior; it is not a localized tab label.

2. Add navigation tabs

Place an **IFSettingsTabButton** for each page. Set its **Label** and **Page Index**. The example uses `Tab0` through `Tab3`, with indices 0-3. Their layout and skins remain normal Blueprint controls.



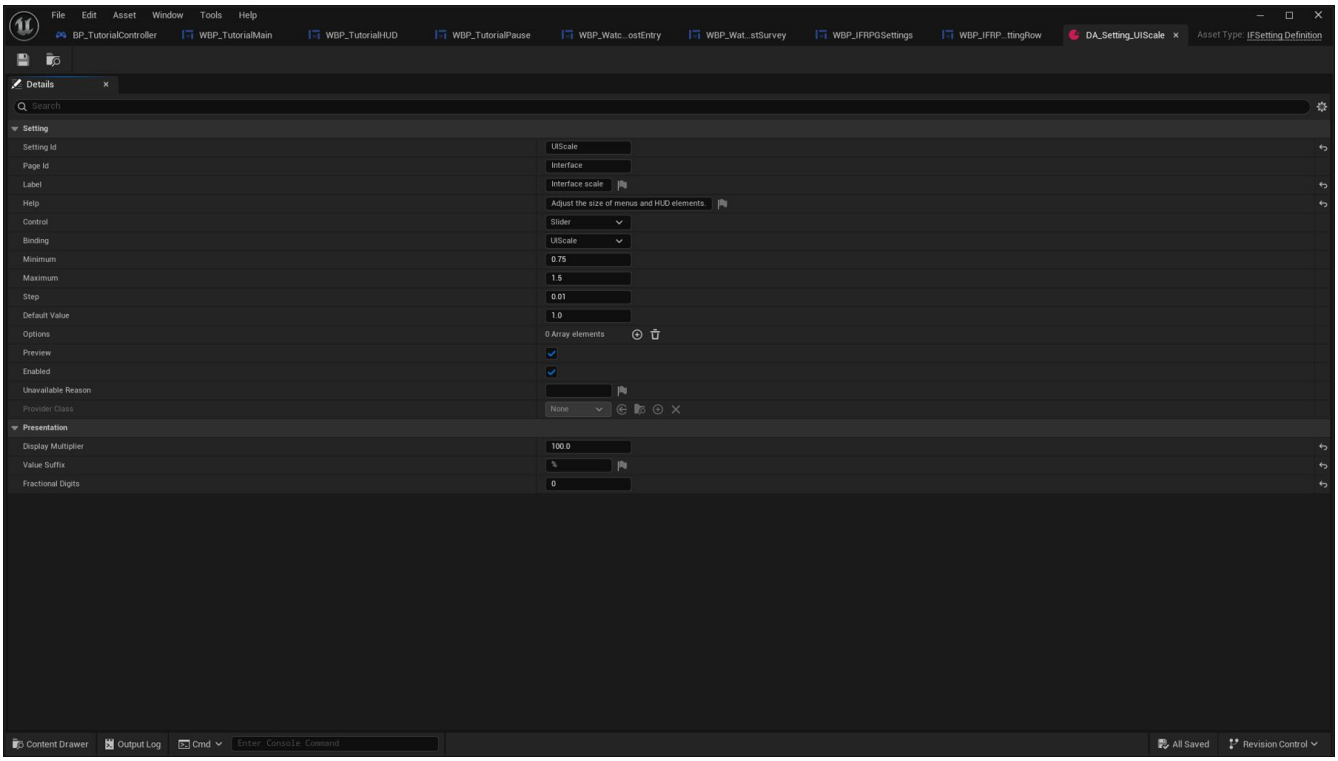
Tab0 selected for its settings Page Index in the actual Designer

The menu connects these buttons automatically. For a custom tab design, call **Select Settings Page** and use **On Settings Page Changed** for presentation/focus or a UMG animation.

Expected: changing tabs retains every draft value. It does not Apply or Cancel the session.

3. Define a real value

Create a Data Asset of class **IFSettingDefinition**. Open `/Game/ForgeExamples/UI/Settings/DA_Setting_UIScale` to compare fields.



Actual UI scale setting definition with its identity, page, binding, control and numeric bounds

Set **Setting Id**, **Page Id**, localized **Label/Help**, **Binding**, **Control**, **Minimum/Maximum**, **Step** and **Default Value**. UI scale uses its builtin binding. A slider's bounds must stay within that binding's supported range. **Display Multiplier**, **Value Suffix** and **Fractional Digits** control display formatting: a stored 1 can display as 100%.

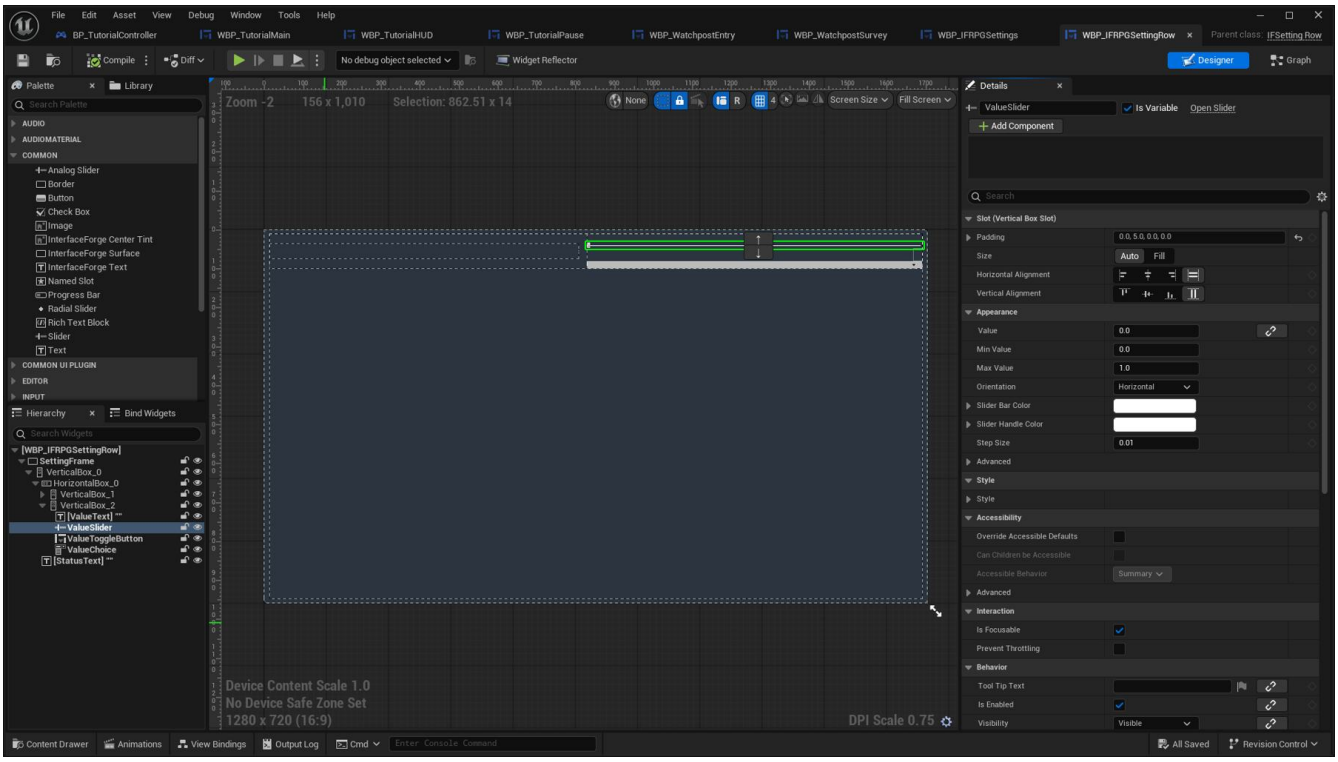
Toggle values are 0/1. Choice definitions supply localized options with stable numeric values. Do not use translated display labels as persistence keys.

Expected: one definition describes the value independently of its widget layout. Builtin UI/audio preferences use UIForge's settings service; supported video options use Unreal GameUserSettings.

4. Create your own row in Blueprint

Create a Widget Blueprint with parent **IFSettingRow**, or inspect `/Game/ForgeExamples/UI/RPG/WBP_IFRPGSettingRow`. Build the layout you want. Name only the controls it uses:

Designer name	Widget	Meaning
LabelText / HelpText	Text Block or IFText	Definition text
ValueText	Text Block or IFText	Formatted numeric value
ValueSlider	Slider	Slider value/range/step
ValueToggleButton	IFButton	Themed On/Off control
ValueToggle	Check Box	Alternative boolean control
ValueChoice	Combo Box String	Choice options
StatusText	Text Block or IFText	Validation/unavailable reason



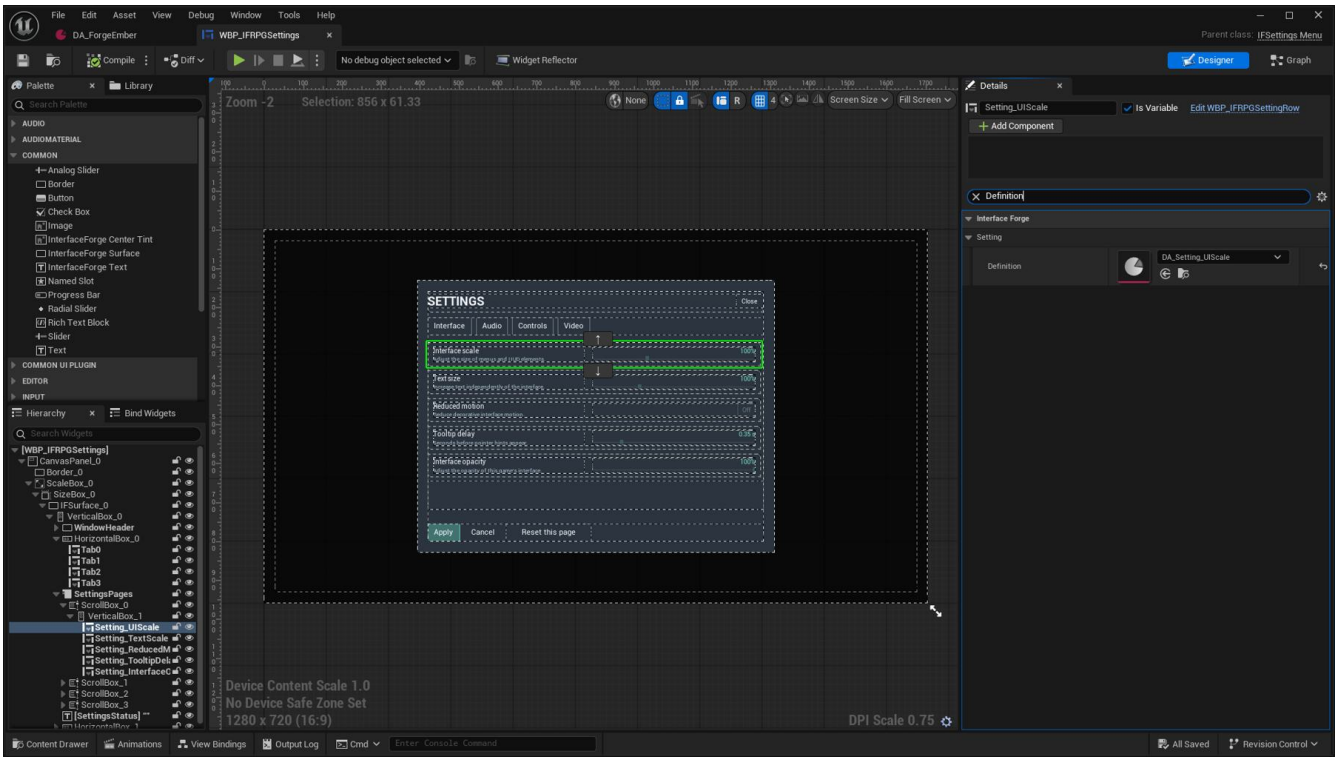
Saved row Designer with its label/help column and replaceable value controls

The parent initializes the useful controls and collapses those not selected by the definition. In Designer, a row with a definition can preview its label/default without opening a settings transaction. A bare row layout without a definition is a template.

For a custom control, implement **On Setting Refreshed** to present the value and call **Submit Setting Value** when the player edits it. Use **On Setting Submitted** for success/failure feedback. Keep engine setters out of each individual row.

5. Place the row and assign its definition

Place your row on its page and set **Definition**. In the saved settings screen, select `Setting_UIScale` and choose `DA_Setting_UIScale`.



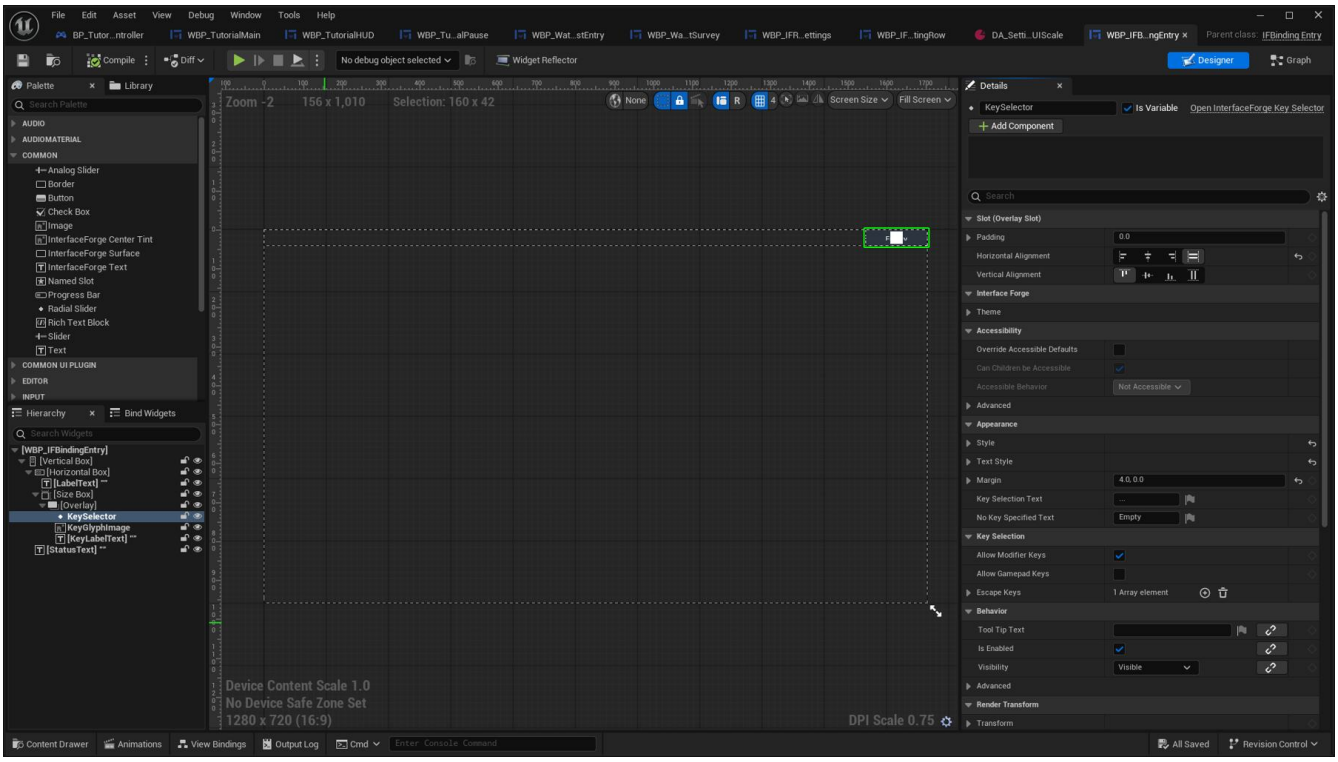
Setting_UIScale selected in the settings screen for its Definition assignment

The menu discovers Designer rows across its pages and initializes their sessions. For rows created at runtime, register definitions in **Additional Definitions** before opening, then **Create Widget → Set Definition → Add Child → Initialize Setting**, using the menu's **Editing Session**.

Expected: you can replace the row class, help layout, colors or control art while retaining the setting's rules and persistence contract.

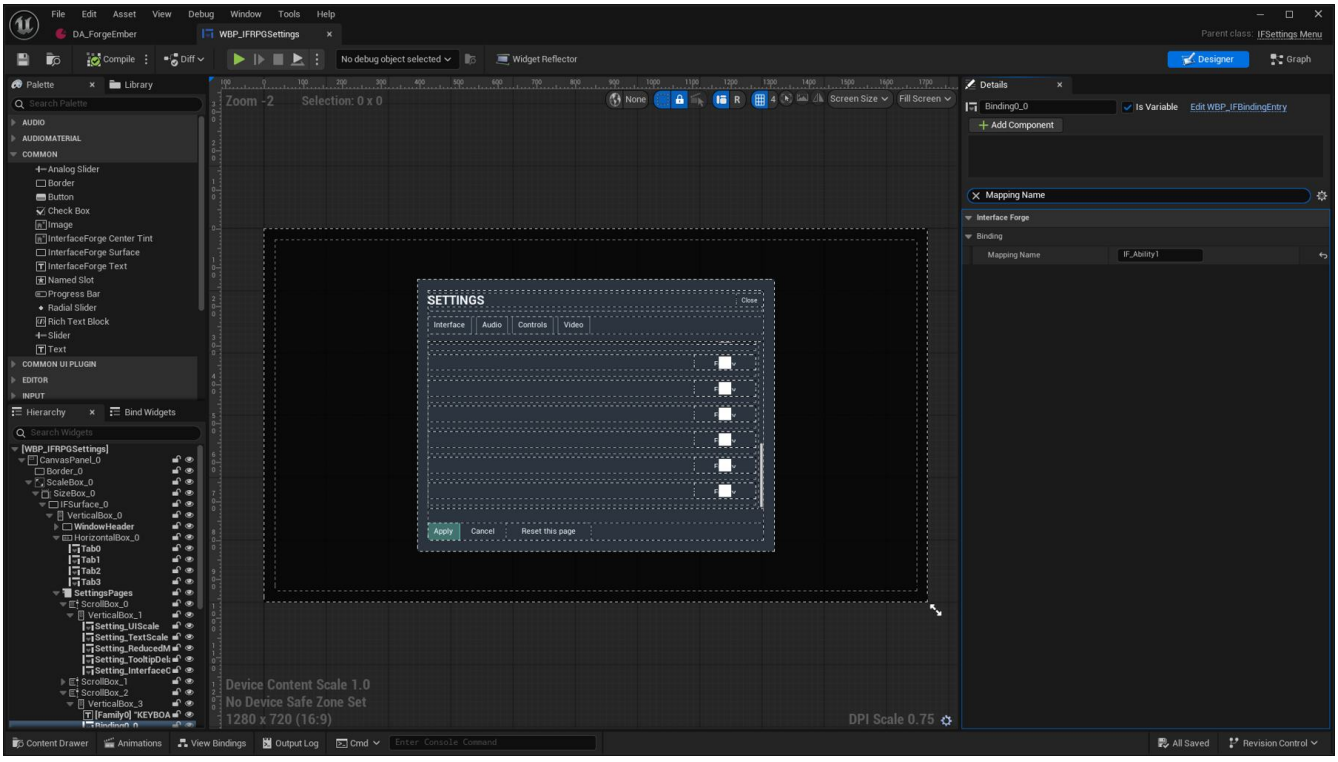
6. Add input rebinding

Enable **Include Input Bindings** on the menu. Place Blueprint children of **IFBindingEntry** on Controls. `/Game/ForgeExamples/UI/WBP_IFBindingEntry` shows the editable key selector and optional glyph/text overlay.

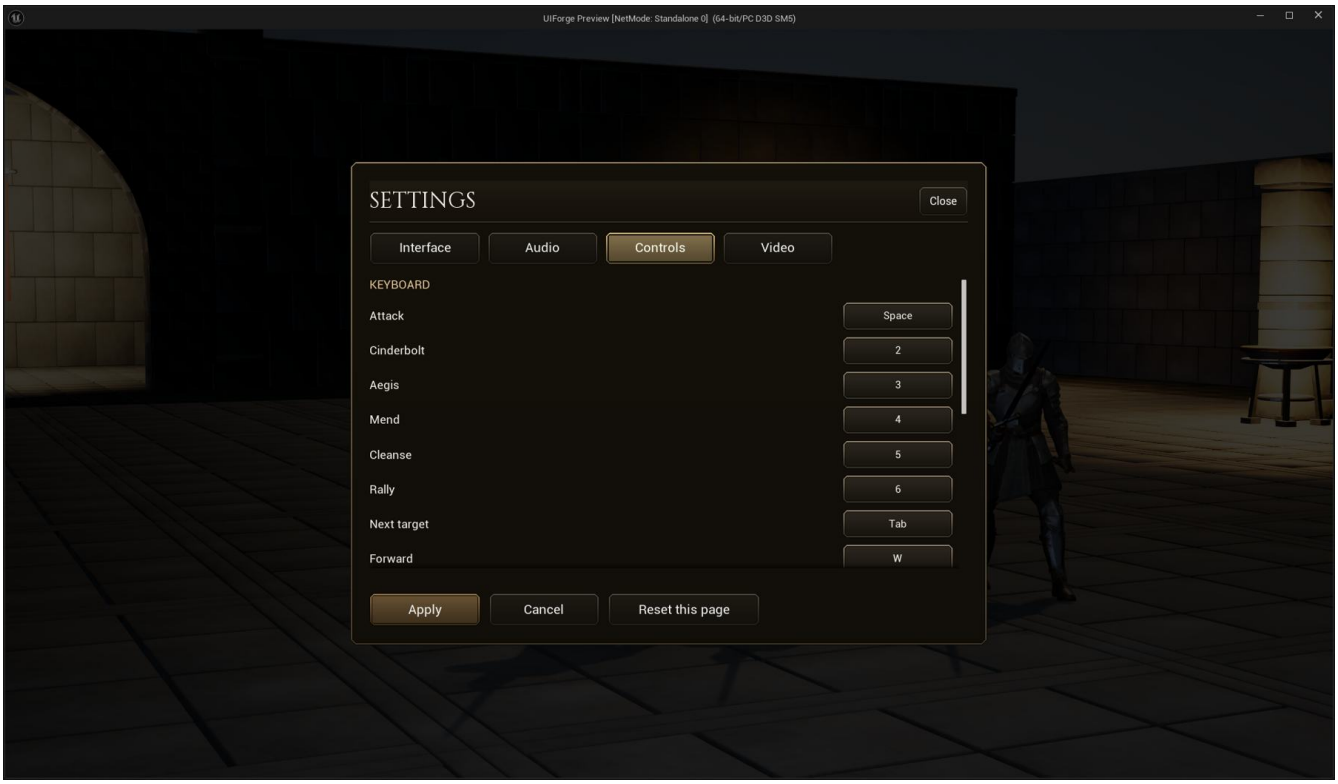


Editable binding row with KeySelector selected in its real UMG hierarchy

On each row instance, set **Mapping Name**, **Family** and **Label**. The mapping must already be registered through Enhanced Input User Settings and exist in a player-mappable context.



A settings binding instance selected for its actual Mapping Name



Current playable Controls page using real key capture fields

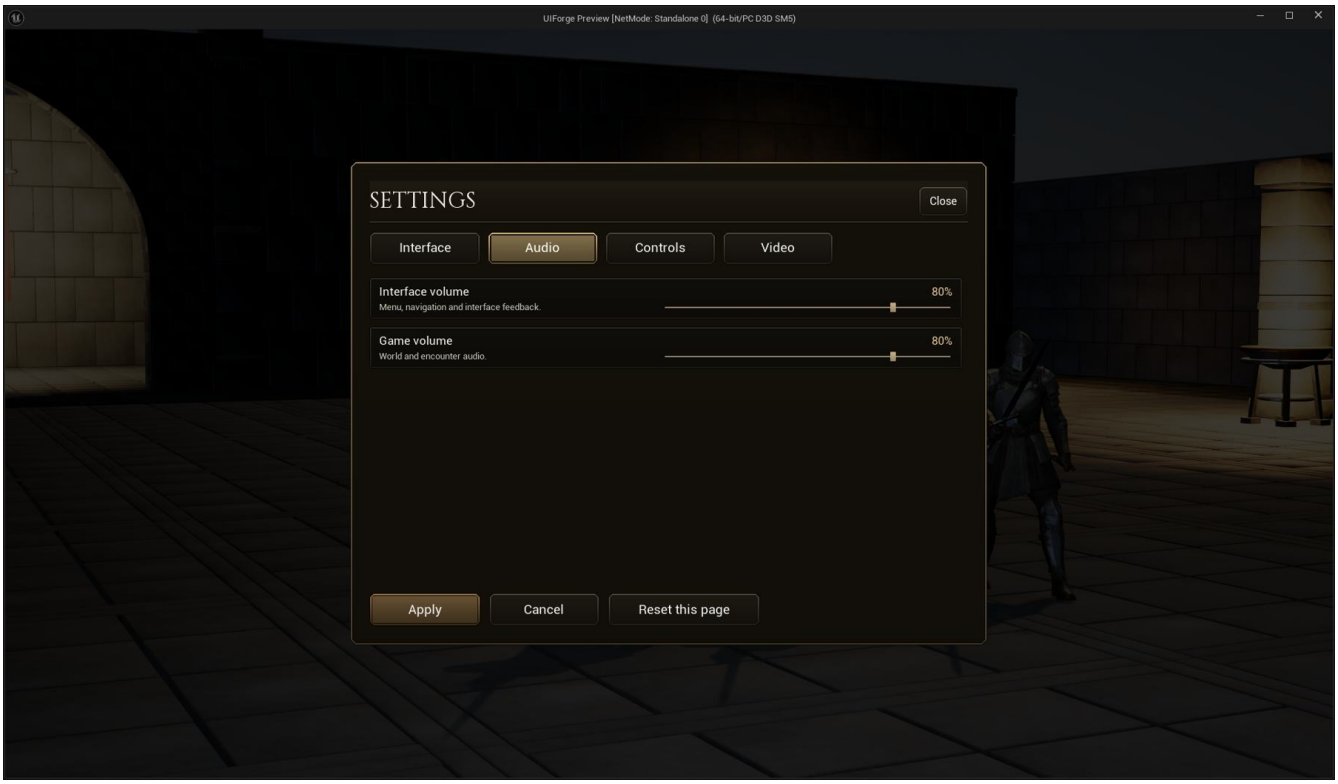
Do not add a second IFRemappingScreen inside this menu: that would introduce another transaction owner and Apply button. Bind **IFActionPrompt → Mapping Name** to the same registered name so prompts follow applied keys.

Expected: selecting a key starts capture; a valid change stays staged until Apply. Cancel discards it. Software controller fixtures do not certify physical devices; controller/touch hardware checks remain deferred.

7. Wire one footer and focus contract

Use IFButton controls named `ApplyChoice`, `CancelChoice`, `DefaultsChoice` and a Text Block named `SettingsStatus`. The menu supplies behavior. Label DefaultsChoice **Reset this page**: it resets the current page, not every category. Use an explicit separate control calling **Restore All Defaults** for a global reset.

Set **Initial Focus Name = Tab0**, **Fallback Focus Name = CancelChoice**. Open the screen through the root's **Push Screen**, on **Menu**, from either main or pause. Back cancels the session; an open dropdown receives dismissal first.



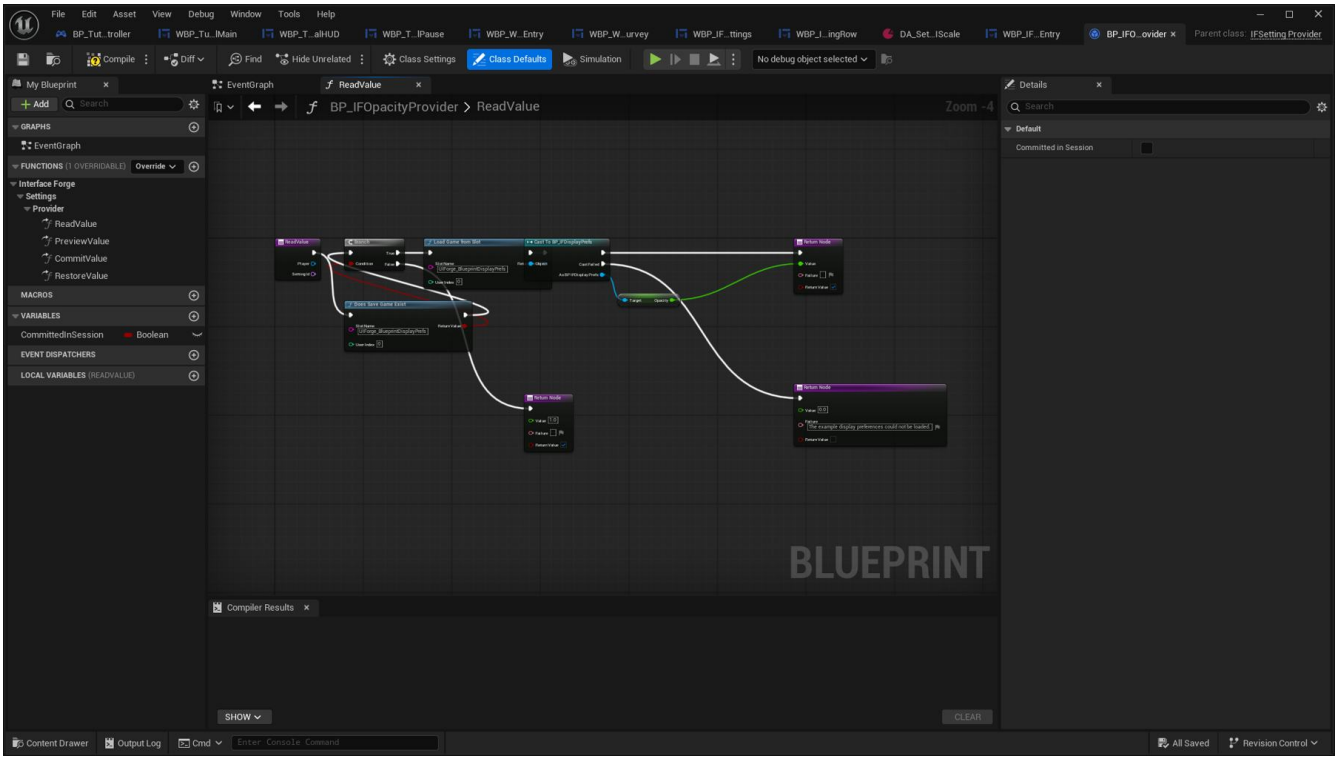
Audio page within the same settings screen and shared Apply/Cancel footer

Expected: an audio or scale preview changes the example immediately; Cancel restores the prior presentation. Apply commits supported values. The detailed settings contract explains rollback limits: SaveGame, Enhanced Input and GameUserSettings are not a single atomic disk transaction.

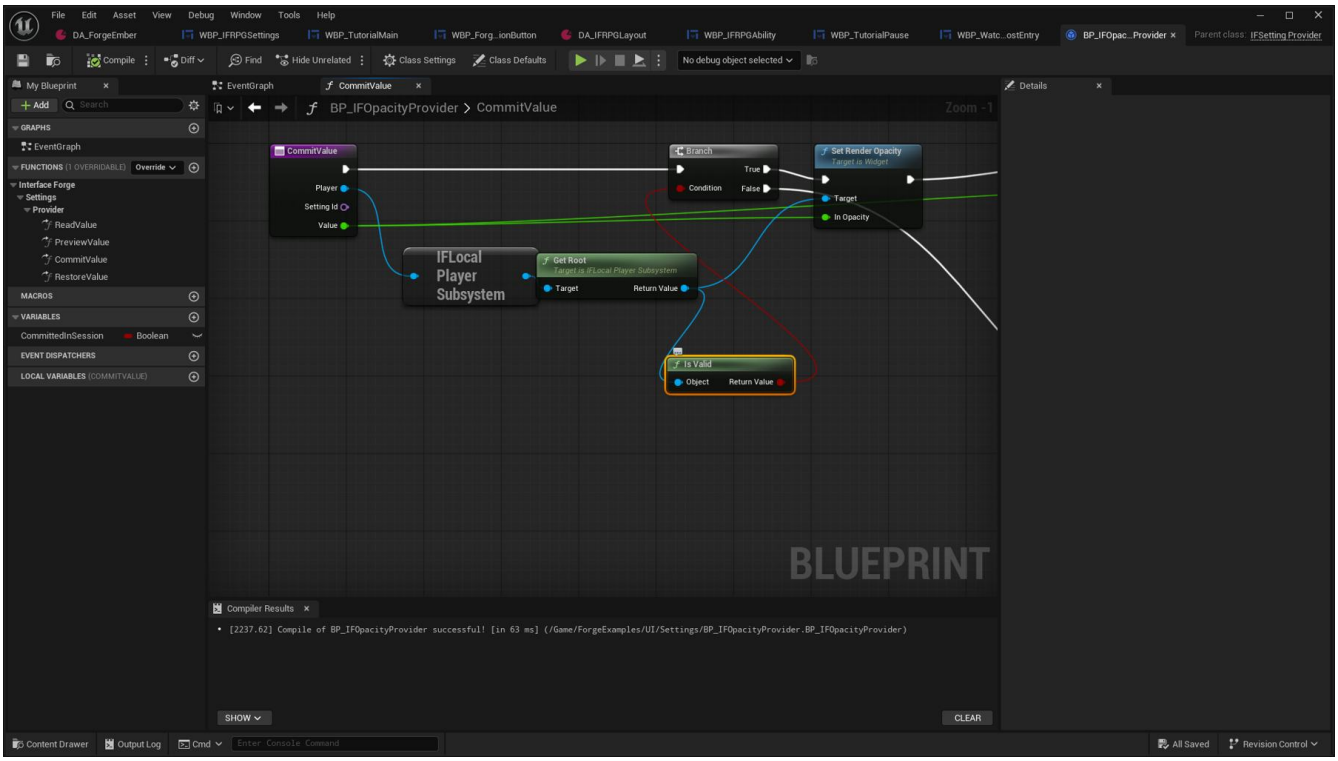
8. Extend with a Blueprint provider

For a game-owned setting, create a Blueprint child of **IFSettingProvider**. Implement **Read Value**, **Validate Value**, **Preview Value**, **Commit Value**, **Restore Value**. Assign the provider class on a Custom setting definition.

The included `/Game/ForgeExamples/UI/Settings/BP_IFOpacityProvider` is a complete example. Read loads the game's own preference; Preview updates the local root; Commit persists; Restore undoes preview or a partial commit.



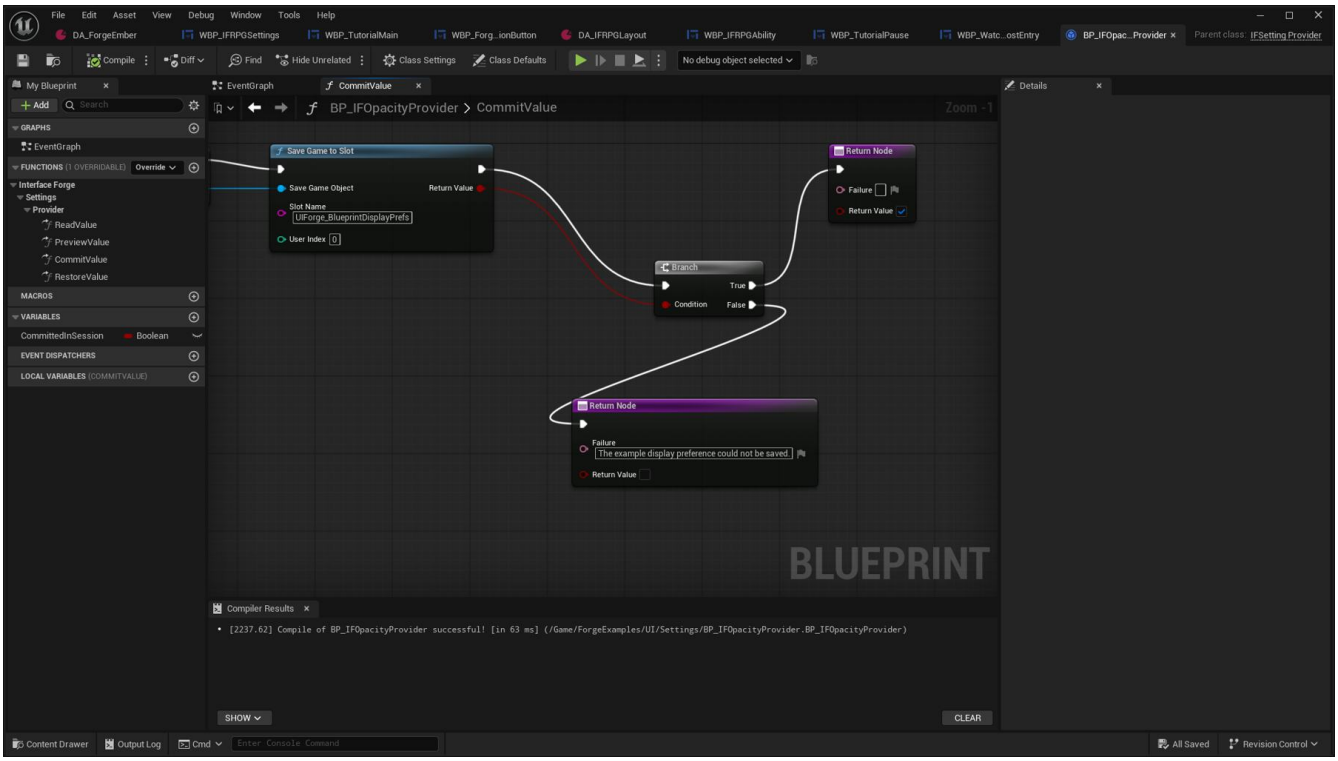
Actual Blueprint provider ReadValue graph loading game-owned display preferences



Actual Blueprint provider CommitValue detail validating the local root before Set Render Opacity

Use the supplied Player to get IFLocal Player Subsystem → Get Root. Branch on Is Valid before Set Render Opacity; return a failure when the root is unavailable.

After the preview succeeds, the example uses **Create Save Game Object** (`BP_IFDisplayPrefs`), sets **Opacity**, then calls **Save Game to Slot**. Its return value must drive a success/failure Branch.



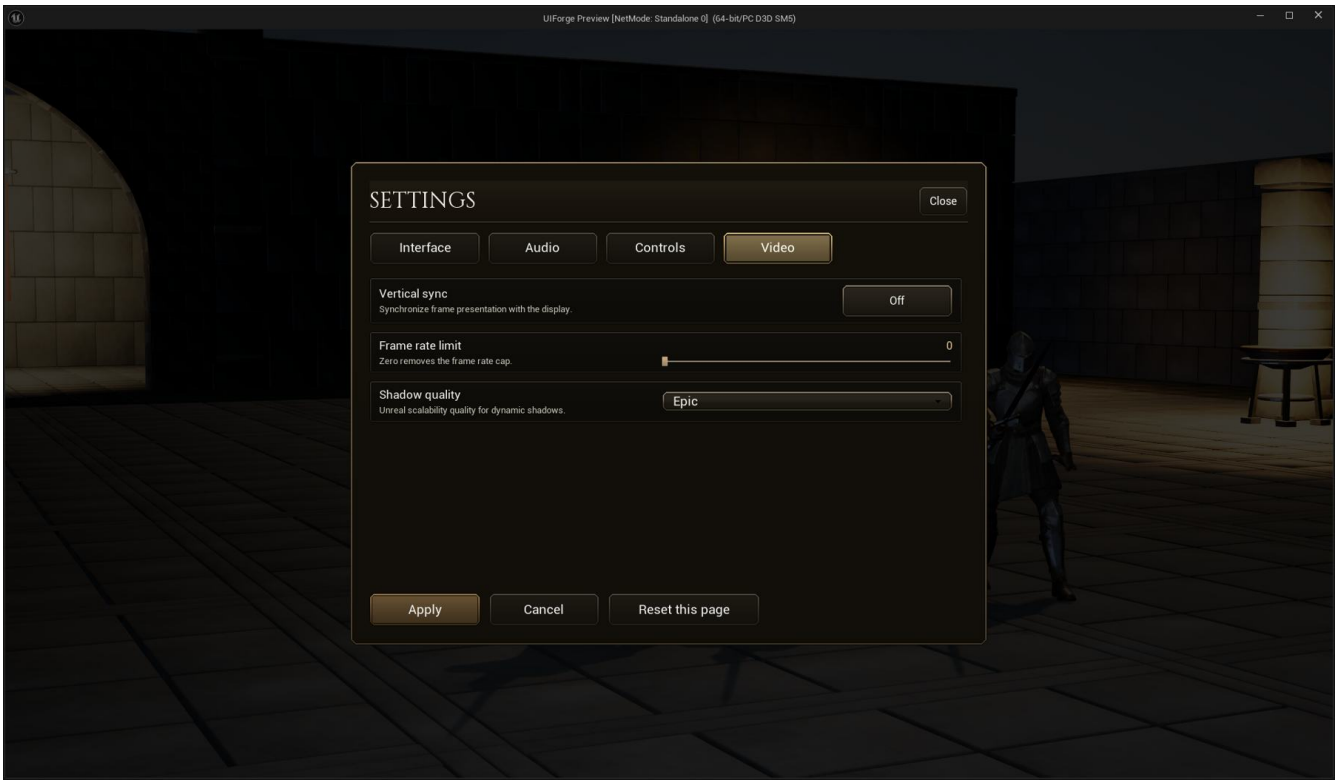
Actual Blueprint provider Save Game to Slot with its boolean result connected to success and failure returns

The slot is this example's own display preference, not UIForge's builtin preference slot. Return false and a useful Failure text when saving fails. Preview alone must never be reported as a successful persistent commit.

Keep the owning controller and stable Setting Id supplied by the session. At game startup, apply your saved custom setting after creating the root through your provider's public contract. Builtin UI/audio/video bindings already have their own engine-backed paths; don't duplicate them in a custom provider.

9. Check the complete flow

Preview a scale/audio change, change tabs, stage a key, Cancel, reopen and confirm the old values. Repeat with Apply, restart the game process and verify persistence and remapping-aware prompts. Test Reset this page separately from Restore All Defaults. Check validation, long labels, keyboard focus and dropdown Back behavior.



Supported video values staged through Unreal GameUserSettings in the current example

These builtin video rows cover the supported quality/VSync/frame-limit bindings. They are not a display-resolution confirm/revert implementation. Add that workflow using Unreal's existing GameUserSettings confirmation APIs if your project needs it. See settings and audio, localization and full game integration for the surrounding setup.

START HERE

Customize the same interface in Blueprint

Follow these steps to change appearance, layout and data without editing plugin C++. Click any real editor/runtime image to enlarge it; **Actual size · 100%** preserves readable editor text with scrolling. Gameplay stays in your game's adapter.

1. Duplicate a theme into your project

Enable **Show Plugin Content** in the Content Browser. Open `/InterfaceForge/UIForge/Themes/DA_ForgeEmber`, then duplicate it into your game's Content folder before customizing. `DA_ForgeMoonsteel` is the cool alternative.



Actual core theme Data Asset with editable presentation properties

Author Body Font, semantic colors, spacing, sounds, transitions, **Button Styles** and **Surface Styles**. Call **Set Theme** on **IFLocalPlayerSubsystem** before **Create Root**. Use **Theme Override** for a local widget variation.

Expected: multiple screens share one material/type/color family; changing a theme does not copy their widget trees or gameplay rules.

2. Style buttons as one family

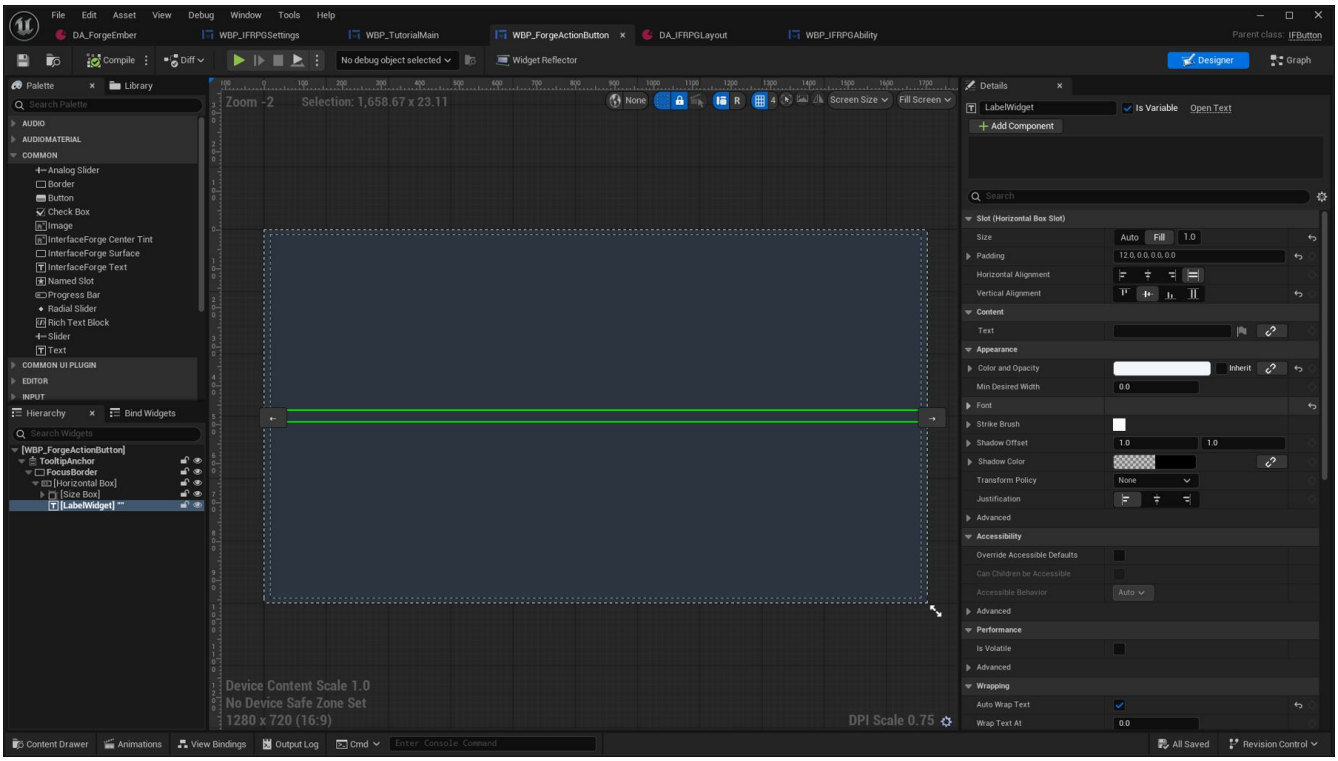
Under **Button Styles**, author `Standard`, `Primary`, `Quiet`, `Destructive`. Each **Button Skin** contains Normal, Emphasized, Pressed, Selected and Unavailable brushes, text colors and Feedback Seconds. Texture/material brushes use normal Slate nine-slice margins.



Primary button skin expanded to show its state brushes, text colors and feedback duration

On IFButton/IFCommandButton set **Button Role**, or use **Skin Name** for another key. For live edits use **Set Button Role**, or set Skin Name then **Refresh Appearance**. Missing keys retain the semantic fallback.

For a custom tree, derive a Widget Blueprint from IFButton; use IFCommandButton when you need data-source action routing. Optional `FocusBorder`, `LabelWidget`, `AvailabilityWidget` and `TooltipAnchor` names participate in built-in presentation. Alternatively drive your tree with **On Appearance Updated** and **On Visual State Changed**.



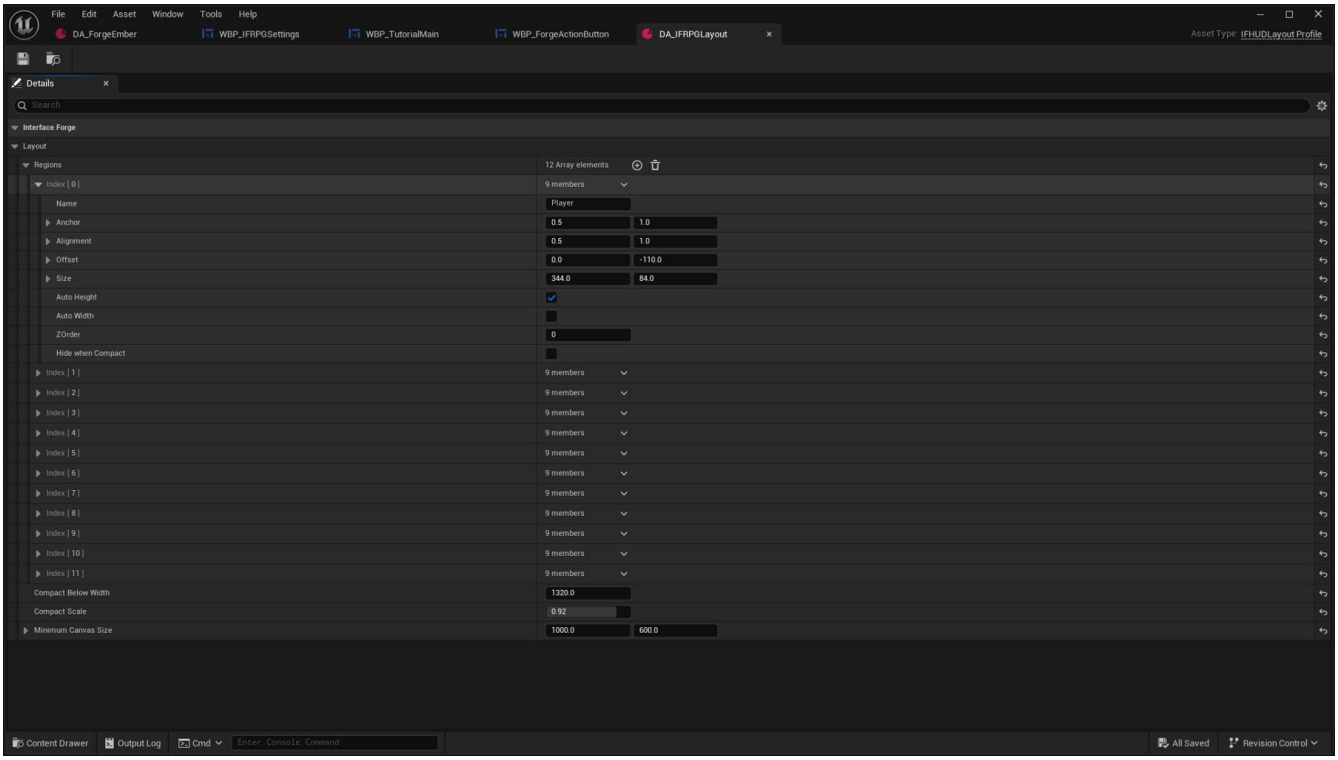
Saved Blueprint action-button Designer with its replaceable content tree

LabelWidget is selected inside TooltipAnchor → FocusBorder → Horizontal Box. This reusable template receives its label from IFButton; the empty template text is not a finished game button preview.

Expected: your art changes while click behavior, CommonUI focus and tooltip contracts remain usable. Keep decoration hit-test invisible. Native labels preview in Designer; Blueprint-owned trees retain their own slot alignment. The built-in finite tint/text feedback honors Reduced Motion. Your own UMG animation should check **Get Presented Preferences** → **Reduced Motion** too. See the exact styling contracts.

3. Replace a HUD region

Open `/Game/ForgeExamples/UI/RPG/DA_IFRPGLayout`. Expand **Layout** → **Regions** → **Index[0]**. Its named regions define anchors, alignment, offsets, dimensions, automatic sizing, ordering and compact-layout visibility.



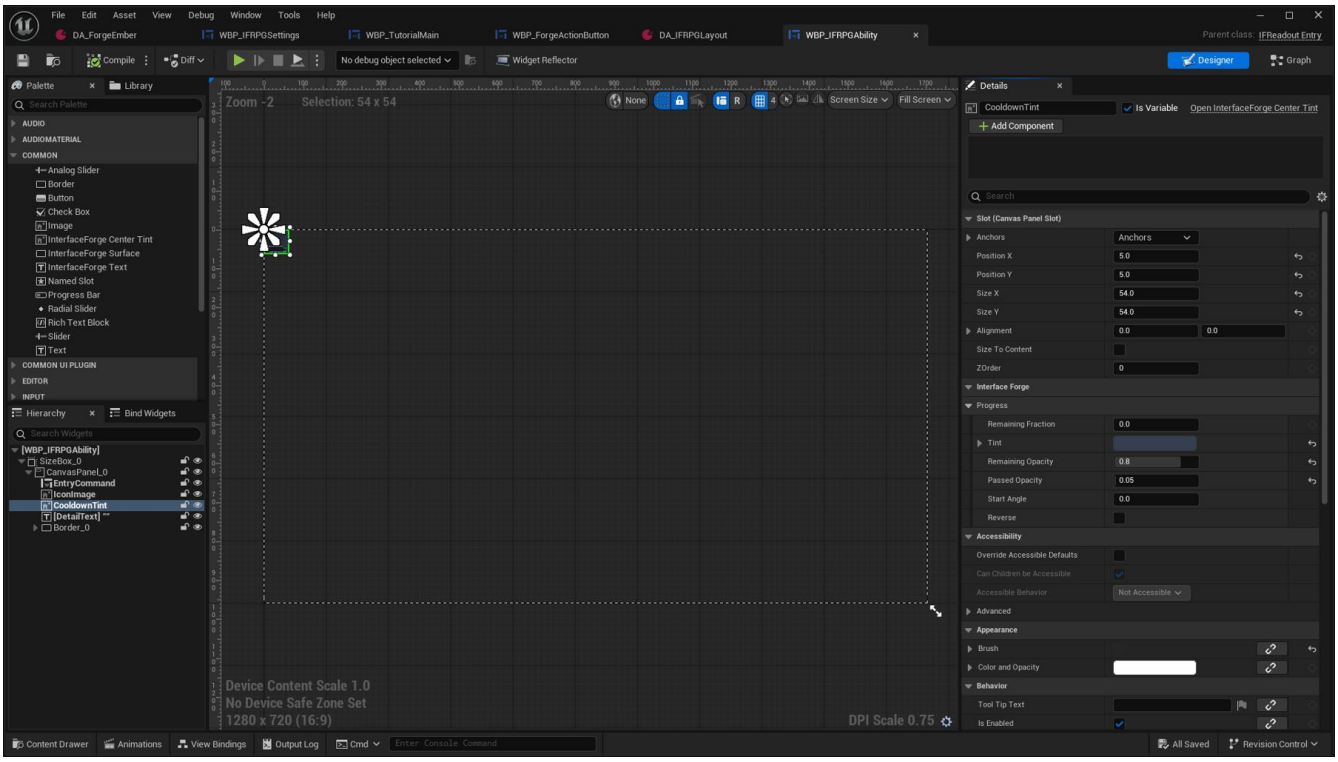
Actual HUD layout Data Asset with editable named regions

Create an IFHUDLayout Blueprint with your profile, then use **Set Region Widget** to assign a widget to a region name. Widget creation/data binding belongs to your Blueprint; the profile controls placement. Connect the replacement to the existing model or your own interface-provided data.

Expected: a different party frame, action control or context panel can occupy that region without a new data backend. Test long labels and enlarged text; uniformly increasing every size is not a complete responsive design. See responsive layouts and HUD authoring.

4. Customize an icon and its cooldown

Open `/Game/ForgeExamples/UI/RPG/WBP_IFRPGAbility` and select `CooldownTint`. Its ordinary Designer tree contains the command hit target, icon, tint, countdown and remapping-aware ActionPrompt.



Actual ability entry Designer with CooldownTint selected

Expand Interface Forge → Progress in Details. The entry's icon/data are assigned at runtime; this view teaches the tint properties and its layering over IconImage.

IFCenterTint provides the independently authored center-origin interior tint based on the inspected Sakura behavior. Tint and remaining/passed opacity are presentation properties; the example supplies its normalized recovery fraction through the entry. Keep the key prompt inside the icon so it cannot cut through the outer frame.

Supply your own icon texture, colors and countdown styling. For other presentations, IFProgressWidget exposes Linear, Radial, Square Perimeter, Segmented and Charges. These are separate choices, not substitutes for the center-origin tint.

Expected: one recovery value can drive another visual without ability rules moving into the UI backend. The center-tint guide documents actual parameters/direction, and progress recipes cover the other styles.

5. Switch the theme live

Call **Set Theme** on the same local-player subsystem with your replacement IFTheme. These two actual captures use the same window, data and actions.



Current ability browser using the warm theme



The same ability browser after Set Theme selects the cool theme

Expected: named button/surface roles and text update together. Replace a Widget Blueprint class to change layout; supply another IFDataSource implementation to change the information.

6. Create your own entry

Use the visual integration recipe to inspect the saved screen/entry graphs. Initialize stable data/context, update from notifications, route validated actions to the source, and release subscriptions on removal/recycling.

Item cards, quests and entity cards are presentation. InterfaceForge does not implement inventory storage, equipment, crafting or replication. MassForge integration stays in an optional adapter; the core works independently.

START HERE

Current source presentation and illustrated guides - 2026-10-27

The owner found the latest edge/alignment correction better and requested a MassForge-style visual manual. The current gallery now shows fresh, individually reviewed UE5.8.3 source captures. The new game-integration, tabbed-settings and customization routes show actual editor locations, selected properties, Blueprint graphs and expected results. Images enlarge by click or keyboard.

This is approval to continue documentation, not full product/release approval or competitor parity. The visual-integrity audit retains the earlier defects and correction gates. The executable and distribution archives have **not** been refreshed for this documentation/Designer-preview pass. Physical controller/touch checks remain deferred. This pass's record separates current checks from historical evidence.

Prior competitive pass - functional history, visual result rejected

The earlier2026-10-27 competitive pass implemented button states, window/header/surface skins, two independent core themes, compact menus/settings, art-based resources, a real ability browser and an event-sized RTS roster. Blueprint styling, settings authoring, ability-browser recipe, and scoped verification describe those contracts. The package/source archives predate the current correction; the current gallery is source-only and has now been refreshed.

Source builds,95 Blueprint compiles, independent core load, real game actions, native remapping, public customization and software input checks pass. The exact installed-source Win64Development package passes a27-observation native interaction/restart check, and its source/config/content matched the archives at that pass. Those archives and the executable have not been refreshed for the current correction. Physical controller/touch and owner visual approval remain pending. Older completion language below is historical and does not establish acceptance of this quality goal.

UIForge - current review candidate

Latest RPG correction: top-center target, rebuilt skill entries and actual rebinding workflow. The previous RPG revision was rejected by the owner; its historical passing tests did not constitute visual acceptance. This local revision fixes missing/stale shortcut defaults, wrapped keycaps, health-bar style, camera/genre separation, window sizing and map presentation. Review the current gallery and scoped verification. Physical controller/touch remain deferred; visual approval belongs to the owner.

Latest local follow-up: HumanMotion_v022 character presentation is integrated into the MMO/RTS examples with editable Animation Blueprint/Blend Space, contact timing, grip, blended actions and pause synchronization. Final77 Blueprint compiles,16 animation checks,12 RPG/17 army scenarios and a rebuilt Win64 packaged native run pass. Controller/touch limitations remain unchanged.

The army/HUD revision replaces the rejected small encounter presentation with recognizable RTS controls and a compact RPG interface. It is a **local Unreal Engine 5.8.3 / Win64 candidate for review**, with scoped evidence in verification. The gallery contains current engine windows; earlier eight-unit captures are historical.

What changed

- **Ironfront:** 24 friendly soldiers and 18 opponents; click/marquee/Shift/same-type selection, control groups, contextual orders, queued destinations, block/line/column formations, hold/stop, portraits, recruitment/cancel, minimap orders and camera panning/zoom. Navigation and the small local army simulation belong to ForgeExamples.

- **Emberfall:** continuing attack and chase, animation-timed contacts, explicit cast priority, movement interruption, target lifetime, party inspection and an actual objective journal with tracking. The inspected center-origin cooldown tint is retained through the public material/widget controls.
- **Presentation:** compact original palettes and SVG icons, stable resources, health/selection rings, responsive cards, a flowing command/production dock, scrolling portraits and action captions that expand with accessibility text size. Main/pause menus and the hub use saved Designer trees.
- **Public Blueprint features:** collection models, virtualized list/tile entry binding, recycled command context, batched selection feedback, primary/secondary map actions, live fraction signals, content-sized HUD regions and local style tokens. The examples use these same contracts.
- **Input:** CommonUI/CommonInput remain the foundation. Mixed HUDs use temporary mouse capture and opt-in applied-key routing. Device intent has drift/jitter thresholds; held inputs release across ownership changes. Stack activation focus is left to CommonUI so submenus restore the previous control.

Where to start

Open **ForgeGateway**, choose **Emberfall** or **Ironfront**, then use New Game, the HUD, pause, settings, save/load and return. The native RTS controls are documented in army and RPG HUD authoring. In Emberfall, WASD moves, Tab cycles living targets, 1 starts the continuing attack and 2 casts Cinderbolt; prompts follow the applied bindings.

To build your own interface, follow integration into an existing game, Blueprint settings composition and HUD/entry authoring. Settings have ordinary Designer-authored tabs, reusable rows, data definitions and Blueprint providers. The independent Blueprint tutorial is **/Game/UIForgeTutorial/Walkthrough**. The compact scheme maps all current files, classes, widgets and graphs.

Verification boundary

Current evidence includes native editor army and MMO interaction, API encounter/lifecycle checks, software input switching, 76 saved Blueprint compiles, 2,000-model TileView recycling and independent core builds/load with optional plugins absent. The packaged example is checked separately; use the verification page for the final cook and native test state. A layout screenshot is not an interaction pass.

Core depends on Unreal CommonUI/CommonInput, Enhanced Input, UMG and Slate. It has no MassForge, inventory, example-game or editor-authoring dependency. The internal **InterfaceForge** module/mount remains stable; the product/project name is **UIForge**. This installation reports 5.8.3 CL58210709 and compatible descriptor baseline 5.8.0.

Physical controllers and real disconnection, touchscreens, consoles, other platforms/engines, split-screen and multiplayer remain unverified. The examples are local interfaces with small game adapters, not a production MMO or complete RTS economy. Example saves retain unit states, resources and encounter progress; pending orders, control groups and recruitment queues reset on load. UI work stays on Unreal's required game thread; event-driven models and virtualization avoid per-widget world scans. No isolated GPU/FPS guarantee is claimed.

Game Master's design remains exclusive. Only inspected Sakura behavior is independently reproduced. No inventory mechanics, third-party game UI assets, publication or licensing change is part of this revision.

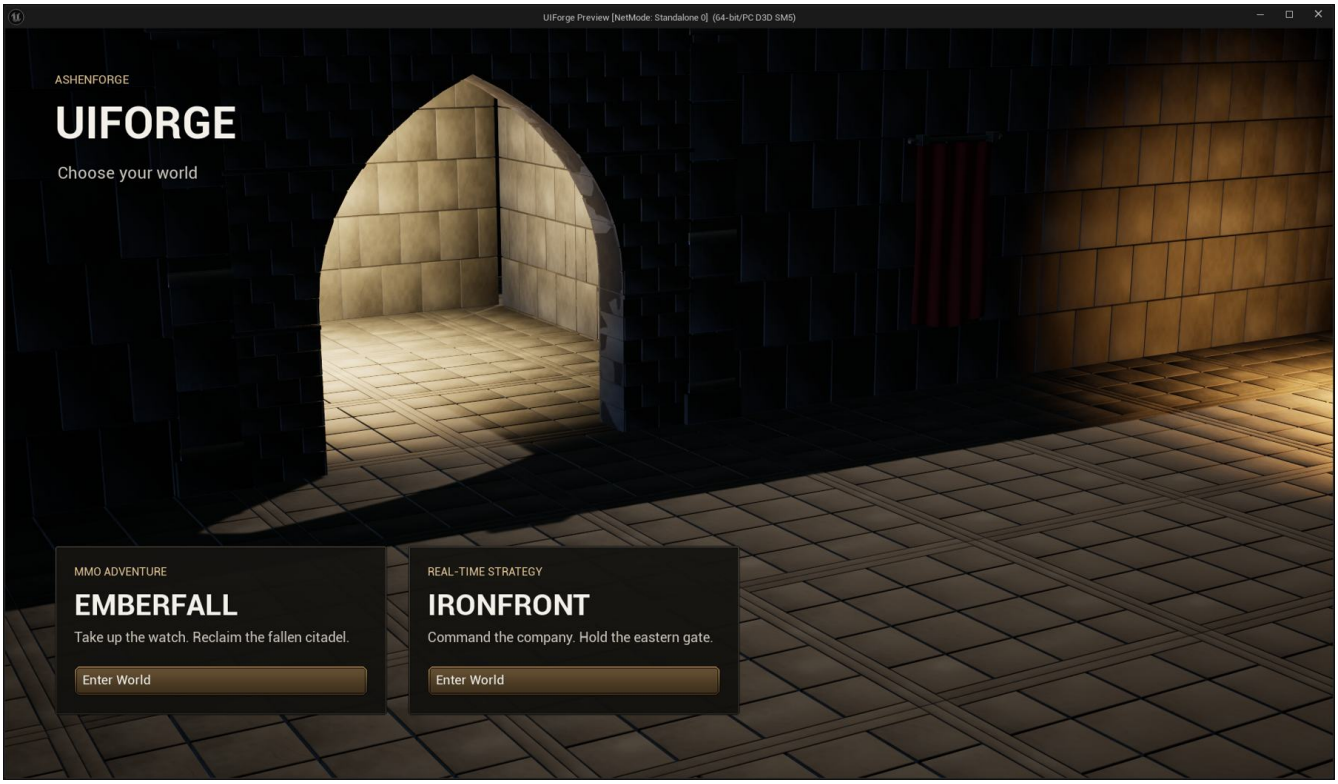
START HERE

Play the current examples

These are actual UE 5.8.3 **source-project** interfaces after the edge/alignment correction. The capture cursor is temporarily hidden and then restored. These are not mockups or images from the older packaged candidate. The examples use public widgets, entries, themes, layouts and CommonUI stacks; their small gameplay backends stay in ForgeExamples.

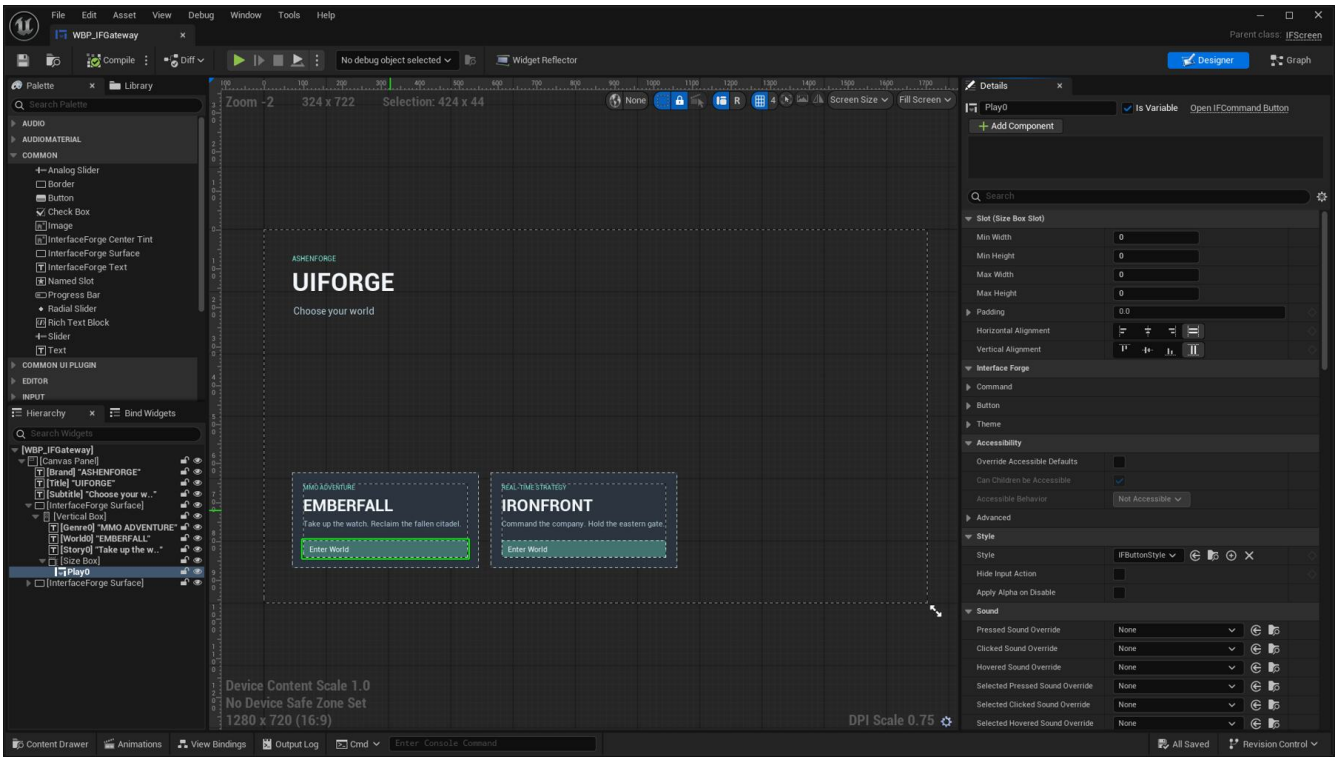
Open `/Game/ForgeExamples/Maps/ForgeGateway` and press Play to choose a genre, or open Emberfall/Ironfront directly. The older package has not been refreshed for these changes. Physical controller/touch remain unverified; this gallery does not certify performance or release readiness.

Start at the example hub



Actual example hub with separate Emberfall MMO and Ironfront RTS entry controls

Choose **Enter World** on the genre you want to play. The launcher itself is `/Game/ForgeExamples/UI/WBP_IFGateway`, a saved IFScreen Blueprint. Open it in Designer to change the card text, composition or presentation; `Play0` and `Play1` are public IFCommandButton controls.



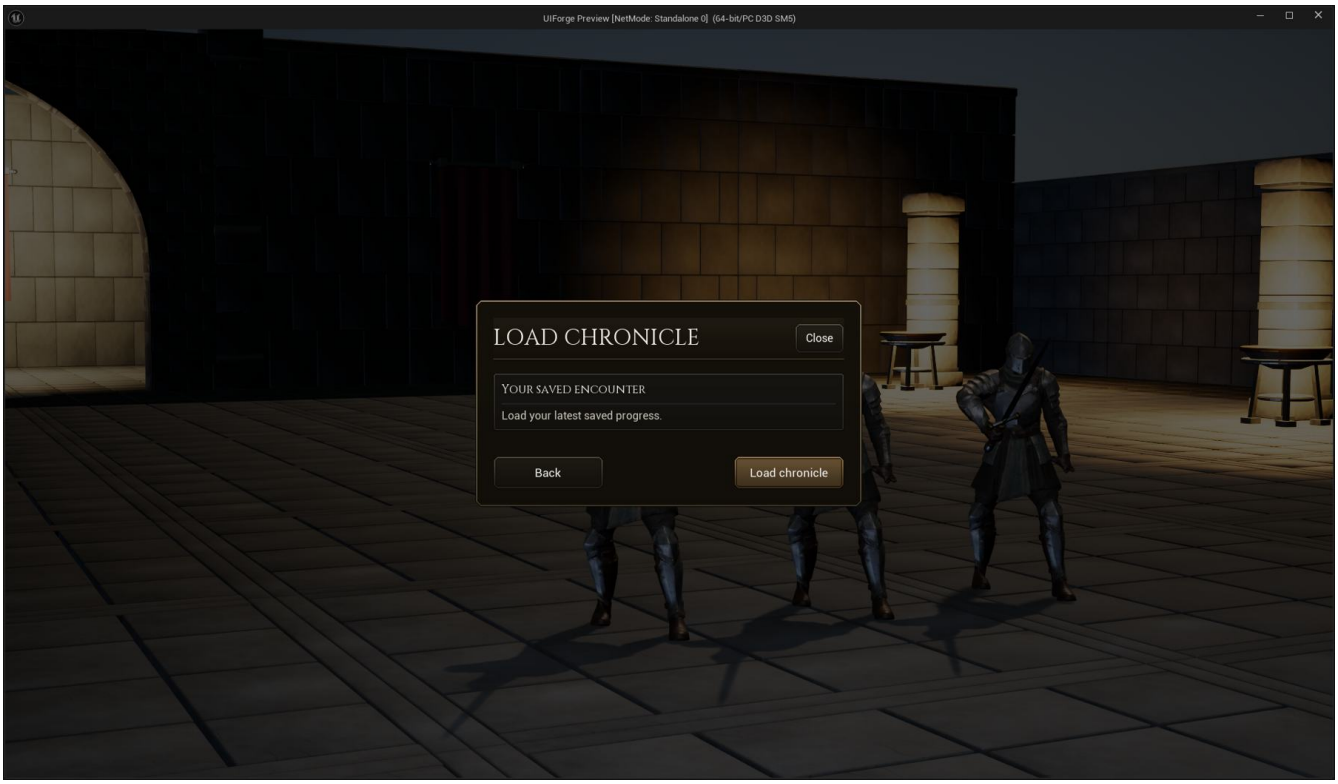
Actual WBP_IFGateway Designer with the MMO Enter World command selected in its ordinary widget hierarchy

Emberfall: main menu and dialogs

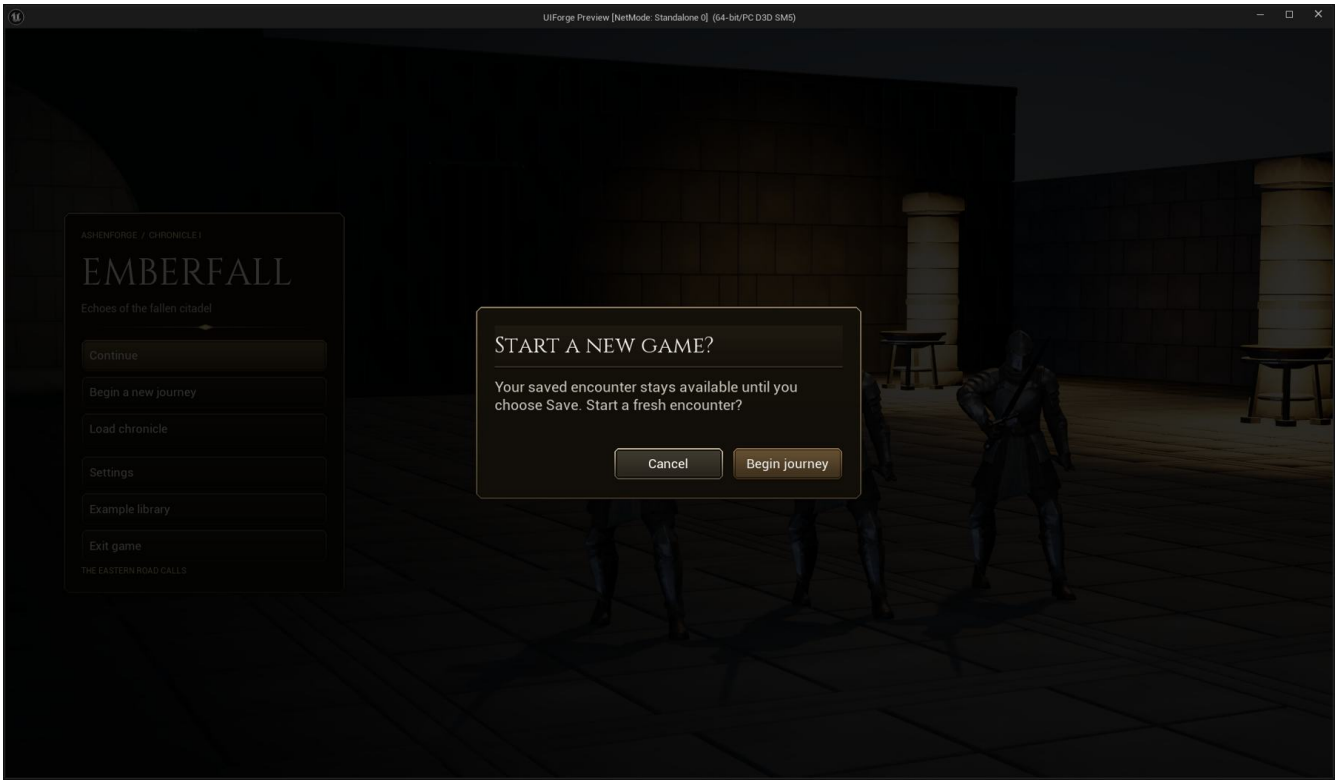


Current MMO main menu with Continue, new journey, load, settings, library and exit

Choose **Begin a new journey** for a fresh encounter or **Continue** when a save exists. New Game preserves the saved encounter until you explicitly Save. Load, settings and confirmation use the shared screen/modal contracts.



Load chronicle window with save information and actions



New journey confirmation using the shared dialog

Emberfall: third-person HUD and windows



Current third-person RPG HUD with party, target, minimap, objective and action controls

Move with the configured movement mappings, select a target and activate an action. Attack requests repeating approach/contact behavior; the other skills validate range, resource and recovery in the example backend. Prompts follow applied mappings, so your saved keys can differ from the pictures.



Combat Arts window showing the encounter's six real actions

The browser activates the same actions as the bar. Center-origin cooldown tint is independent of the rules. See ability-browser authoring and visual customization.

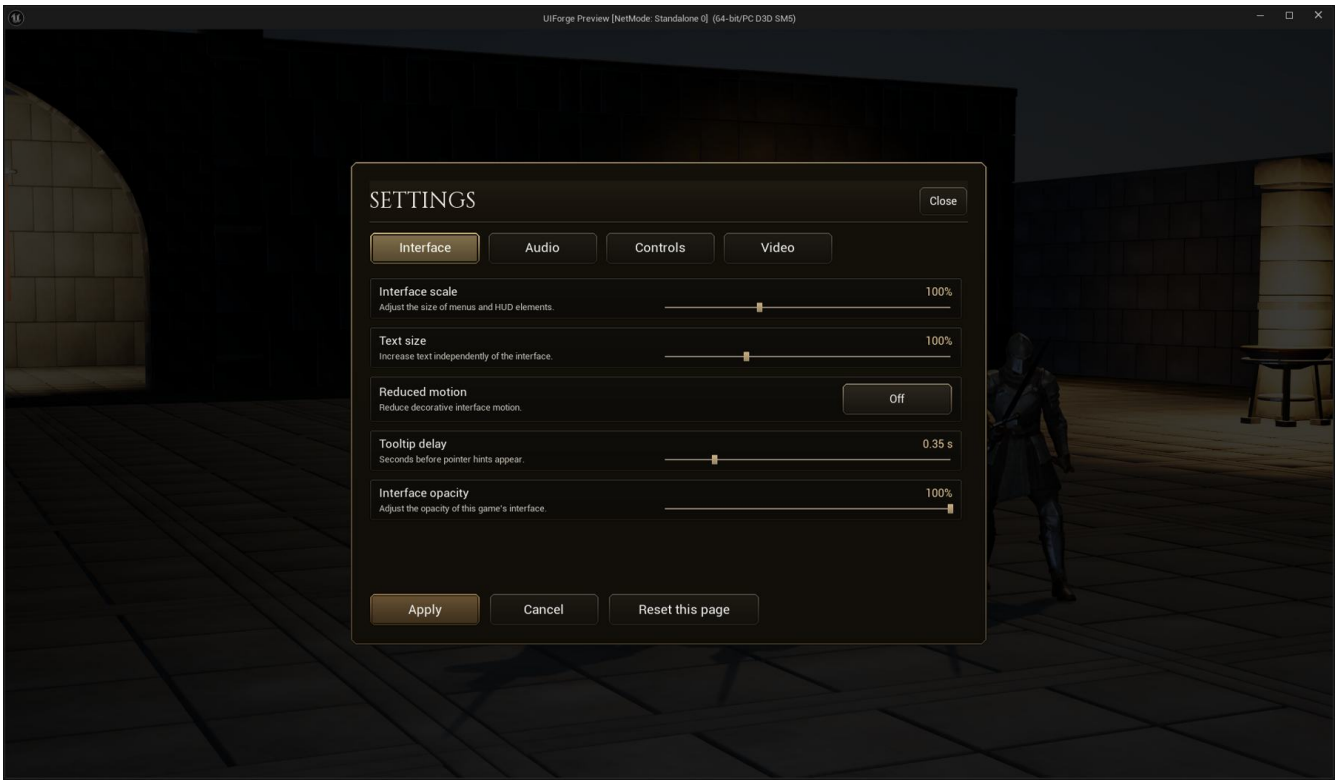


Chronicle window with the live objective and tracking action

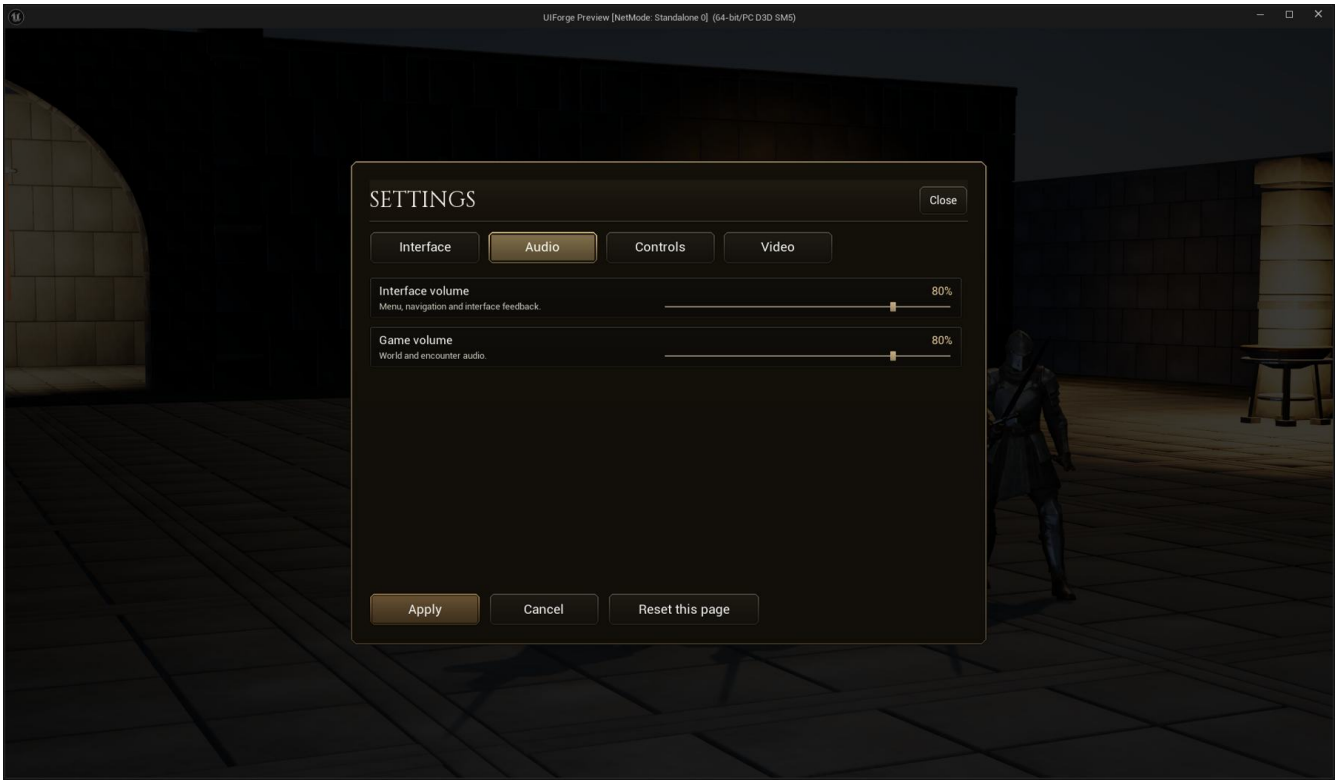


Pause menu after switching to the cool theme, with resume, save, load, settings and return

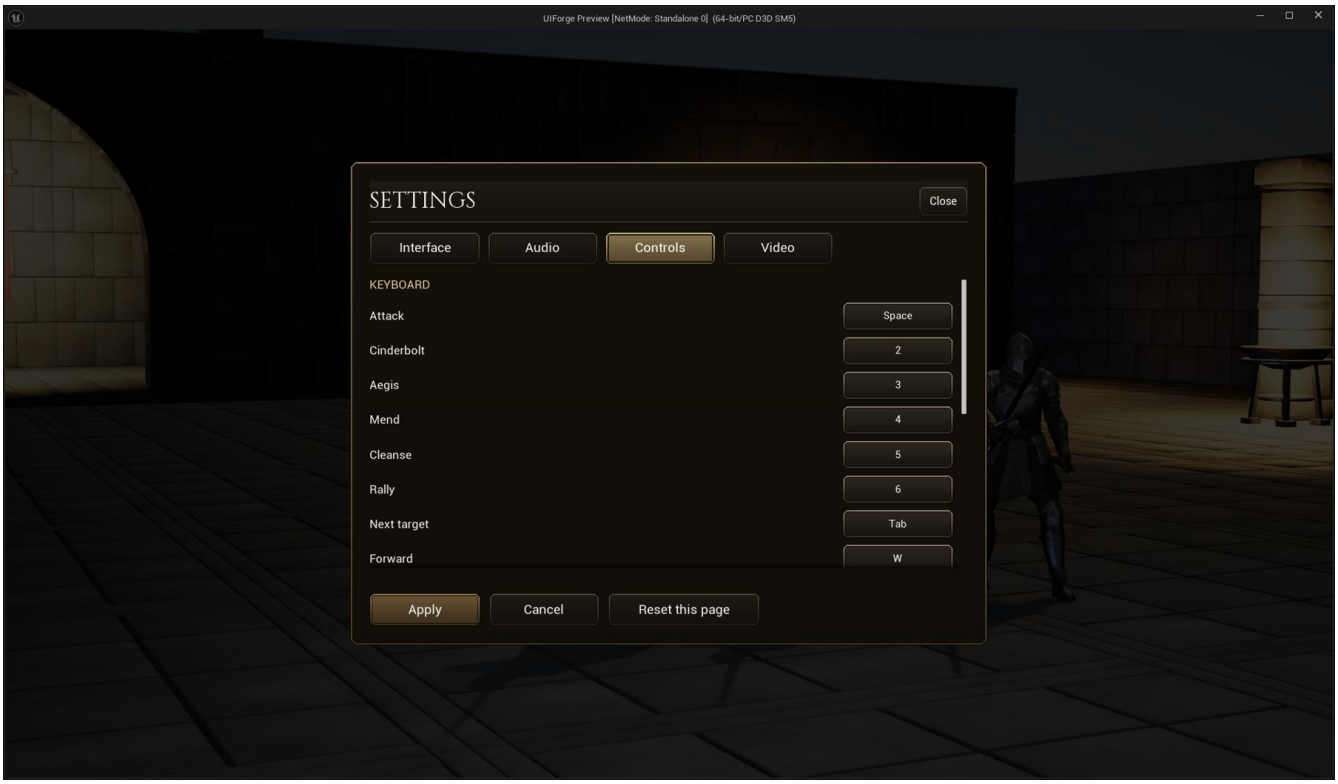
One settings system, four editable pages



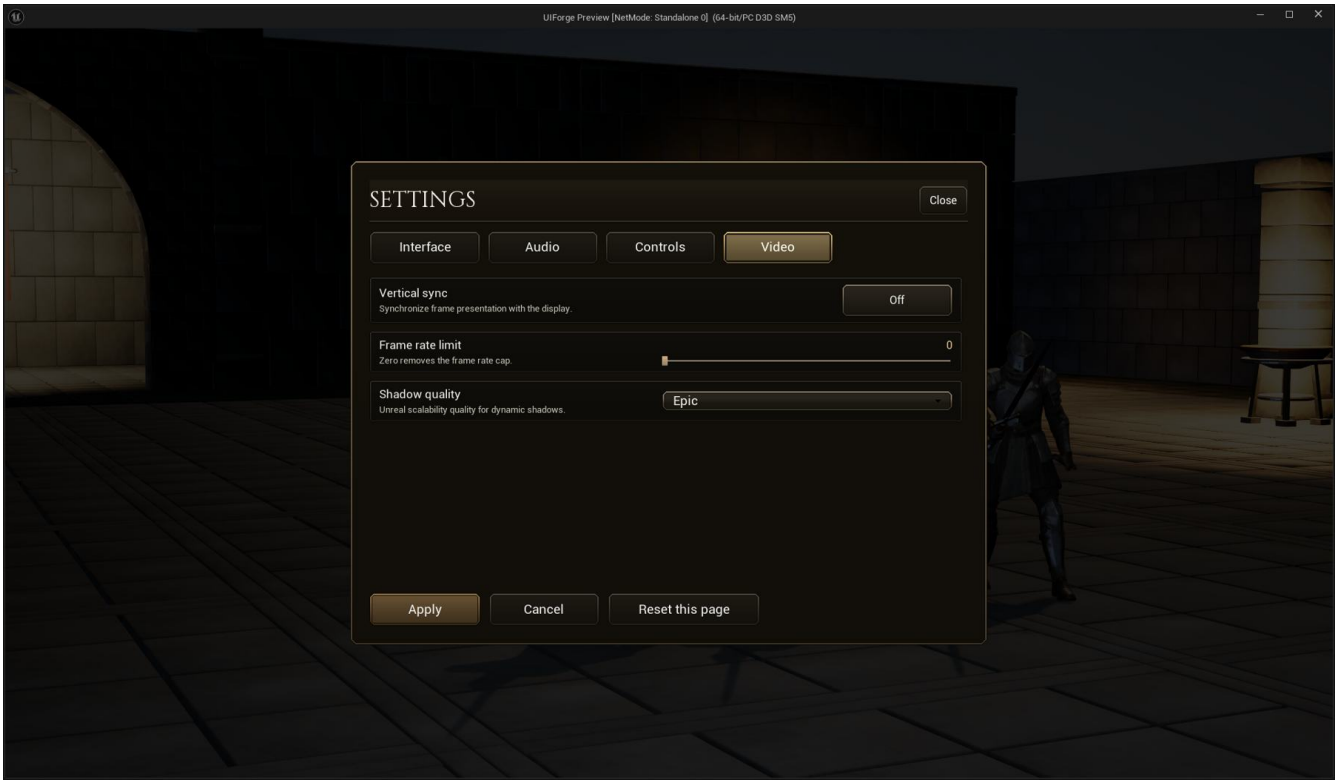
Interface settings with scale, text size, motion, tooltip delay and opacity



Separate UI and game audio controls



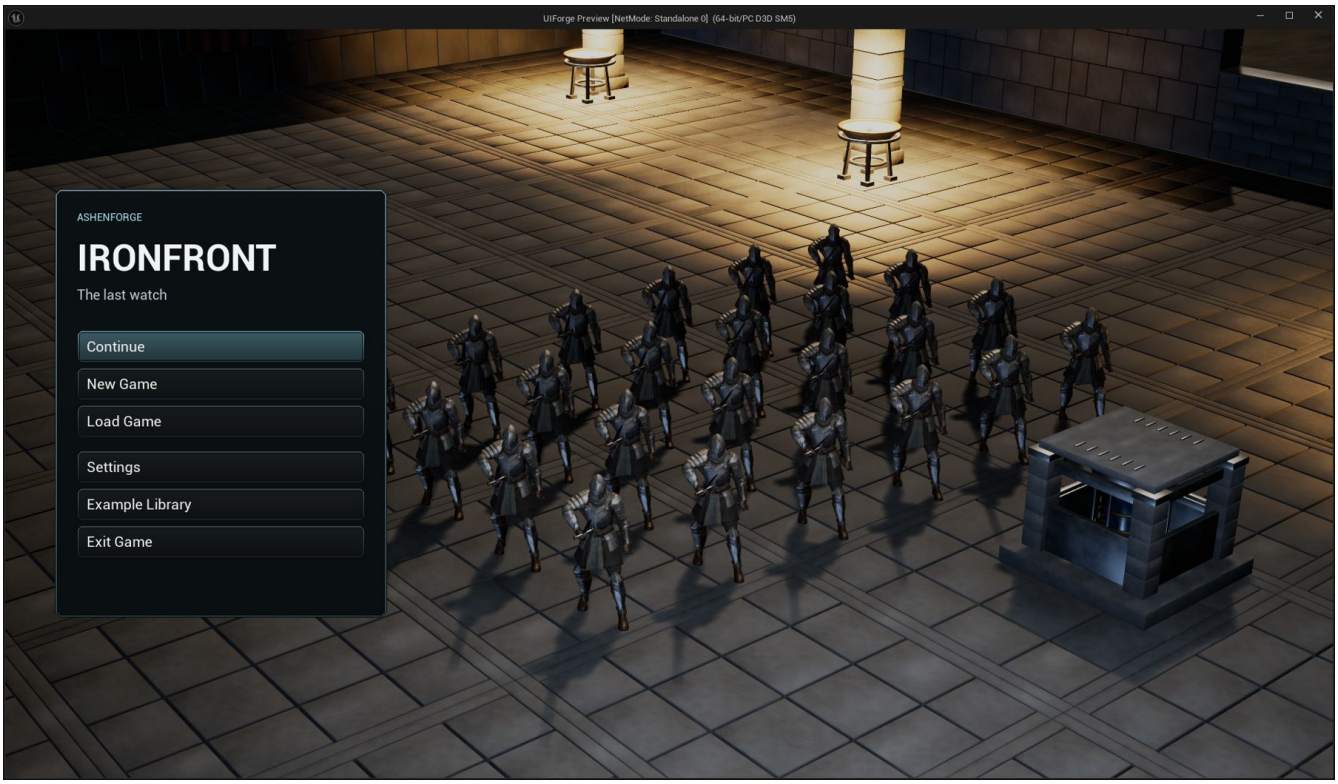
Enhanced Input rebinding within the same draft session



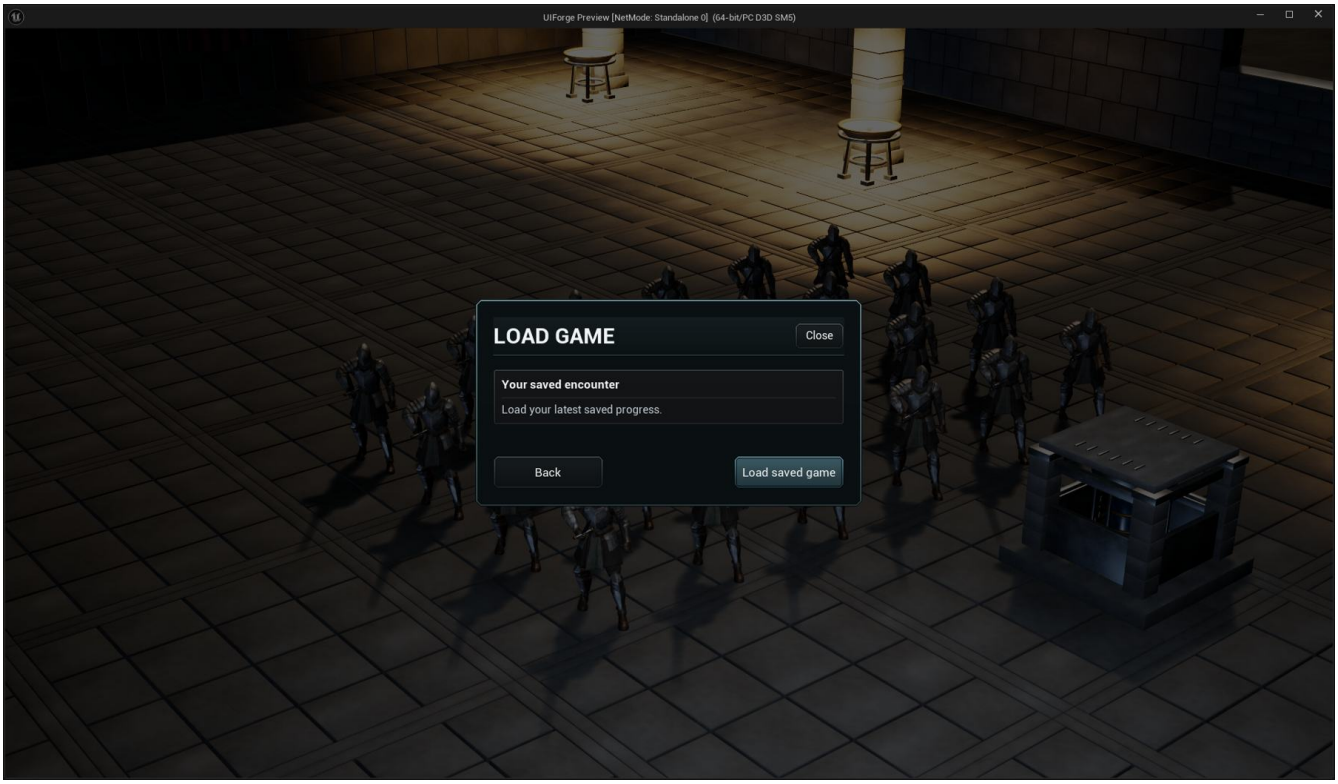
Supported engine video options with shared Apply, Cancel and page defaults

Scale/audio can preview immediately; Cancel restores prior presentation. Apply commits supported values. Remapping uses registered Enhanced Input contexts and the same footer. Build this settings menu shows its actual Designer hierarchy, definitions, binding rows and Blueprint provider.

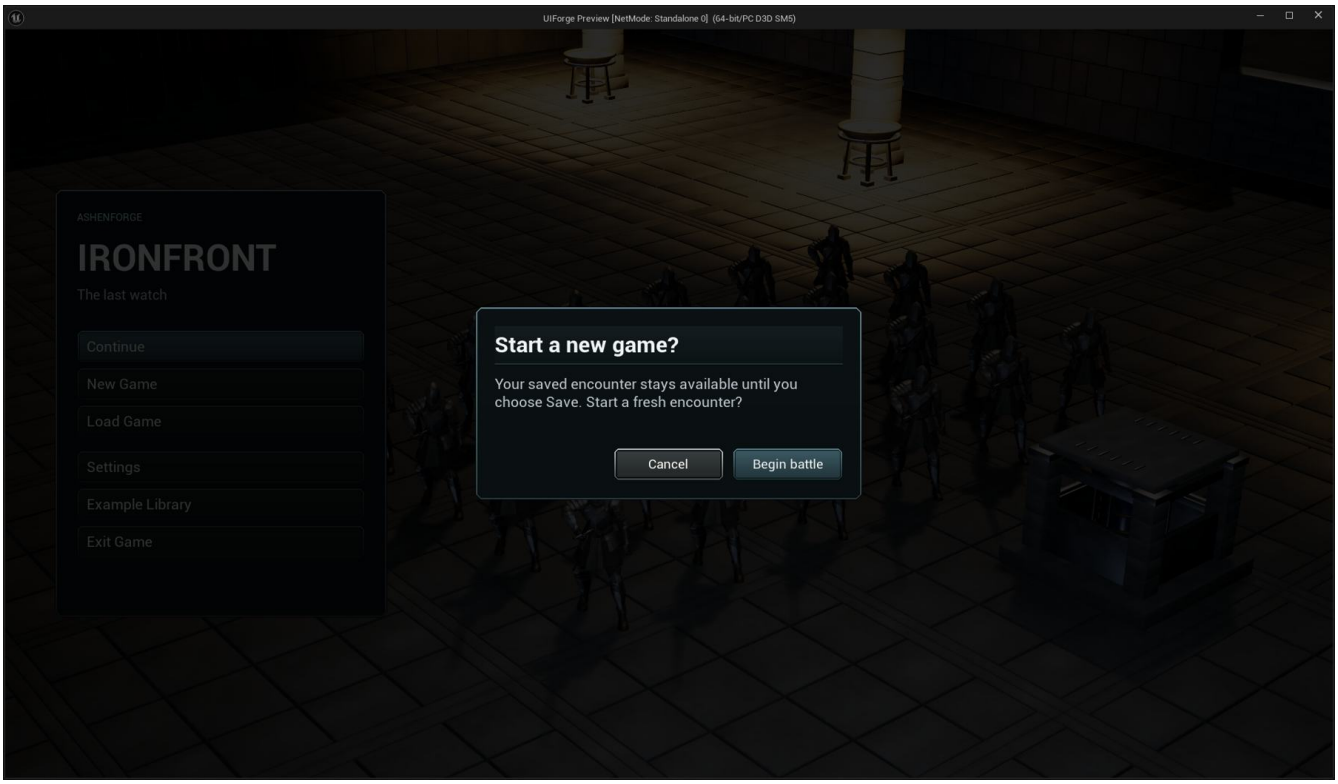
Ironfront: an army interface



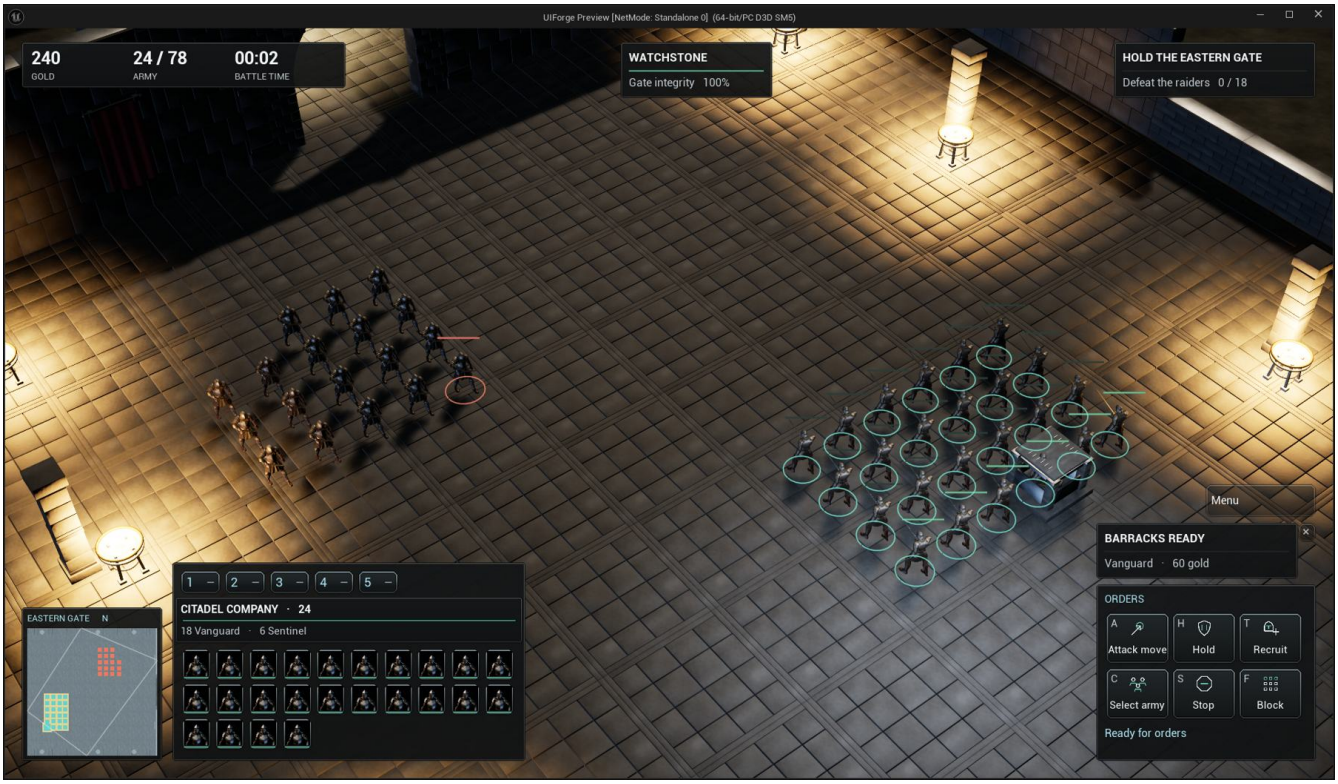
Current RTS main menu



RTS load window using the shared load-screen contract

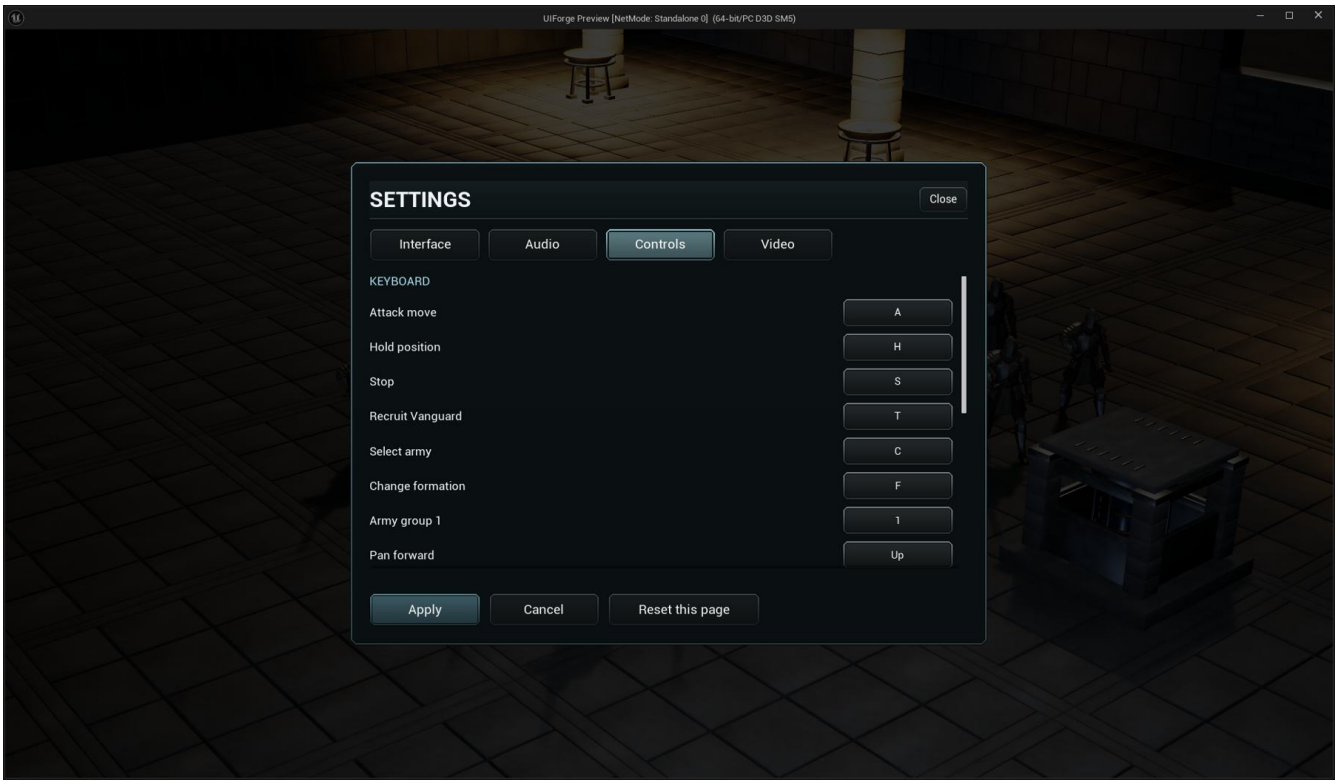


RTS new-game confirmation



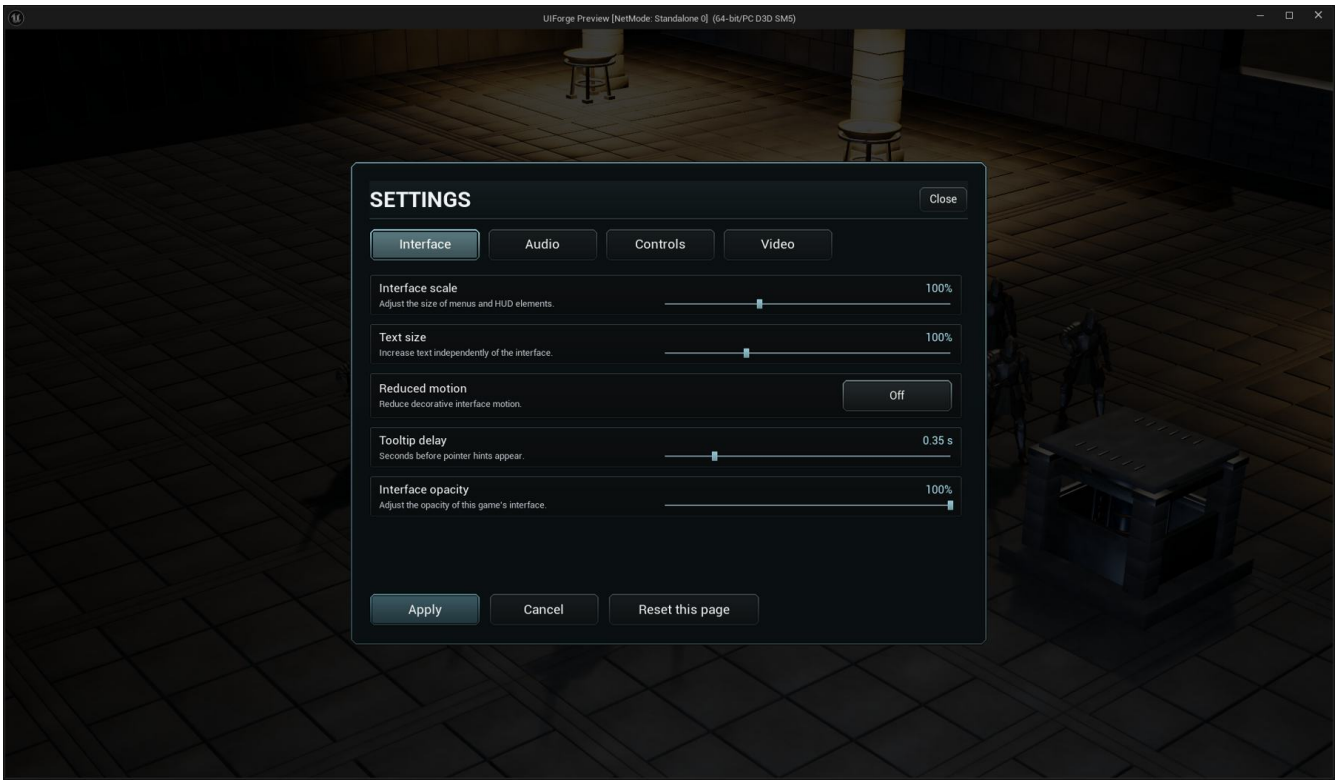
Army HUD with 24 selected units, roster, groups, resources, minimap, orders and recruitment

Click a unit or drag a selection rectangle, Shift-select additively, then right-click a destination. Groups, formations, queued orders, attack-move, hold/stop, recruitment and cancellation belong to the example backend. Roster/context controls observe selection changes rather than polling the full population per widget.

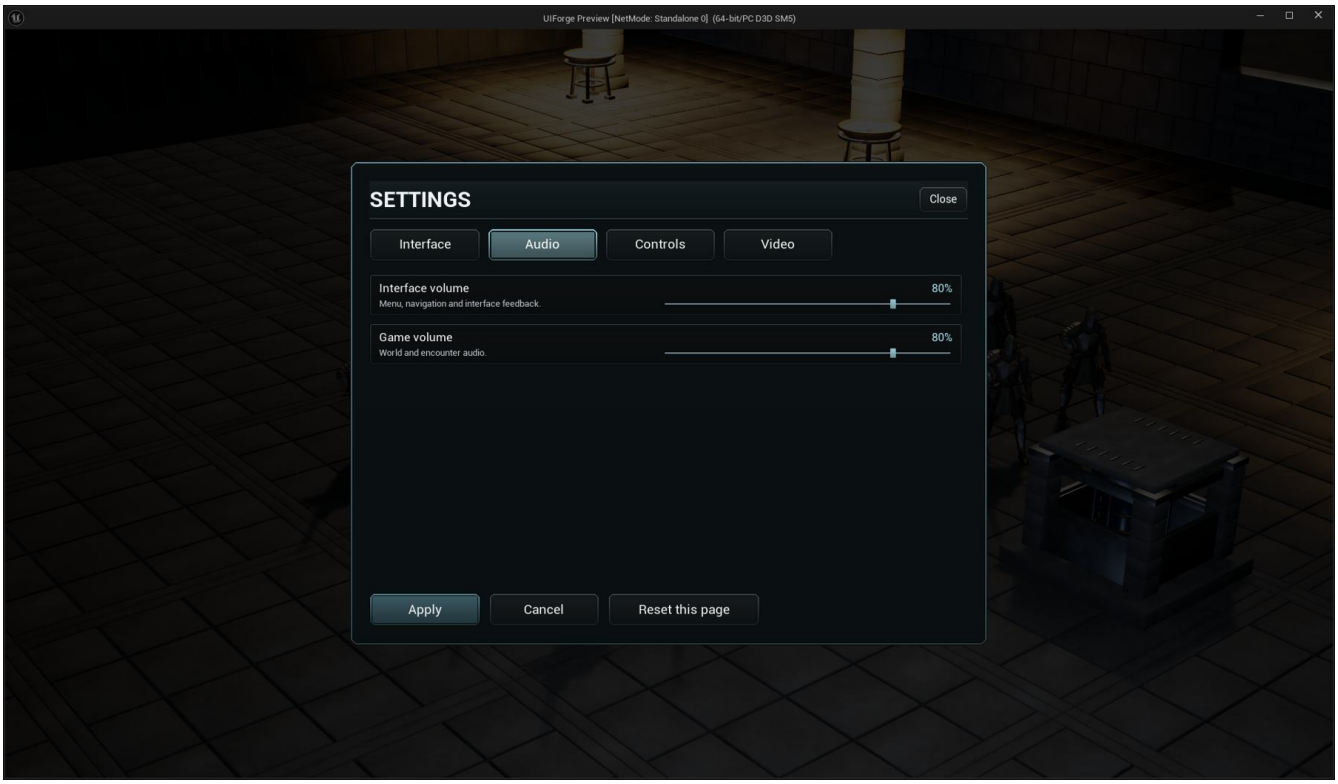


RTS Controls page with its own registered mappings

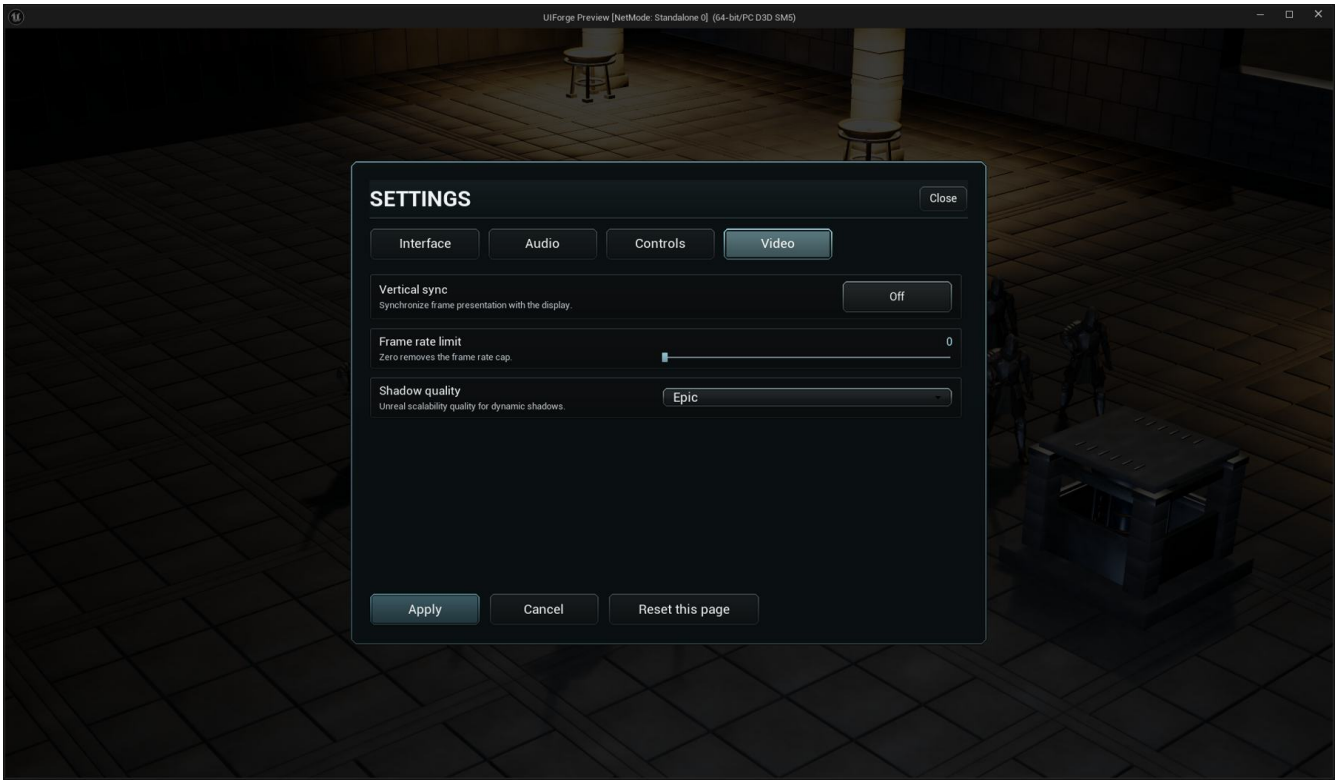
The same four-page settings screen uses the RTS theme. The game-specific difference is its registered command/camera mappings.



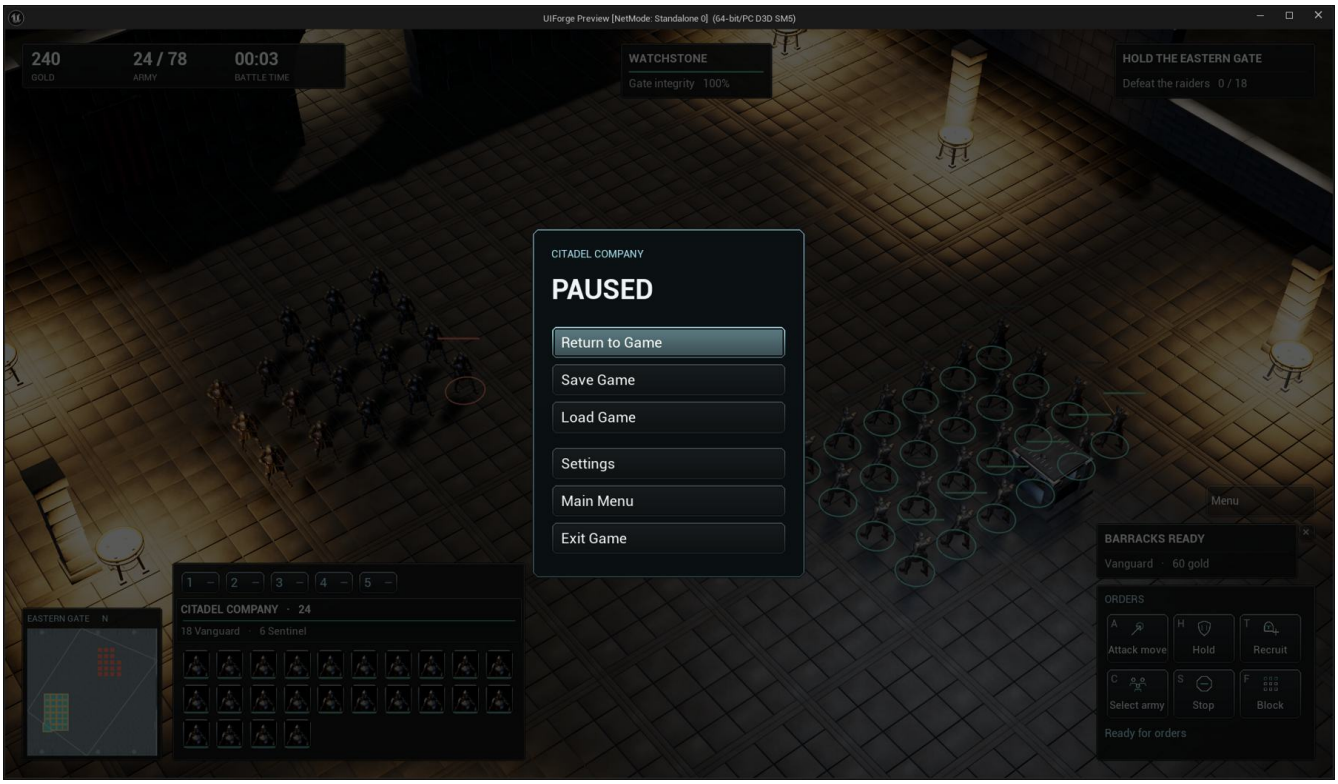
RTS Interface page



RTS Audio page



RTS Video page



RTS pause using the same screen and action contracts

Build your own interface

Follow visual game integration, visual settings, then theme, button, entry and HUD customization. The complete scheme supports code/asset lookup; normal authoring does not require reading internal source.

These are original AshenForge presentations. Game Master's exclusive design is not distributed. Inventory mechanics remain outside InterfaceForge. Further visual refinement and outstanding platform/device checks are recorded separately from functional history in current status.

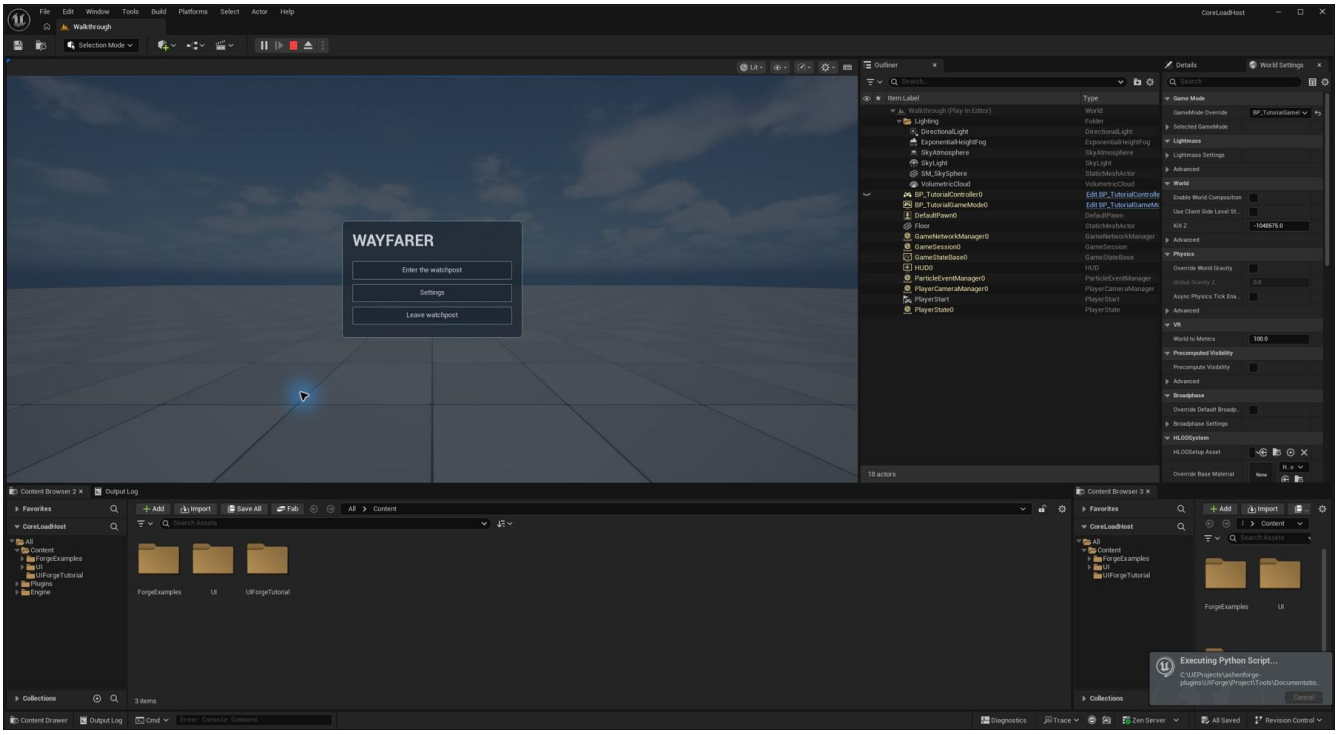
START HERE

Implement UIForge in your game

Follow the illustrated integration route: exact locations, real Designer/graph images and expected results. This page supplies the detailed contract/reference.

This walkthrough builds a local-player interface in an ordinary Unreal project: main menu, a gameplay HUD with an observable Blueprint record, pause, a confirmation dialog, and the four-tab settings menu. The game keeps its own controller, pawn, rules and saves. UIForge supplies presentation, CommonUI layers, focus, styling and settings coordination. No inventory, MassForge or ForgeExamples runtime module is required.

Verified on UE 5.8.3 / Win64: the saved controller, game mode, screens, custom entry and settings run in a separately copied project with only InterfaceForge installed, without any ForgeExamples C++ module, MassForge or inventory. Native mouse input completed main → HUD/entry → pause → settings → confirmation → main. The custom opacity and an H keyboard remap survived restart. The playable MMO/RTS examples are separate, fuller demonstrations; this host is deliberately a small integration test, not another showcase or an MMO backend. Physical controller tests remain deferred.



Actual independent project running the Blueprint tutorial

1. Install into an ordinary project

Use Unreal **5.8.3 CL58210709 on Win64** for the documented test target. The plugin descriptor's compatible version is **5.8.0**, matching this installation's compatible-engine baseline. That field is not a claim that every 5.8 patch/platform was tested. Source installation requires the Windows C++ build tools even when the game's assets are Blueprint-only.

Close your game's editor. Put `InterfaceForge` under `<YourGame>/Plugins/`, with its `.uplugin`, Source, Config, Content and Resources. Exclude development Intermediate/Saved and stale binaries from another engine. Enable InterfaceForge, CommonUI and EnhancedInput, then rebuild/reopen. CommonInput is brought in by CommonUI. Optional adapter directories are not part of the core install.

For an existing game, keep its GameMode/PlayerController and add the local setup below to their existing Blueprint extension points. Do not replace a working possession, network or input system simply to add menus. For a clean project, create `BP_TutorialController` from PlayerController and `BP_TutorialGameMode` from GameModeBase; select that controller on the GameMode and assign the mode in your map's World Settings.

Expected result: the plugin's classes appear in the parent-class picker and the Content Browser can show `/InterfaceForge/UIForge` when **Show Plugin Content** is enabled. A missing plugin mount or skipped incompatible plugin is a setup failure, even if the game's native module links against InterfaceForge.

2. Configure CommonUI and input

Set these Project Settings before testing navigation:

Setting	Value / reason
Engine → General Settings → Game Viewport Client Class	CommonGameViewportClient; gives CommonUI its normal input routing
Common Input Settings → Input Data	IFNavigationInputData, or a compatible project replacement; default Enter/Escape and controller Accept/Back
Common UI Settings → Common Button Accept Key Handling	Trigger Click; accepting a focused IFButton produces its click event
Enhanced Input → Enable User Settings	Enabled; registered player-mappable contexts populate remapping
Input default classes	EnhancedPlayerInput / EnhancedInputComponent when using Enhanced Input

Choose your project's controller data through CommonInput platform settings. The included `BP_IFGenericController` and original geometric face glyphs are a generic example, not licensed console branding or proof of a specific device. Keyboard prompts resolve actual Enhanced Input mapping names rather than hard-coded display strings.

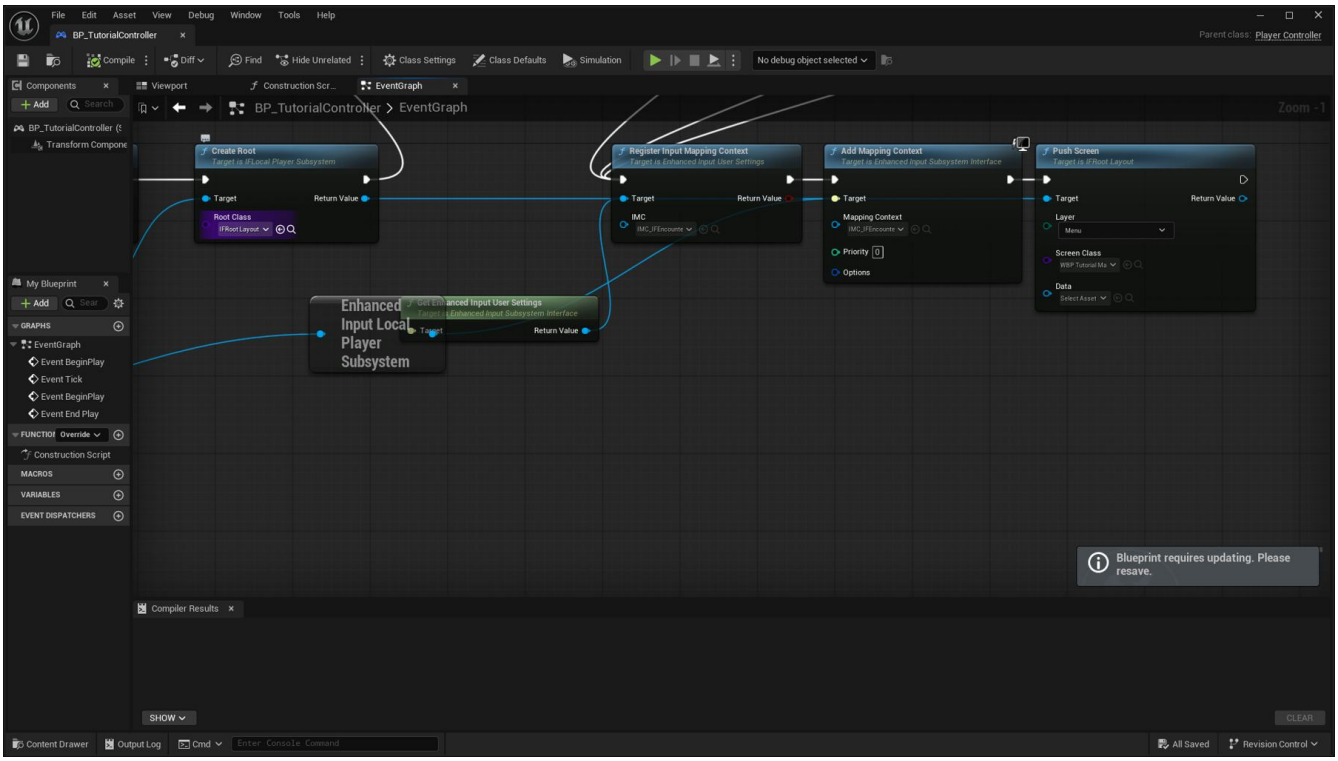
IFScreen's **Contain Navigation** defaults on for Menu-policy screens and modals. It stops unhandled directional/Tab navigation at the screen boundary; child explicit/wrap rules still work. Disable it only when deliberately sharing a focus region. This prevents a dialog-edge direction from selecting a covered menu button.

In your local controller's **Event BeginPlay**, branch on **Is Local Controller**. From Self, use **Get Enhanced Input Local Player Subsystem**. **Get Enhanced Input User Settings** supplies the settings object; check it is valid before calling registration. Call **Register Input Mapping Context** on the returned user-settings object, with your player-mappable context on **IMC**. Call **Add Mapping Context** on the Enhanced Input subsystem with the same asset and your chosen Priority. Registration makes rows available for settings; adding the context enables actual gameplay input. Do not register a parallel list of imaginary keys.

Existing games should register their own context once, preserve existing priorities and avoid **Clear All Mappings**. Keep a hard reference to the context from your controller or another cooked asset. A context loaded only by an untracked string can work in the editor and be missing from a package.

3. Create the local root

In the local BeginPlay branch, use **Get IFLocalPlayerSubsystem** from Self. Call **Set Theme** with your IFTheme data asset, then **Create Root** with Root Class `IFRootLayout` or your root Widget Blueprint. Retain the subsystem or use **Get Root** when needed; do not create an extra root for each menu.



Actual controller graph: create the local root through the public Blueprint API

Blueprint workflow (text source; interactive diagram in the web guide)

```
flowchart LR
  A[Event BeginPlay] --> B{Is Local Controller}
  B -->|true| C[Get IFLocalPlayerSubsystem from Self]
  C --> D[Set Theme]
  D --> E[Create Root: IFRootLayout]
  E --> F[Push Screen: Menu / WBP_TutorialMain]
```

Push Screen Target is the returned root; Layer is **Menu**; Screen Class is your saved IFScreen Widget Blueprint; Data is your presentation adapter or None. The root creates HUD, Menu and Modal stacks by default and includes a Safe Zone. A Designer-built root can instead register its own Common Activatable Widget Stacks through **Register Layer**. Keep the layers' visual/input order consistent.

In **Event EndPlay**, call **Remove Root** on the local-player subsystem before abandoning this controller. Before **Open Level**, remove the old root and let the next local controller initialize its own. Do not carry stale widgets, world actors or old data subscriptions through travel. For server-only controllers there should be no local UI setup.

Expected result: one root per local player, one visible menu, and keyboard Accept/Back handled by CommonUI. Avoid calling engine **Set Input Mode** every tick; IFScreen's Input Policy supplies CommonUI's desired mode.

4. Design the main menu

Create `WBP_TutorialMain`, parent **IFScreen**. In Designer, use a full-screen Canvas/Overlay, a scrim, a Scale Box set **Scale To Fit / Down Only**, a bounded Size Box, IFSurface and a Vertical Box. Use IFText for the title and IFButton-derived widgets for the actions. Keep ordinary navigation targets about 48 logical units high; tune typography and padding against real viewport sizes instead of enlarging the whole tree.

Set **Input Policy = Menu**, **Initial Focus Name = PlayChoice**, **Fallback Focus Name = SettingsChoice**, and **Show Pointer Cursor = true**. Names are Designer variable names; nested focus targets can use a path such as `HUDLayout/Pause`.

For each IFButton, select **On Button Base Clicked** in the graph. SettingsChoice calls **Get Root Layout → Push Screen**, Layer Menu, Screen Class your IFSettingsMenu Blueprint. It does not call Create Widget/Add to Viewport directly. For PlayChoice, construct your own data-source object, push the HUD on the **HUD** layer with that object on Data, then **Deactivate Widget** on the main menu. Own the adapter in an appropriate controller/game instance variable if it must survive screen replacement; a screen's Data reference also retains it for that screen's lifetime.

Keep instructional feature descriptions in the developer guide. The menu should look and behave like a game's entry point. Continue/Load should only be enabled when your game's save system reports a valid save; UIForge does not invent a gameplay save format.

5. Connect a gameplay HUD through an adapter

Create a Blueprint implementing **IFDataSource**, using your existing actor, component or UObject. Implement:

- **Get UIModel(Model Name)**: return a stable UObject for that named display, or None. Do not allocate a new object for each query.
- **Request UIAction(Action Name, Context)**: validate the requested action in the game and return whether it was accepted. UI widgets cannot grant authority to a client.

Use IFPresentationData for common Title, Detail, Fraction, Quantity and Enabled fields, or your own observable type. When gameplay changes, update the model and call **Notify Data Changed**. Make health/resource models reflect actual game events; do not scan the world from each widget Tick.

The independent tutorial reuses the saved **BP_WatchpostSource** as a small Blueprint adapter. Its `Record` action validates Context and a four-observation limit, changes Quantity/Fraction/Detail, then notifies. Replace this with your health/objective/interaction system by changing the adapter, not by rewriting the entry lifecycle.

Create `WBP_TutorialHUD`, parent **IFScreen**, **Input Policy = Game And UI**. Place an IFHUDLayout inside your project's Safe Zone and assign an IFHUDLayoutProfile with named regions. Call **Set Region Widget(Region, Widget)** for your developer-created widgets; check its Boolean for a missing region. Regions control anchors, size, offsets and optional compact behavior. Alternatively use ordinary UMG anchors for a small fixed arrangement. Avoid applying an extra manual DPI factor: the project's DPI curve and UIForge's UI scale already compose.

HUD **Event Initialize Screen** must first execute **Parent: Initialize Screen**, forwarding **Data**. Then **Get UIModel**, Model Name `Record`, supplies **Initialize Entry** on your entry variable. Accessing Screen Data before the parent call is a common initialization error.

6. Create your custom reusable entry

Create a Widget Blueprint with parent **IFEntryWidget**. Design its own IFSurface, IFText fields, progress and optional IFButton. The entry is not a special game-specific C++ subclass.

Event / node	Blueprint work
On Entry Initialized	Optional one-time work for the newly assigned Data
On Entry Refreshed	Entry Data → Cast to IFPresentationData → set text, fraction, quantity, availability
On Entry Selection Changed	Change a semantic text role/outline and Refresh Appearance
On Entry Released	Clear visible transient state and release your own additional subscriptions
Entry button click	Find Parent Widget Of Type(IFScreen), execute it, Cast to IFScreen → Request UIAction(Action Name, Entry Data)

The base subscribes to IFPresentationData change notifications and removes the old subscription on reassignment/release. For other model types, bind your own event and unbind that exact object during release. Do not subscribe to the entire entity population. The engine's List View/Tile View owns pooling and virtualization; assign your entry class, then **Add Item** or **Set List Items** with the models. Selection is model-based; do not retain pooled visual rows as gameplay identity.

The saved WBP_WatchpostEntry demonstrates this lifecycle and a project-authored segmented record control. A different entry can be assigned to a List View or region without changing the adapter contract. Entry recipe and customization show the real event graphs.

7. Pause, dialogs and return

Create `WBP_TutorialPause`, parent IFScreen, Input Policy Menu, initial focus Resume. A HUD Menu button uses **Get Root Layout → Push Screen(Menu, WBP_TutorialPause, Data)**. Override HUD **On Handle Back Action** to push that same menu and return true, so Escape/controller Back opens pause rather than destroying the HUD. Pause's Resume calls **Request Back**. CommonUI restores the underlying screen's navigation target.

For an actual single-player pause, have your game's controller/adapter call **Set Game Paused(true)** when opening pause, and restore false on Resume/return/travel. Keep UI input able to execute while paused. Do not pause a network server merely because one client's local menu opened. The tutorial's saved Blueprint HUD and pause screen use these game-owned nodes; stack activation by itself does not pause the world.

Confirmation: **Construct Object from Class(IFDialogData)** with a retained owning UObject as Outer. Set Title, Message, Confirm Label and Cancel Label. **Bind Event to On Completed** before calling **Get Root Layout → Push Dialog(Data)**. In the handler, branch on Result **Confirmed** before acting. **Cancelled** must retain the current game. The modal defaults focus to Cancel, reports exactly once, consumes Back and keeps input out of underlying screens. Replace **Default Dialog Class** on your root with an IFDialog Blueprint to change the whole visual.

For Return to Main Menu, the confirmed game adapter removes the root, unpauses if appropriate, then opens your menu level. For an in-world menu, clear HUD/Menu layers and push the menu again. For Exit, call your platform's normal Quit Game path only after confirmation. UIForge does not save progress automatically when a dialog closes.

8. Build the four-tab settings menu

Follow Build a settings menu in Blueprint. Create an IFSettingsMenu Widget Blueprint with **SettingsPages** (Widget Switcher), **Page Ids** in matching order, IFSettingsTabButton headers with Page Index, reusable IFSettingRow widgets with Definition, and **ApplyChoice / CancelChoice / DefaultsChoice / SettingsStatus**. These optional variable names attach behavior to your Designer tree; the native class does not build the menu layout.

Use Interface, Audio, Controls and Video pages. Set **Include Input Bindings** for IFBindingEntry rows tied to your registered mapping names. The same footer owns preference and binding drafts across tabs. Back/Cancel reverts preview and discards keys; **Reset this page** changes only that category's draft; Apply commits and returns only on success. Failures remain on screen. Do not embed a standalone IFRemappingScreen that commits separately.

For a project-specific setting, create IFSettingDefinition with Custom binding and a Blueprint IFSettingProvider. Implement **Read Value, Validate Value, Preview Value, Commit Value, Restore Value**. The saved BP_IFOpacityProvider demonstrates a real SaveGame and root opacity preview/rollback. Create the provider after your root at startup and Read/Validate/Preview its saved value to apply it outside the settings menu. Use the owning Player pin and stable Setting Id; never change another local player's root.

Create IFAudioRouting with a SoundMix and distinct sibling UI/game SoundClasses, and call **Set Audio Routing** on IFSettingsSubsystem during setup. Assign theme feedback sounds to the UI class and gameplay sounds to the game class. Test each volume independently, including mute and Cancel. For UI preferences choose **Set Storage Profile** before opening an edit session, not during editing.

Video rows use Unreal **GameUserSettings**. Current builtin rows cover VSync, frame limit, normalized resolution scale and scalability quality. They deliberately do not change display resolution/window mode. If your game adds those, use Apply Resolution Settings plus a timed Keep/Revert dialog and engine **Confirm Video Mode / Revert Video Mode**; never save an unconfirmed mode just because a slider changed. SaveGame, input profiles and engine config are separate writes; this workflow does not advertise an atomic multi-file transaction.

9. Style, motion and progress

Create an **IFTheme** data asset or duplicate an included theme. Assign fonts, typography, palette, spacing, focus/confirm sounds, transition duration and button feedback. **Set Theme** updates the local-player interface; IFText/IFSurface/IFButton and setting controls follow it. Setting rows can disable **Follow Theme Colors** to retain Designer styles. Use localized FText labels/help, allow prose to wrap, keep numbers/key labels unwrapped, and test the longest supported language.

Use Widget Blueprint animations for project transitions and **On Input Presentation Updated** for layout-specific presentation. Honor **Reduced Motion** and expose feedback strength as game settings when adding camera effects. Do not add an independent camera-shake system merely to populate a UI example.

Progress is presentation data, not an ability backend. IFCenterTint reproduces the inspected Sakura center-origin interior tint; overlay it on your icon and drive **Set Remaining Fraction**. Fraction 1 covers the interior, 0 uses passed opacity throughout. Tint, Remaining Opacity, Passed Opacity, Start Angle and Reverse are editable; **Refresh Tint** applies property overrides to that image's material instance. Do not substitute an assumed square-perimeter ring. Actual material controls document its parameters. IFProgressWidget supplies other linear/radial/square/segmented/charges presentations through **Configure Progress**, **Set Fraction** and the documented cooldown controls.

10. Verify input changes in the actual flow

Use the common input-presentation service throughout menu, HUD, popup and custom widgets. IFScreen owns cursor/input policy and top-screen focus; do not give every row an independent device detector. IFActionPrompt resolves its **Mapping Name** against applied Enhanced Input mappings and CommonInput controller data. **On Input Presentation Updated** can hide pointer-only decoration or show controller hints; it is not a place to rebind gameplay input.

Test keyboard/mouse → controller → keyboard/mouse on main menu, settings with a key selector, HUD, an open modal, and immediately after transitions. Check restored focus, Back, prompts, cursor, reserved/conflicting keys, insignificant mouse movement, analog drift thresholds and disconnect fallback. Software events can verify routing logic; they cannot verify a physical controller, touch digitizer, console or mobile platform. Physical-controller verification is explicitly deferred because no controller is available.

Unreal's default **Escape** → **Stop Play** editor shortcut can end PIE before the game's Back action receives the key. That happened in the independent host. Test the actual game Back behavior in standalone/packaged play, or deliberately adjust the editor's Stop shortcut for your own testing. Do not change the player's runtime Back mapping to work around an editor command.

11. Cook and test independently

Keep your maps referenced through packaging settings; retain hard/cook references to root/screens, contexts, themes, row classes and custom providers. A project-owned Primary Asset Label can include soft UI classes; its Asset Manager scan rules must include the label. Do not cook the entire development repository to conceal missing references.

Build the game, open a fresh process and verify main menu → settings → HUD → pause → return. Apply a scale and key, restart, and confirm the actual effects and prompt. Cancel a second edit and confirm the committed values remain. Run the same saved Blueprint row/provider in the independent host with no ForgeExamples/MassForge/inventory module. Test failure feedback using an isolated test save profile; never obstruct the player's real save slot.

Capture real Designer, readable graph and runtime images as evidence. Compilation proves types/graphs, not navigation or visual quality. At 1280×720, 1920×1080 and a narrow viewport, inspect text, focus rings, clipping, dropdowns, transitions and tooltip edges at normal and enlarged UI/text scale. Inspect the cooked build as well as PIE.

Troubleshooting

- **Blank screen:** ensure the local BeginPlay branch ran, Create Root succeeded, Screen Class is cooked and Push Screen uses the returned root. Check skipped-plugin/missing-package errors first.
- **No focused button:** Initial Focus Name must resolve to a visible enabled target on the active screen. Check CommonGameViewportClient and Trigger Click.
- **Bindings unavailable:** enable Enhanced Input user settings, execute Get Enhanced Input User Settings, register the player-mappable context and use its exact Mapping Name.
- **Entry never updates:** keep a stable model, initialize after Parent Initialize Screen, and call Notify Data Changed after mutation. Do not bind a temporary snapshot.

- **Cancel leaves a change:** route controls through the shared session; a custom provider must undo its own preview and partial commit. Avoid separate child Apply handlers.
- **World/HUD input leaks through a dialog:** push the dialog on Modal with Menu policy; do not Add to Viewport an unregistered widget beside the stacks.
- **Packaged prompt/action missing:** verify cooked references to the context and controller data, then test the applied mapping in a fresh process.

Keep the settings composition, current product status and verification pages alongside this guide for version-specific tested scope.

Character presentation in the separate MMO/RTS game is covered in Example character animation, including the editable Animation Blueprint and action contact markers. Your game can keep its own character animation system.

START HERE

Build a genre HUD with public tools

These APIs are part of the active UE5.8.3 rebuild. Follow current status for verification boundaries. Examples are saved Designer trees under `/Game/ForgeExamples/UI`; their encounter rules live in ForgeExamples, outside the plugin.

For the revised army roster, per-entry command context, selection overlay, mixed HUD input, automatic-height composition and smooth progress updates, follow army and RPG HUD authoring.

Screen, data and commands

Create a Widget Blueprint with parent **IFScreen**. Use **Get Local Player Subsystem (IFLocalPlayerSubsystem) → Create Root → Push Screen**. Choose HUD, Menu or Modal and supply your data-source object. Implement **IFDataSource** on that object: **Get UI Model** returns a stable observable object; **Request UI Action** validates and performs the game's requested action.

Add **IFCommandButton** in the Designer, set **Action Name**, and optionally **Model Name**. The containing screen routes the command and observes availability. **Use Model Label** controls whether the button takes its label from the model. The model's **Unavailable Reason** explains disabled actions. A command button owns no gameplay rules.

Set a screen's **Initial Focus Name** to a Designer name or nested path such as `HUDLayout/AbilityEntry2/EntryCommand`. **Fallback Focus Name** is used when the preferred control is unavailable. **Back Action Name** optionally routes Back to the adapter (for example Pause or Resume). Leaving it empty uses normal CommonUI behavior. A command button's **Request Screen Back** invokes that handler directly without adding a Back action to your gameplay source. Menus remain activatable screens; passive HUD regions remain ordinary widgets.

Optional screen **Shortcuts** associate Mapping Name with Action Name. They read applied Enhanced Input keys and keep a HUD action such as Tab targeting from being swallowed by widget navigation after a click. They run only on the active top screen outside stack transitions. Editable-text/key-selector focus is excluded. Disable **Shortcuts Enabled** while a custom editor/chat control owns input, or put that control on another screen. Gameplay still validates the requested action.

For a Game And UI HUD with focused buttons, set **Gameplay Analog Keys** to axes reserved for gameplay, such as Gamepad Left X/Y. The existing input observer passes these explicit axes to the owning Player Controller before Slate turns them into navigation. D-pad navigation remains available. Leave this array empty on menus, sliders, chat screens, or screens that need the stick for UI controls. Only the top screen outside transitions can forward axes. Opening another screen, deactivation, destruction, or leaving controller presentation releases forwarded values; movement must not remain held behind a dialog. This opt-in does not create gameplay mappings or movement logic.

Named layout regions

Create a Data Asset of class **IFHUDLayoutProfile**. Add Regions with Name, Anchor, Alignment, Offset, Size and Z Order. Values are Slate units; engine DPI and the player's UI scale apply outside the profile. Compact Below Width and Compact Scale provide a smaller layout at narrow logical widths. Hide When Compact is for optional decoration, not essential controls.

Create a Widget Blueprint with parent **IFHUDLayout**, put a Canvas Panel at its root, and name its direct children to match the regions. Assign **Profile** on the actual Designer instance inside your HUD screen. Use one Safe Zone in the root composition; the shipped root already supplies it. A class default alone does not replace an instance's explicit None value.

Public nodes:

- **Set Region Widget** replaces a named region with any developer-created widget.
- **Set Layout Profile** changes composition without replacing the widget tree.
- **Override Region** changes an individual region for the current session.

- **Reset Region Overrides** restores the profile.
- **On Layout Applied** lets Blueprint respond to compact mode.

Expected result: moving the Actions region in the data asset moves the entire action group. Replacing its widget does not change command routing or input presentation. Invalid names/sizes are rejected rather than silently creating broken geometry.

Readout or custom entry

Use **IFReadoutEntry** as an optional parent, or implement **IFEntry** on your own Widget Blueprint. The entry contract is **Initialize Entry** → **Refresh Entry / Set Entry Selected** → **Release Entry**. Release subscriptions and transient state when recycled; the same widget can display a different object later.

IFReadoutEntry recognizes optional children named **TitleText**, **DetailText**, **ValueText**, **IconImage**, **FractionBar**, **CooldownTint**, **EntryCommand**, **ActionPrompt** and **SecondaryEntry**. Use any subset and build the rest of the Designer tree yourself. Set **Model Name** for automatic screen binding, or call **Initialize Entry** yourself. Bindings reattach when a screen is reused and release when it deactivates.

SecondaryEntry is another IFReadoutEntry. Set the parent's **Secondary Model Name** to bind a related model; None hides the secondary row. The sample **WBP_IFUnitFrame** uses it to combine player health and Resolve in one compact frame while the same Blueprint displays targets and party members without a resource row. The portrait is a static sample texture, not a per-widget scene capture. Replace IconImage or the whole entry in Blueprint.

Set **Mapping Name** to an existing player-mappable Enhanced Input row for the optional ActionPrompt. **Icon UV Min/Max** select an atlas rectangle; default 0..1 uses the entire texture. **Override Fraction Colors** gives health, resources and hostile targets distinct colors without duplicating the entry Blueprint. Call **Refresh Screen Binding** after changing binding options at runtime. The adapter owns soft-icon residency; a UI update never synchronously loads it.

For a custom command-button tree, add an optional **ActionPrompt** child and set the button's **Prompt Mapping Name**. Prompts use CommonInput glyphs where configured. Theme **Compact Key Labels** supplies readable fallbacks for keys such as shoulder/trigger buttons; changing the mapping selects the fallback for the new applied key.

Set **Tooltip Model Name** when a command needs richer information than its compact readout. The sample ability models supply description, cost, range and the current unavailable reason through a separate presentation model; no ability backend belongs to the UI plugin. **Hide When Unavailable** is useful for a contextual defeat panel or an active-status readout. Its subscription remains attached while hidden, so a later model update can show it again.

The MMO ability entry layers are:

```
IFReadoutEntry
  Canvas
    EntryCommand     - actual CommonUI interaction
    IconImage        - atlas rectangle, ignores hit tests
    CooldownTint     - center-origin opening, ignores hit tests
    DetailText       - countdown, visible while fraction > 0
    ActionPrompt     - current applied key or configured controller glyph
    TitleText        - short action name
```

Set the remaining fraction from your source; this widget does not start an ability or an authoritative timer. See material parameters and the inspected behavior.

The sample **WBP_IFAbilityButton** is a Blueprint child of IFCommandButton. Its Designer root is an **IFTooltipAnchor** named TooltipAnchor containing a surface named FocusBorder. Pointer Placement and Navigation Placement are both **Centered Above Anchor**. This is how the example keeps descriptions above the action bar without a special native ability-button class. The RTS world entry **WBP_IFRTSWorldLabel** contains only FractionBar; the MMO variant adds TitleText. Both receive the same IFEntry initialization/update/cleanup contract.

Minimap

Add **InterfaceForge Map**, set **Model Name**, and optionally **Location Action**. Supply an **IFMapData** with normalized marker positions; batch changes then call **Notify Data Changed** once. Set **Background** to your baked map image. The map draws at most 2,048 markers and performs no world scans or scene captures.

On Location Pressed returns a normalized coordinate. A configured Location Action routes the same data object, whose **Requested Location** is updated. Your adapter converts coordinates to its own world bounds and validates the order. Mouse/touch select directly; keyboard/controller navigation can adjust the map cursor and confirm. Terrain baking is example editor tooling, not a runtime dependency.

Controls screen

Create a Widget Blueprint with parent **IFBindingEntry**. Add optional **LabelText**, **KeySelector** (InterfaceForge Key Selector) and **StatusText**. Set Mapping Name, Label and Family on each instance. **Refresh Binding** shows the registered preview key; selecting a key calls the public remapping subsystem's validation and staging workflow. It does not invent a second key-binding file.

The current four-tab menu embeds those entries under **IFSettingsMenu**, sharing its preference/input draft and Apply/Cancel. Follow settings composition. For a dedicated controls-only flow, **IFRemappingScreen** remains an optional parent: Designer buttons **ApplyChoice**, **CancelChoice**, **DefaultsChoice** and text **RemapStatus** attach automatically. That standalone screen owns its own transaction; do not nest it under the shared-menu session. **Apply Bindings**, **Cancel Bindings** and **Restore Binding Defaults** are callable from custom Blueprint events too.

Register your gameplay Input Mapping Context with Enhanced Input User Settings and add it to the local player's Enhanced Input subsystem. The example does this for `/Game/ForgeExamples/Input/IMC_IFEncounter`. Actual gameplay reads these actions, and **IFActionPrompt** reads the applied engine mapping. Staging changes does not change gameplay until Apply. Physical controller behavior remains unverified until hardware is available.

Themes and ownership

Emberfall and Ironfront use saved IFTheme assets. Change typography, surface/text/accent/outline colors, spacing, button padding, sounds and transition duration in these assets. Widgets can select their own theme overrides. Use **Set Theme** for a whole-player change. Respect the existing Reduce Motion and UI/text scale preferences when adding Blueprint animations.

The current atlas uses original SVG polygons/lines/circles, with editable vectors and code/provenance in ContentSource/UI. The previous image-generated atlas and its original provenance are preserved privately and excluded from current release media. Imported castle/knight assets are AshenForge-owned and reference-remapped under /Game/ForgeArt. Game Master's exclusive design is not distributed. Core requires neither MassForge nor any inventory plugin.

The new **WBP_IFJournal** and **WBP_IFJournalEntry** use **BP_WatchpostSource**, an existing Blueprint implementation of IFDataSource. Its Record model and Record action drive the custom entry without game C++ changes. The Return button uses Request Screen Back. This is a practical source/widget replacement example within the current RPG flow; the practice counter is not serialized by the encounter save.

START HERE

Code and Blueprint scheme

The exhaustive folder, file, class and Blueprint graph index is available in the documentation website under Code and Blueprint scheme. The downloadable structure-index.json and engine-schema.json accompany that page. This manual includes the public authoring recipes; the generated implementation listing remains a separate reference.

START HERE

Install and run InterfaceForge

The active **UIForge** redesign targets **Unreal Engine 5.8.3 CL58210709, Win64**. Its runtime plugin/module/mount name remains **InterfaceForge** to preserve Blueprint references. The earlier multi-engine packages and acceptance records are historical; they do not contain or verify this redesign. Read current product status. Physical controller testing remains user-deferred.

Run the interfaces

Open `C:/UEProjects/ashenforge-plugins/UIForge/Project/UIForge.uproject` with UE5.8.3. Build **UIForgeEditor Win64 Development** using the installed Visual Studio C++ tools if prompted. The plugin is inside this project's `Plugins/InterfaceForge`. Work in this one project; do not create additional engine/project copies.

Play `/Game/ForgeExamples/Maps/ForgeGateway`. Choose **Emberfall** for the MMO-style local encounter or **Ironfront** for RTS defense. Each opens a proper main menu with Continue, New Game, Load Game, Settings, Example Library and Exit. Continue uses an existing example save; New Game preserves that save until you explicitly save again. In game, Menu opens Pause with Save/Load/Settings/Return. Return and Exit use confirmation dialogs.

Emberfall uses a third-person camera and game-owned Enhanced Input mappings. Move, select/target with mouse or Tab, activate actions and inspect party, target, objective, map and journal displays. Applied bindings override the default keys. The center-origin cooldown is a visual control independent of the encounter's skill rules.

Ironfront supports unit picking, marquee/additive/type selection, groups, contextual and queued orders, formations, recruitment/cancellation and a virtualized roster. Right-click a destination after selecting units; the game adapter owns movement, authority and costs. These local demonstration rules are separate from InterfaceForge. Follow the current playable guide rather than historical scout-example instructions.

Settings now uses the Designer-authored `WBP_IFSettingsMenu`: Interface, Audio, Controls and Video tabs share one draft and one Apply/Cancel footer. Switching tabs retains edits; Back/Cancel restores previewed preferences and staged keys. **Reset this page** affects only the current category. Applying a binding updates its Enhanced Input action and prompt. The sample controller registers its context; your game must register its own. Follow Blueprint settings composition to build or replace this menu without project C++. Controller switching is implemented and software-tested, with hardware verification still deferred. Touch/platform compatibility is not inferred from desktop tests.

Edit the new saved Widget Blueprints, profiles and themes in `/Game/ForgeExamples/UI`. The example actor's UI/art properties choose classes and data-source adapters. Use the same public features described in Blueprint product layouts. The MMO-style example is local; it does not supply an online MMO backend or inventory mechanics.

Historical baseline examples

The routes below preserve earlier assets and evidence. They are not the entry point or visual acceptance for the current redesign.

Open `/Game/Maps/ForgeOutpost` and Play. **Begin expedition** opens the RTS HUD. **Patrol** completes the road objective and adds supplies; **Train** spends supplies and advances production. **Pause** opens settings, the journal, and return confirmation. These interactive actions belong to the sample adapter, outside the UI plugin.

From the main menu choose **Visit Emberwatch**, or open `/Game/Maps/ForgePilgrimage` in the editor, for the original Emberwatch RPG interface. Accept the keeper's request, **Approach**, then **Kindle** the hearth. **Ward** updates party/status and drives the independently reproduced Sakura center tint over original sigils. It reuses the same menu/settings/dialog flow with another data source. RPG authoring and cooldown controls explain the graphs.

Pause → Field journal → Watchpost survey opens a custom Blueprint screen/entry driven by a Blueprint IFDataSource. **Observe ridge** advances the survey; four observations complete it. The theme choices demonstrate live customization. Customization shows source, entry, theme, and root replacement without plugin source edits. Choose

WBP_ForgeFlowRoot through **Create Root** or the example controller's Root Class to use the alternative Designer-built root.

Edit saved widgets under `/Game/UI/Blueprint` in the Designer and graphs. The editor authoring module maintains development assets; packaged runtime does not load it. Users do not need to run authoring tools to customize the assets. **DA_ForgeInterfaceAssets** cooks the sample's soft UI classes; add your own screens to your project's Asset Manager rules.

World navigation documents the actual menu/provider travel route and **Remove Root** lifecycle before **Open Level**. IF7 runtime and visual review covers both genres, clean settings reloads and12world-travel trails. Broader devices/platforms/configurations remain unverified.

Add the plugin to your project

The current rebuild is a **source installation for UE5.8.3**. The Review/Review2 binary-archive instructions in older evidence describe earlier revisions; do not use those archives as a substitute for this source. The minimal Blueprint-only host check loads the current core without ForgeExamples, MassForge or inventory, but does not certify a newly distributed precompiled plugin archive. Windows C++ build tools are needed to compile the plugin, including Unreal's temporary target for a Blueprint-only project.

1. Copy `Plugins/InterfaceForge` into your project's Plugins folder, excluding generated Binaries, Intermediate, and Saved. Rebuild for your engine.
2. Enable **InterfaceForge**, **CommonUI**, and **EnhancedInput**. CommonInput comes through CommonUI. The runtime has no example, inventory, MassForge, or game-specific controller dependency.
3. Set **Game Viewport Client Class = CommonGameViewportClient**. Select **IFNavigationInputData** in Common Input Settings for default Enter/Escape and generic controller Accept/Back, or retain compatible project input data.
4. Set **Common UI Settings → Common Button Accept Key Handling = Trigger Click**.
5. From your local PlayerController, get **IFLocalPlayerSubsystem**, call **Create Root** with IFRootLayout or your root Blueprint, then **Push Screen** with an IFScreen Blueprint on Menu or HUD. Set **Initial Focus** to an enabled control. Quick-start graphs give the actual nodes and expected result.

Call **Set Theme** to supply project styling. DA_ForgeMineral and DA_ForgeEmber are original sample themes. Menus and timelines, scrolling, and safe areas cover presentation customization.

For remapping, enable **Enhanced Input User Settings**, register a player-mappable context before opening controls, and use its exact Mapping Name in IFActionPrompt. Controls and glyphs cover capture, validation, applied prompts, and persistence limits. Keep gameplay execution in your controller/provider.

For audio, create a project-owned **IFAudioRouting** asset with a SoundMix and distinct UI/game SoundClasses, then **Set Audio Routing**. Settings covers staged editing, Apply/Cancel/defaults, and save failures. Language settings uses Unreal's culture system and localization tools.

Extend and troubleshoot

Use the RTS command, journal/entry, progress, tooltip, notification, loading, and world UI recipes for saved examples with actual node names. The root README and standalone example README give the playable starting routes.

Missing keyboard acceptance: check viewport/input data, enabled controls, and top-screen focus. Missing remapped prompts: check registration and Mapping Name. Call parent **Initialize Screen** before accessing **Screen Data**. The current tabbed menu coordinates preferences and remapping under one editing session; underlying SaveGame, Enhanced Input and engine config writes are separate operations and are not an atomic disk transaction. Historical `WBP_AshenSettings/WBP_AshenControls` examples retain their older separate Apply behavior.

Game Master's visual design is exclusive and is not distributed. The examples use original layouts, art, themes, animations, and audio. Reference captures and deferred inventory work are excluded by DistributionPolicy.json. Inventory mechanics are outside this goal. Optional integration, physical devices, mobile/consoles and split-screen retain the limits in the evidence. The earlier IF1 candidate passed a full UE5.6 Shipping package build. Later standalone revisions have24actual

Shipping primary/reload passes and36menu world-travel trails across the three engines; see Shipping verification for the revision and feature limits. Refreshed candidate installation/runtime and final delivery acceptance remain open.

START HERE

Your first InterfaceForge screen

Start with the step-by-step visual route: exact locations, real Designer/graph images and expected results. This page supplies the detailed contract/reference.

Start with the installation setup: enable InterfaceForge/CommonUI/EnhancedInput, use **CommonGameViewportClient**, select **IFNavigationInputData**, and set **Common Button Accept Key Handling** to **Trigger Click**. The plugin's core does not require an example controller, MassForge or inventory.

Create and display a screen

1. Create a Widget Blueprint named **WBP_MyMenu**, with parent **IFScreen**. In class defaults set **Input Policy = Menu**.
2. Add a Vertical Box and an **IFButton**. Set its label to **Continue**. Keep it enabled and focusable.
3. In **Event On Initialized**, set **Initial Focus** to the button instance. Conditional focus can use CommonUI's desired-focus override.
4. In your local PlayerController's **Event BeginPlay**, use **Get Local Player Subsystem**, choosing **IFLocalPlayerSubsystem**. Call **Create Root**, class **IFRootLayout**, then **Push Screen**, layer **Menu**, class **WBP_MyMenu**. Data can be None for this first screen.
5. Bind the button's **On Button Base Clicked** to your own action. For a first check, **Print String** confirms activation in development. Use **Deactivate Widget** when the screen should close.

Blueprint workflow (text source; interactive diagram in the web guide)

```
flowchart LR
    A[PlayerController: Event BeginPlay] --> B[Get Local Player Subsystem: IFLocalPlayerSubsystem]
    B --> C[Create Root: IFRootLayout]
    C --> D[Push Screen: Menu / WBP_MyMenu / Data None]
```

Expected: the menu sits in a local-player-owned safe-area root. Keyboard navigation focuses the button; Enter activates it with the configured CommonUI acceptance policy. **Go Back** closes the top back-handling screen. A main menu that must remain present should override **On Handle Back Action** and open a confirmation instead; the saved menu recipe demonstrates this.

Add project styling and input presentation

Create an **IFTheme** data asset and call **Set Theme** on the same local-player subsystem. Change Surface/Text/Accent, Body Font, padding, sounds and transition duration. The themed controls update without replacing their trees. A control's **Theme Override** plus **Refresh Appearance** supplies a local variation.

In your screen implement **On Input Presentation Updated** to change hints or layout for Pointer, Keyboard, Gamepad or Touch. The shared detector already observes intent; do not add one detector per button. Input switching and glyphs explain remapping-aware prompts and project controller data. Physical controller and touch hardware remain unverified.

Move from the first button to a game interface

Use **Push Screen** on HUD for your gameplay screen and on Modal for dialogs. Supply an object implementing **IFDataSource** as screen Data. Its **Get UI Model** returns presentation objects; **Request UI Action** delegates validated commands to your game. This keeps rules outside the UI plugin.

The Watchpost survey is a saved custom Blueprint screen, entry and source using this exact contract. Playable interface routes covers the RTS/RPG examples. Settings adds real preview/Apply/Cancel/defaults without inventing a second game-settings framework.

If the button does not activate, check the viewport class, input data, accept-key policy, enabled state and **Initial Focus**. If Screen Data disappears in an override, call parent **Initialize Screen** first.

START HERE

Blueprint workflows (implementation checkpoint)

These recipes name the current reflected APIs. The standalone playable slice uses saved Blueprint main/pause screens, settings, controls, modal, RTS/RPG HUDs, readouts and a themed virtualized journal. See BLUEPRINT_MENUS_HUD.md, BLUEPRINT_RPG.md, BLUEPRINT_SETTINGS.md, BLUEPRINT_CONTROLS.md, BLUEPRINT_DIALOGS.md, BLUEPRINT_JOURNAL.md, BLUEPRINT_RTS_COMMANDS.md and Blueprint tooltips for exact graphs and runtime evidence; INSTALLATION.md explains setup. Runtime evidence includes controls capture/apply/cancel/defaults, prompt updates and editor-game restart persistence. Physical-device and release gates remain open. No inventory system is required. Game Master's bespoke design stays exclusive; these sample trees/assets are original.

Bring up your interface

Enable InterfaceForge, CommonUI and EnhancedInput. Set Project Settings → Engine → General Settings → Game Viewport Client Class to `CommonGameViewportClient`. Set Common Input Settings → Input Data to `IFNavigationInputData` for the quick-start Enter/Escape and generic controller accept/back bindings, or choose your own `CommonUIInputData` Blueprint. Do not overwrite an existing project's input data without checking its actions.

For focused-button Enter/Space activation, set Common UI Settings → **Common Button Accept Key Handling** to **Trigger Click**. Existing projects may still use Ignore; in that mode each button needs a triggering input action instead. The standalone example chooses Trigger Click. Keep CommonUI routing and synthetic controller accept intact. See Epic's accept-key setting and input technical guide.

Create a Widget Blueprint with parent `IFScreen`. Add your content and focusable CommonUI buttons (`IFButton` supplies a labeled default). In class defaults choose `Input Policy`: Menu, Game and UI, or Gameplay. Set `Initial Focus` to your first enabled control when constructing the widget. Use CommonUI's desired-focus Blueprint override for conditional targets. HUDs that should remain visible disable the inherited `Is Back Handler` default.

From the owning local PlayerController's BeginPlay:

Blueprint workflow (text source; interactive diagram in the web guide)

```
flowchart LR
    B[Event BeginPlay] --> S[Get Local Player Subsystem: IFLocalPlayerSubsystem]
    S --> R[Create Root: IFRootLayout]
    R --> P[Push Screen: Menu, your WBP, optional Data]
```

Expected: that local player owns a safe-area layout with HUD/Menu/Modal CommonUI stacks. `Push Screen` calls `Initialize Screen` with your Data before activation. In an override, call the parent to retain `Screen Data`. `Deactivate Widget` returns through a stack; `Go Back` closes the top Modal/Menu. Engine Back handling supports the configured input data. `Register Layer` lets a custom `IFRootLayout` Blueprint supply its own CommonUI stacks.

Report a completed action

For brief action results, use **Get Local Player Subsystem: IFNotificationSubsystem** → **Post Notification**. Pass a Scope and use **Dismiss Scope** as that flow ends. Replace the passive body through **Default Notification Entry Class** on your root Blueprint; Blueprint notifications documents coalescing, lifetime, placement and live class/host refresh.

Respond to input once

In any `IFScreen` implement `Event On Input Presentation Updated`. Switch on `EIFInputPresentation`: Pointer, Keyboard, Gamepad, Touch. Change hint visibility, spacing or other presentation. The same event reaches HUD screens behind a modal, but only the top screen owns cursor/focus policy.

For a plain UserWidget use `Get Local Player Subsystem: IFInputPresentationSubsystem`, `Get Input Presentation` for the initial value, and bind `On Input Presentation Changed` in Construct. Unbind in Destruct. No detector is needed per widget. Focus restoration and cursor visibility still belong to the containing screen. The native service does not consume actions. It ignores key repeats, net mouse movement below six pixels within 0.2 seconds, and analog axes below 0.4 until a deliberate threshold crossing. Thresholds are exposed on the subsystem. `Ignore Next Pointer Move` precedes your own cursor warp. Engine simulated controller accept is recognized.

Expected: fresh keyboard input updates keyboard presentation; meaningful mouse travel/click/wheel updates pointer presentation; a controller press updates CommonInput and navigation presentation immediately. External CommonInput filters/locks and platform support still govern the effective family. Physical controller/touch evidence is pending; the API alone is not a compatibility claim.

Analog Intent Threshold defaults to0.4 and **Analog Release Threshold** to0.2. A held axis must return below the release threshold before another deliberate crossing can switch presentation, so constant drift/held-stick noise cannot repeatedly reclaim the interface. This extends CommonInput intent detection; it does not replace CommonUI navigation deadzones or consume the axis. Tune these properties on the owning local player's subsystem, not per widget. The platform device mapper's disconnect handler restores that player's last non-gamepad presentation only when its last active gamepad disconnects. Its actual physical behavior remains unverified. The user explicitly deferred hardware checks on2026-10-12 because no controller is available.

The root normally enables observation automatically. If your own flow temporarily calls **Set Observation Enabled(false)** while retaining screens, later call **Set Observation Enabled(true)**. Re-enabling publishes a changed effective engine family to those existing widgets and updates the top screen's cursor/focus policy. If the family is unchanged, Pointer versus Keyboard is preserved. Input that occurred while disabled cannot be classified retroactively as a pointer/keyboard intent; a newly observed MouseAndKeyboard family uses Pointer until fresh keyboard input arrives. External CommonInput filters/locks remain respected.

Software edge tests operate the actual saved custom screen behind a dialog: subthreshold analog drift, deliberate crossing, held-axis suppression/release, unrelated/active device-mapper disconnect notifications, key repeats, touch start/end, keyboard restoration, observer resume and a caller-owned cursor warp. Device notifications are injected into the engine delegate without changing hardware connection state. Synthetic touch is a Win64 event-routing test, not mobile/touchscreen certification. Exact runtime evidence distinguishes these checks from the explicitly deferred physical tests.

IFActionPrompt resolves the current applied Enhanced Input key and asks CommonInput's configured platform/controller data for its brush. When a brush is unavailable, it shows the key's readable name rather than an empty prompt. Supply your project's CommonInput controller data to obtain device-specific art; the current sample opts into an original four-face-position generic set. Neither synthetic family switching nor a text fallback establishes glyph-art or device certification. See controls/remap recipe.

Show a cancellable dialog

`Construct Object from Class` → `IFDialogData` (use the owning screen as Outer). Set Title, Message, Confirm Label and Cancel Label; bind `On Completed`. Call `Push Screen` with Modal, `IFDialog`, and the data object. Switch on result Confirmed/Cancelled.

Expected: the HUD remains underneath a dark scrim. Cancel is initially focused. Escape/right-face Back returns Cancelled exactly once. Returning restores the opener's CommonUI focus. The dialog contains no gameplay operation; your callback decides what confirmation means.

Reusable entries

For a quick entry create a Widget Blueprint with parent `IFEntryWidget`. Implement `On Entry Initialized` and cast the data to your own model (or `IFPresentationData`). Store references to labels/images and set their values in `On Entry Refreshed`. Implement `On Entry Selection Changed` for the selected state and `On Entry Released` to cancel custom asynchronous work and clear external subscriptions/tooltips. The base handles ListView/TileView/TreeView object lifecycle and binds `IFPresentationData.OnDataChanged`.

For a completely custom UserWidget implement `User Object List Entry` and `IFEntry`. Route `On List Item Object Set` → `Initialize Entry`, `On Item Selection Changed` → `Set Entry Selected`, `On Entry Released` → `Release Entry`. You own binding teardown in this path. Never retain a previous model when the entry recycles. Modify a generic model's fields, then call `Notify Data Changed` once after the group of edits. Do not rebuild the whole list for one row.

When using **IFEntryWidget**, use its **InterfaceForge | Entry** hook events; the base already routes the engine list lifecycle. The similarly named **UserListEntry** → **On Entry Released** runs after the base clears its context. Use the InterfaceForge hook for cleanup that needs **Entry Data**, and do not wire both paths. Initialization with None is valid for a static entry; the current lifecycle fix releases that initialized context once even without a model. Journal contracts explain the hooks and current verification scope.

Progress independent of gameplay

For a complete different saved screen/entry with a Blueprint IFDataSource, playable actions, theme controls and a Designer root, follow Blueprint customization. The survey uses the same public contracts rather than a native gameplay controller cast. The outline variants below are additional procedural tools. Sakura's requested center-origin tint has its own independently authored IFCenterTint/M_IFCenterTint implementation and runtime evidence; see CENTER_TINT.md. BLUEPRINT_RPG.md documents the editable action-icon composition in the playable Emberwatch example. Physical-input and final polish gates remain open. Add `Forge Progress` in a Widget Blueprint. Choose Linear, Radial, Square Perimeter, Segmented or Charges. `Set Fraction` clamps a finite value to 0-1 and stops a previous cooldown. `Start Cooldown` accepts seconds and uses an absolute game-time endpoint. A zero duration completes immediately. `Get Remaining Seconds` and `Get Displayed Fraction` supply countdown labels. Game time pauses with the world. Reverse and Start Offset control fill direction/origin; square zero starts top-left and radial zero starts top. Segments controls both segmented and charge presentations. These native procedural styles do not require a backend or material. Material variants and visual corner refinements remain pending.

START HERE

Original miniature scenes and replaceable UI

Historical examples below. The active UE5.8.3 product examples are **Emberfall** and **Ironfront**, opened through **ForgeGateway**. See the current gallery, public authoring workflow and current status.

ForgeOutpost and ForgePilgrimage use independently authored miniature scenes to give their interfaces a recognizable playable context. Their presentation uses Unreal's BasicShapes meshes and the original **M_OutpostPalette** material; no Game Master/Sakura artwork, meshes, textures or animations are included. Game Master's design remains exclusive. The specifically requested Sakura cooldown is an independently reproduced UI behavior, with separate provenance in CENTER_TINT.md.

The scene art belongs to the example, not the InterfaceForge runtime. **ForgeSceneArt** adds decorative, noncolliding mesh components to the existing sample characters and configures a stable camera/fill light. Stonework, plinths and landmarks give the scenes a deliberate composition. The decoration moves with the existing actors; recruited scouts receive the same presentation. It adds no equipment, item, combat or authority system. Existing selection/command/quest/interaction/cooldown data remains the adapter's responsibility.

Customize the interface first

Open the saved **WBP_AshenMainMenu**, **WBP_AshenRTSHUD**, **WBP_EmberRPGHUD** and their entries in Designer. They use the same public InterfaceForge screens, theme widgets, entry/model lifecycle, loading, prompts, settings, tooltips and notifications as your project. Change the theme/widget classes/layout/animations/data source there; scene art does not own those decisions. See BLUEPRINT_RECIPES.md, BLUEPRINT_CUSTOMIZATION.md and BLUEPRINT_MENUS_HUD.md.

To use another environment, keep these UI classes and supply a source implementing **IFDataSource**. Your source returns stable models through **Get UI Model**, updates them through **Notify Data Changed**, and validates requests from **Request UI Action**. You can use your own Blueprint source instead of the sample's local actor adapters. No plugin-source changes or particular camera/game controller are required by the UI toolkit.

Blueprint workflow (text source; interactive diagram in the web guide)

```
flowchart LR
    G[Your gameplay or example adapter] --> D[IFDataSource and stable presentation models]
    D --> W[Public Blueprint screens and entries]
    G --> A[Your world art and camera]
    T[IFTheme] --> W
```

Example-only art controls

M_OutpostPalette exposes **Tint**. Use a Dynamic Material Instance and **Set Vector Parameter Value** to recolor your own geometry. The provided character torso components share their body material so the existing RTS selection tint still reads consistently; decorative cloth/boots retain their own colors. These are ordinary engine materials/components, not an InterfaceForge avatar or equipment feature.

The sample adapter exposes its **Camera** reference. Normal Unreal Camera Component/post-process controls can replace the example's exposure, orthographic view and presentation. The current art pass uses fixed manual exposure, disabled motion blur, a subtle vignette, one main directional light and a shadowless movable point fill. Epic's point-light documentation explains mobility, attenuation and falloff controls. Those choices are sample presentation, not a project-wide graphics-settings override.

The native example art helper is optional source code for these two scenes; it is not essential hidden UI behavior. Build your own level/camera/actors in Blueprint and retain the documented UI source contract. Do not rerun internal example-authoring tools over customized assets: duplicate the example widgets into your project and author there

normally.

Runtime, packaged/viewport/input scope, visual failures and remaining gates are in Evidence/SCENE_ART_RUNTIME.md. The scenes are deliberately small interactive UI backends, not complete RTS/RPG games or multiplayer demos.

START HERE

Replace a screen, entry, source and root in Blueprint

The standalone sample has an independently authored **Watchpost survey**. Start an expedition, Pause → Field journal → Watchpost survey. **Observe ridge** records one observation, updates the count and four segments, and becomes unavailable after four observations. Back returns to the journal; closing the journal returns to Pause. This small local interaction source belongs to the example project. Game Master's exclusive design and inventory systems are absent.

Open `WBP_WatchpostSurvey`, `WBP_WatchpostEntry`, `BP_WatchpostSource` and `WBP_ForgeFlowRoot` under `/Game/UI/Blueprint`. They are saved, editable Blueprints. Their runtime graphs call public InterfaceForge and engine nodes; the editor authoring utility is not a packaged dependency. Runtime evidence distinguishes saved Blueprint interaction from synthetic device events and remaining acceptance gates.

A different screen and entry

`WBP_WatchpostSurvey` derives from **IFScreen**. Its Designer contains a full-screen scrim, a **Scale Box** set to Scale To Fit / Down Only, a bounded **Size Box**, an **InterfaceForge Surface**, heading, custom entry and Return button. The entry is `WBP_WatchpostEntry`, parent **IFEntryWidget**, with themed title/detail/count text, **Forge Progress** set to Segmented /4, and a reusable custom **IFButton**.

The screen uses these graphs:

1. **Event Initialize Screen** → **Parent: Initialize Screen**, forwarding Data → **Get UIModel** with Model Name `Record` → **Initialize Entry** on SurveyRecord. The returned object comes from the supplied source, before screen activation.
2. Set **Initial Focus** to SurveyRecord's RecordChoice button. This nested button follows the same CommonUI focus/navigation contracts as the menu.
3. CloseSurvey **On Button Base Clicked** → **Request Back**. **Event On Deactivated** → **Release Entry** on SurveyRecord. Release is idempotent and removes its model binding even while CommonUI retains the screen in a pool.
4. **Event On Input Presentation Updated** → set the example's LastInputPresentation variable. Replace that assignment with your own layout/hint rules; keep focus and cursor policy in the containing screen. The service also updates the screen behind a modal.

The entry's **Event On Entry Refreshed** casts Entry Data to **IFPresentationData**, reads Title/Detail/Quantity/Fraction/Enabled, sets its text, converts Quantity through **To Text (Integer)**, calls **Set Fraction**, and calls **Set Action Availability**. Count and `/ 4` text disable Auto Wrap Text; prose wraps. **Event On Entry Released** clears labels and progress. The convenience base owns Notify Data Changed subscriptions; it does not query a gameplay population every frame. Its editable **Progress Presentation** property and **Apply Progress Presentation** event now select all five styles through public Configure Progress and normal slot alignment; the progress recipe explains Designer preview and live changes using the same source and entry.

RecordChoice **On Button Base Clicked** → engine **Find Parent Widget Of Type** (`IFScreen`) → **Cast to IFScreen** → **Request UIAction** (`Record`, Context = Entry Data). Find Parent Widget Of Type has execution pins: connect the click through that node before using its returned object. Its result cannot be read from an unexecuted node.

Blueprint workflow (text source; interactive diagram in the web guide)

```
flowchart LR
    J[Journal: Watchpost survey click] --> C[Construct Object from Class: BP_WatchpostSource]
    C --> P[Push Screen: WBP_WatchpostSurvey, Data = source]
    P --> I[Initialize Screen / Get UIModel: Record]
    I --> E[Initialize Entry: WBP_WatchpostEntry]
    B[Observe ridge] --> A[Request UIAction: Record / Entry Data]
    A --> V[Blueprint source validates and edits its model]
    V --> N[Notify Data Changed]
```

```
    N --> R[On Entry Refreshed: count / text / segments / availability]
```

A Blueprint-only data source

`BP_WatchpostSource` derives from **IFPresentationData** and implements **IFDataSource** in Class Settings. For this one-record example the stable model is the source object itself. **Get UIModel** returns Self only for `Record`, and None for unknown keys. It allocates no new model per read.

Request UIAction requires Action Name `Record`, Context equal to Self, and Quantity below4. The accepted branch increments Quantity, sets Fraction to Quantity×0.25, updates Enabled/completion prose, calls **Notify Data Changed**, then returns true. The other branch returns false without changing data. These are local demonstration rules, not multiplayer authority or a gameplay system supplied by the UI plugin.

The journal's SurveyChoice graph uses **Construct Object from Class**, with the journal as Outer, followed by **Get Root Layout** → **Push Screen** on Menu. Screen Class is WBP_WatchpostSurvey and Data is the new source. Each new opening gets a new survey session. To use your game, replace that class/data pin with your own object implementing IFDataSource and return your stable observed model. No plugin source changes are required.

Theme replacement and one-widget overrides

The journal's Mineral/Ember buttons call **Set Theme** on **IFLocalPlayerSubsystem** from **Get Owning Player**. The same screen, pooled entries, buttons and underlying HUD receive token changes. These choices are session previews; the example does not persist a theme preference.

Create or duplicate an **IFTheme** data asset to edit Surface/Text/Accent/Muted Text/Outline, Body Font, button padding/corners, icon roles, feedback sounds and Transition Seconds. Assign it through Set Theme. Use project-owned or distributable assets. Themes change presentation tokens; arbitrary layout or Blueprint animation changes belong in the widget/root.

For one text/button/surface/progress widget set **Theme Override** in its Blueprint Designer, or set that property in a graph and call **Refresh Appearance**. Set None to follow the player's theme again. Runtime verification changes only the survey title's override while its screen retains the player theme; this fixture setup is distinct from the actual journal's saved Blueprint theme-button graph. Changing an override does not clone the tree.

A Designer-built root and engine transitions

`WBP_ForgeFlowRoot` derives from **IFRootLayout**. Its Designer uses **Safe Zone** → **Overlay**, containing HUD and Menu **Common Activatable Widget Stacks**, a **IFNotificationLayer**, then the Modal stack. Menu's engine Transition Type is Horizontal; HUD and Modal use Fade Only. Engine Transition Curve remains Cubic In Out. This is a configurable engine preset, not a separately implemented animation system. See Epic's CommonUI transition types.

Event On Initialized calls **Register Layer** for HUD/Menu/Modal with the Designer stacks, then root **Register Notification Layer** with Results. Its class defaults choose the saved confirmation/loading classes. The notification layer chooses WBP_ForgeNotification as its body. Root registration handles lifetime: removing the root clears messages and unregisters the presenter. None explicitly unregisters it; a removed root rejects re-registration. Use a presenter owned by the same local player.

Theme Transition Seconds and reduced-motion preference still control all registered stack durations. The engine Designer controls transition type and curve; the root keeps CommonUI's activation, pooling, focus restoration, input guard and transition events. The survey now also includes an editable UMG SurveyArrive timeline and actual activation/preference/deactivation graphs. Blueprint animations explains editing it and restoring its visible baseline on interruption. Arbitrary timelines and custom exit choreography remain separate gates.

From your local PlayerController call **Create Root** with WBP_ForgeFlowRoot instead of the default root. In the example you can create a Blueprint child of ForgeExamplePlayerController, set its **Root Class** class default, select it in your Blueprint GameMode's Player Controller Class, and assign that GameMode to the map. The current default maps retain their existing roots. The opt-in runtime fixture replaces the root through the same public Create Root call and runs the existing menu →

HUD → pause → journal → survey flow.

Troubleshooting and limits

Missing entry data: call parent Initialize Screen and return the exact requested model key. Stale updates: release/unbind before replacing a model; do not retain an earlier source in a recycled entry. Unusable nested focus: set Initial Focus after supplying context and use a focusable enabled CommonUI button. Empty controls in a package: include your new Blueprint classes through the project's engine Asset Manager/cook rules; the example label lists the owned UI classes. The root must register every stack before pushing screens.

Current evidence is Win64 DX11 at1280×720 and800×600 with150% UI/text. Synthetic gamepad/pointer/keyboard events test shared Blueprint presentation while a modal is open; they do not certify a physical controller, touch device or platform. See the separate animation record for actual timeline playback, interruption and Reduced Motion proof. Arbitrary timelines, tree/tile replacements, long localized strings, safe-area variations, split-screen and full product polish remain separate gates.

MENUS AND NAVIGATION

Blueprint menus and an observable HUD

The standalone example defaults to **WBP_AshenMainMenu**, **WBP_AshenPause**, **WBP_AshenRTSHUD**, **WBP_AshenSettings**, **WBP_AshenControls** and **WBP_ForgeConfirmation**. These are ordinary saved Widget Blueprints using public InterfaceForge APIs. The earlier native menu/HUD classes are retained as development references. Game Master's visual design stays exclusive to Game Master; these sample trees and presentation are independently authored.

Menu flow

The main and pause Designer trees contain InterfaceForge Surface, IFText and WBP_ForgeActionButton widgets. `Event On Initialized → Set Initial Focus` selects their primary button. Each button uses `On Button Base Clicked`.

Operation	Actual public nodes	Expected result
Begin	Request UI Action, Action Name = BeginExpedition	Adapter enters gameplay and pushes its HUD
Settings	Main: Get Root Layout → Push Screen Async, Layer = Menu, soft class = WBP_AshenSettings; Pause: Push Screen	Settings enters the stack; Cancel rolls back its preview and restores the opener
Journal	Get Root Layout → Push Screen, Class = WBP_FieldJournal, Data = Screen Data	Same adapter supplies the journal's observable models
Resume	Deactivate Widget	Pause closes; adapter sees the HUD on top and resumes its small demo
Quit/Return	Construct Object from Class: IFDialogData; set labels; Bind Event to On Completed; Get Root Layout → Push Dialog	Modal asks the player; Confirmed requests the relevant adapter action

The Blueprint **On Handle Back Action** override returns true. Main calls its OpenExitConfirmation custom event; Pause calls Deactivate Widget. This keeps main visible when Back opens a dialog. The Confirmed callback calls `Request UI Action` with QuitGame or ReturnToMainMenu. Cancelled performs no game action. These action names are **example-adapter contracts**, not built-in InterfaceForge commands.

The Outpost backend implements **IFDataSource**, validates the current flow/state and owns the demo actions. InterfaceForge simply forwards requests. Replacing it requires an object implementing `Get UI Model` and `Request UI Action`; the plugin needs no reference to Outpost or the example controller. A missing source returns None/false.

The main screen's **Initialize Screen** graph calls its parent and the editable **Refresh Menu Presentation** event. **Read Menu Text** uses public **Get UI Model** and IFPresentationData Title/Detail, with explicit fallbacks for missing/empty models. Emberwatch supplies identity, Begin/Resume and confirmation wording; Outpost retains its Ashen Reach/expedition defaults through the same classes. Every reinitialization resets missing fields instead of retaining the previous source's text. Actual nodes, optional model names and live refresh.

The current Designer keeps **MenuPublisher** and the bounded **MenuTitleFit** outside **MenuActions**. Main's introduction has its own **MenuReading** Scroll Box. Only the action column scrolls when necessary. This prevents identity or long prose from consuming all navigation space. The current main menu includes Begin, Settings, Visit the other world and Quit. At enlarged settings the bounded action column may scroll; focusing Quit scrolls it fully into view. Keep the existing PrimaryChoice/SettingsChoice/JournalChoice/ExitChoice widgets when reflowing so their saved click graphs and Initial Focus remain connected. Exact dimensions and customization choices.

Blueprint workflow (text source; interactive diagram in the web guide)

```
flowchart TD
    C[Canvas: stretch slot, margin 24] --> F[Scale Box: Down Only]
    F --> B[Size Box: preferred card bounds]
    B --> S[InterfaceForge Surface]
    S --> V[Vertical Box: MenuColumn]
```

```
    V --> P[MenuPublisher: semantic accent]
    V --> T[MenuTitleBounds → Scale Box → MainTitle or PauseTitle]
    V --> B[Main only: MenuDetailBounds → MenuReading → MainDetail]
    V --> A[Fill: MenuActions → MenuChoices → public buttons]
```

Current UE5.6 source and packaged interaction results, visual review, failures and scope are recorded in menu presentation evidence. Arrow navigation is synthetic Slate input; it does not certify physical controllers or touch. Game Master's layouts, style, artwork and animations remain excluded.

Composing HUD elements

WBP_HUDReadout and **WBP_ProductionReadout** derive from IFEntryWidget. They are reusable outside lists too. The HUD Designer places them inside original themed panels, beside custom action buttons and an IFActionPrompt.

Blueprint workflow (text source; interactive diagram in the web guide)

```
flowchart LR
    A[Event Initialize Screen] --> B[Parent: Initialize Screen]
    B --> C[Get UI Model: Resources / Selected / Objective / Queue / Notice]
    C --> D[Initialize Entry on the matching Designer widget]
    D --> E[Entry: On Entry Refreshed]
    E --> F[Cast Entry Data: IFPresentationData]
    F --> G[Set Text: Title and Detail]
    F --> H[Queue: Set Fraction]
    I[Backend edits stable model] --> J[Notify Data Changed]
    J --> E
```

Initialize each entry when screen data changes. The entry base replaces old subscriptions, reads the initial model, listens for Notify Data Changed and unbinds on release. The Blueprint readout handles visuals. It does not search the world or poll the entity population. The queue uses a theme-aware Forge Progress in Linear mode; **Use Theme Colors** chooses Accent/Outline. Custom colors remain available by disabling that option.

After setting Detail, `Text Is Empty → Branch → Set Visibility` collapses an empty detail and restores Self Hit Test Invisible for nonempty text. This avoids empty caption spacing without losing later updates. `On Entry Released` clears both text fields, collapses Detail and resets queue fraction to zero before another model reuses the entry. These are ordinary Blueprint nodes, not example-only runtime helpers.

HUD command entries forward SelectNext, ToggleGroup, Patrol, Rally and Recruit to the same adapter. They consume separate stable command models for labels, availability and reasons; Blueprint RTS commands documents their actual graphs, group selection and nested navigation. Pause forwards PauseExpedition. The example backend responds with changed status/resources/queue/objective values. The IFActionPrompt resolves IFExampleInteract from Enhanced Input, including applied remaps. Game and UI input policy preserves gameplay routing when no menu/modal is active.

To customize: edit either readout's Designer tree/On Entry Refreshed graph; change panel anchors/classes in WBP_AshenRTSHUD; replace the command mapping, data object, theme or action button class. Keep model objects stable and broadcast after edits. Essential presentation stays in the Blueprint assets/public plugin features; scene/demo behavior stays in the example project.

The current HUD's cards share viewport-relative header/bottom docks and Fill-weighted columns. This prevents the previous fixed-width cards from overlapping at enlarged scale. Standard Scroll Boxes keep longer contexts and orders reachable; Instant Scroll and centered navigation reveal the focused command. Docs/RESPONSIVE_LAYOUTS.md gives the exact Designer arrangement. No action/data graphs changed during this layout upgrade.

Verification boundary

The saved custom button/entry workflow also supports contextual descriptions through Blueprint tooltips. WBP_OutpostCommand and WBP_EmberAction bind their existing models with Set Tooltip Data after availability and clear it on release. WBP_ForgeTooltip provides theme-aware content through public entry events. Evidence/TOOLTIP_RUNTIME.md records source/cooked normal/enlarged, live updates, synthetic switching, focus/modal/world-pause checks and the latest cooked RTS regression. Physical devices and complete UI polish remain open.

Main and Pause UE5.6.1 DX11 live automated mouse/keyboard checks passed: main Back prompt/cancel, Begin through adapter, Pause Back/resume, return confirmation binding and settings push/Back/focus restoration. The first authoring run failed on an unallocated Back return pin; creation now uses the inherited function signature and the rerun compiled with 0 errors/warnings. The Blueprint HUD populated all five models, dispatched patrol with visible movement/objective/+40 supplies, and spent 60 supplies on training with advancing progress and scout count increasing from six to seven. Runtime wrapping/empty-caption gaps were repaired through Blueprint and visually retested; resource values now fit one line and captions collapse. Keyboard Pause/resume restored the HUD's Patrol focus. Physical controller/touch, performance, additional resolutions, replacing the backend entirely in Blueprint and packaging this newer revision remain open.

MENUS AND NAVIGATION

Menu roles and editable arrival motion

The original Outpost and Emberwatch menus use the same public **IFButton** and **IFScreen** features. Their trees, colors and animation are independently authored; Game Master's visual design is not included.

Give actions a clear hierarchy

Select an **IFButton** in the Widget Designer and set **InterfaceForge** → **Button** → **Button Role**, or call **Set Button Role** on an existing button.

Role	Default presentation	Sample use
Standard	Theme surface, normal outline; accent on hover/navigation focus	Settings and Field journal
Primary	Accent-tinted surface and accent outline, strengthened on emphasis	Begin and Resume
Quiet	Transparent background/outline at rest; normal accent emphasis on focus/hover	Main-menu Quit
Destructive	Danger-colored label and Danger emphasis	Return to main menu, after confirmation

Roles style an action; they do not execute it, change its validation or add confirmation. Bind **On Button Clicked** to your public action/dialog graph as usual. **Set Availability** and the widget's Enabled state retain disabled presentation. Standard is the default, preserving existing buttons. Theme **Accent**, **Danger**, **Surface**, **Outline**, **Text**, **Muted Text**, padding and corner radius remain the shared defaults. A button's **Theme Override** can replace them locally.

For a custom button tree, implement **On Appearance Updated** to paint your own focus/hover/availability presentation. Read **Button Role** there if your design distinguishes these actions. The native role treatment styles the existing named **FocusBorder** and **LabelWidget**; it does not automatically recolor arbitrary child artwork. Always preserve a visible focus cue for Quiet actions. Dynamic label/model updates do not replace the button or its focus target.

Content-fit the pause card

In **WBP_AshenPause**, **MenuSurface** sits inside a normal **Size Box**. Height Override is cleared; **Max Desired Height = 420**. The card uses its desired content height at the normal scale, while the existing responsive parent and bounded action region handle larger type. This releases unused vertical space without shrinking text or removing any of the four actions. This is an editable example layout, not a required plugin container or a guarantee that arbitrary translated text will fit.

Author the arrival in Blueprint

Open either **WBP_AshenMainMenu** or **WBP_AshenPause**, select **MenuArrive** in the Animations panel. Its ordinary UMG timeline animates **MenuSurface** → **Render Opacity** from **0.82 at 0 seconds** to **1 at 0.20 seconds**. It changes readability gently without delaying input, async loading or Back. The background/scrim remains separate.

The saved graphs use these actual nodes:

Blueprint workflow (text source; interactive diagram in the web guide)

```
flowchart TD
    A[Event On Activated] --> S[Sequence: existing activation behavior]
    S --> B[Stop Animation: MenuArrive]
    B --> O[Set Render Opacity: MenuSurface = 1]
    O --> B[Branch: ReduceMenuMotion]
    B -->|False| P[Play Animation: MenuArrive, one loop, Restore State true]
    D[Event On Deactivated] --> C[Stop Animation → Set Render Opacity = 1]
    U[Event On Preferences Updated] --> F[Break IFUIPreferences → Set ReduceMenuMotion]
    F --> T[Branch: Reduced Motion]
```

```
    T -->|True| C
```

Keep existing activation/deactivation behavior in the Sequence when adding decorative motion. Stop/reset the owned timeline on deactivation and when Reduced Motion becomes true. Disabling Reduced Motion on an already open menu does not replay its arrival. **On Preferences Updated** supplies the current preference on construction and live changes; no per-widget settings polling is needed.

Duplicate the screen, edit its timeline or replace MenuSurface, then supply the class to **Push Screen** or **Push Screen Async**. Use normal CommonUI stack pushes/removals for navigation. Directly deactivating the active stack widget removes that screen; it is not an animation restart operation. See custom animation recipe and menu data recipe for data and lifecycle details.

Expected: the menu arrives fully readable, Begin/Resume is distinct, Quiet Quit still reveals keyboard/controller focus, destructive Return remains guarded by confirmation, and changing Reduced Motion during arrival restores opacity to 1. Runtime scope and remaining device/presentation gates are recorded in menu hierarchy evidence.

MENUS AND NAVIGATION

One menu Blueprint, different game flows

WBP_AshenMainMenu and WBP_AshenPause are shared by the original Outpost and Emberwatch examples. Their labels and confirmation copy consume optional **IFDataSource** models through ordinary Blueprint graphs. There is no native game-specific menu subclass or additional core dependency. Game Master's design remains exclusive; these trees and assets are independently authored.

Supply your game's wording

1. On your adapter Blueprint, implement **IFDataSource** → **Get UI Model**.
2. Use **Construct Object from Class: IFPresentationData** for a named model, set its **Title** and/or **Detail**, and retain it in your adapter. Return that object for the agreed Model Name.
3. Pass the adapter as **Data** to **Push Screen** or **Push Screen Async**. The screen calls parent **Initialize Screen**, then its editable **Refresh Menu Presentation** custom event.

For the shared sample menus, these optional names are agreed with their adapters:

Model Name	Field	Consumer	Emberwatch example
FrontEnd	Title / Detail	MainTitle / MainDetail	EMBER + line break + WATCH / introduction
FrontEndBegin	Title	Main PrimaryChoice label	Begin journey
FrontEndResume	Title	Pause PrimaryChoice label	Resume journey
PauseIdentity	Title / Detail	Pause publisher / heading	JOURNEY AT REST / Paused
ReturnPrompt	Title / Detail	Return dialog title / message	End this journey? / current journey will be reset
ReturnAction	Title	Return dialog confirmation label	End journey
QuitPrompt	Title / Detail	Main quit dialog title / message	Leave Emberwatch? / default close-game message
QuitAction	Title	Main quit confirmation label	Optional; default Quit

These names belong to the sample's editable Blueprint contract, not a required InterfaceForge schema. Outpost supplies no optional menu models and uses the saved Ashen Reach/expedition fallbacks. You can choose different names and your own UObject models by editing the helper graph. Gameplay action names remain a separate adapter contract: changing “Begin journey” does not rename or bypass BeginExpedition validation.

Actual saved helper graph

Open either menu's **Read Menu Text** function, category **Menu Presentation**. Inputs are **Model Name**, **Fallback Text**, **Read Detail**; output is **Return Value**. It uses public **Get UI Model**, **Cast To IFPresentationData**, **Select Text**, **Text Is Empty**, **Branch** and normal return nodes.

Blueprint workflow (text source; interactive diagram in the web guide)

```
flowchart TD
    F[Read Menu Text: Model Name, Fallback Text, Read Detail] --> M[Get UI Model → Cast To IFPresentationData]
    M -->|Cast Failed| F[Return Fallback Text]
    M -->|Success| S[Select Text: Detail when Read Detail; otherwise Title]
    S --> B[Text Is Empty → Branch]
```

```
    B -->|True| E
    B -->|False| R[Return selected model field]
```

The main **Refresh Menu Presentation** event calls this helper for FrontEnd Title/Detail and FrontEndBegin Title, then **Set Text** or **Set Label** on the existing named widgets. Pause does the same for PauseIdentity and FrontEndResume. Every initialization resolves every field, including fallbacks, so a previously reused Emberwatch menu cannot leave journey text behind when its new source is missing or unrelated.

Fallback Text is an ordinary editable input value on each helper call. Edit those pins when changing defaults; update the corresponding Designer labels for an accurate editor preview. They are saved values, not automatically read from the live widget's previous label. Using a previous runtime label as a fallback would reintroduce stale context. You can replace these literals with your own Blueprint variables or localization data.

Menus do not subscribe automatically to every optional model. These are normally flow identity/configuration values. After an intentional live edit, call **Refresh Menu Presentation** on the existing screen. It rereads the models without replacing the screen, moving focus or restarting transitions. For continuously changing gameplay displays, use the documented observable **IFEntry** workflow instead of polling menu fields.

Confirmation text and behavior

Open Exit Confirmation constructs a fresh **IFDialogData**. Before each **Set Title**, **Set Message** and **Set Confirm Label**, it calls **Read Menu Text** for QuitPrompt/QuitAction or ReturnPrompt/ReturnAction. Missing models or empty fields keep the saved fallback. **Bind Event to On Completed → Push Dialog** remains unchanged. Confirmed requests QuitGame or ReturnToMainMenu; Cancelled changes no gameplay state. The helper reads current copy when opening, rather than retaining a previous dialog's data or callbacks.

Expected: Emberwatch uses journey text through the same classes that Outpost uses for expedition text. Settings, journal, Return to main menu, initial/restored focus and semantic theme/input behavior retain their existing contracts. Only the adapter decides what an accepted action does.

Verification scope

Menu-binding runtime record records exact versions, failures, default/enlarged source/package flows, same-instance source reset, missing/unrelated/empty source handling and a live label refresh retaining Settings focus. Synthetic Slate input and programmatic source changes are explicit. Physical-device tests, arbitrary translations and full release polish remain open.

MENUS AND NAVIGATION

Menu action cues

Use **CommonActionWidget** for CommonUI navigation actions and **IFActionPrompt** for registered Enhanced Input gameplay mappings. Both use CommonInput's configured controller data; they serve different action sources.

Build the menu band in Blueprint

1. In your **IFScreen** Widget Blueprint, add a Horizontal Box beside your menu identity. Add a **CommonActionWidget**, an **IFText** key fallback and an **IFText** action label for each action. Keep a fixed gap between the two action groups, outside the glyph widgets: a collapsed glyph must not remove the gap.
2. On **Construct**, call **Get Default Accept Action** → **Set Input Action** on the first CommonActionWidget. Use **Get Default Back Action** for the second. These nodes read the project's configured Common Input Settings **Input Data**, including your replacement class.
3. Implement a custom event, such as **Refresh Menu Navigation**. Call it after initialization and from **On Input Presentation Updated**. **Get Input Presentation** on the owning player's **IFInputPresentationSubsystem** gives Pointer, Keyboard, Gamepad or Touch. Show the band for Keyboard/Gamepad; collapse it for Pointer/Touch. The sample uses this presentation policy; your Blueprint can choose another.
4. For each icon, **Get Display Text** → **Set Text** on the action label. **Get Navigation Action Key** takes **Self** and the same action row → **Get Key Display Name** → **Set Text** on the fallback. **Get Icon** → **Get Brush Resource** → **Is Valid** determines whether to collapse the fallback or show it. CommonActionWidget handles its icon updates.

Blueprint workflow (text source; interactive diagram in the web guide)

```
flowchart TD
    C[Construct] --> R[Get Default Accept / Back Action]
    R --> S[CommonActionWidget: Set Input Action]
    S --> U[Refresh Menu Navigation]
    E[On Input Presentation Updated] --> U
    U --> V[Get Input Presentation → band visibility]
    U --> T[Get Display Text → action label]
    U --> K[Get Navigation Action Key → Get Key Display Name → fallback]
    U --> I[Get Icon → Get Brush Resource → Is Valid → fallback visibility]
```

Expected: a controller button changes the band to configured icons; meaningful mouse input hides it; keyboard intent restores actual row key names. Returning from a dialog or another screen retains CommonUI focus/back behavior. The getters resolve presentation only: they do not bind, execute or remap actions. CommonUI owns Accept/Back routing. Enhanced Input controls edits affect registered gameplay rows, not these separate CommonUI defaults.

Editable examples and limits

WBP_AshenMainMenu and **WBP_AshenPause** contain **MenuIdentityBand**, **MenuNavigationCues**, **NavigationAcceptIcon/Key/Label** and **NavigationBackIcon/Key/Label**. Their saved Blueprint graphs use the nodes above. Replace the row, layout, fonts or icon class in your project; no plugin-source edits are required. The band does not add focusable controls or replace the existing screen stack.

The original generic controller data supplies four face-position glyphs. The example has no keyboard/brand glyph pack and uses readable key fallback. Configure additional controller data for your target hardware. Large localized labels, arbitrary replacement key names, other platforms and physical controller/disconnection tests remain separate verification gates. The user has explicitly deferred physical-device checks.

MENUS AND NAVIGATION

Blueprint navigation hint hosts

An optional host gives keyboard/controller descriptions a reserved place in your interface. Pointer hover keeps the anchor's popup. The same source object drives both presentations. Use this when a floating focus tooltip would cover status, commands or result messages.

Create your own body and host

1. Create a Widget Blueprint with parent **IFEntryWidget**. Add your presentation widgets. The sample **WBP_ForgeContextHint** contains one **InterfaceForge Text**, **HintDetail**, Base Size11, Body role and automatic wrapping.
2. Its **On Entry Refreshed → Cast Entry Data to IFPresentationData → Get Detail → Set Text (HintDetail)** graph displays the action description. **On Entry Released → Set Text (empty)** clears it. The source button already shows its title and unavailable reason. You can add title, icon, reason or another data type to your own body using the same entry contract.
3. In your **IFScreen** Blueprint, add an empty **InterfaceForge Surface**, Border or other **Content Widget**. Put it in a bounded Overlay, with horizontal Fill and vertical Center. This container must belong to that screen and have no Designer content; its content is managed while a hint is visible.
4. **Event Construct → Register Navigation Hint Host**, Host = your named container, Entry Class = your body Blueprint, **Yield To Notifications = true**. Check the Boolean return when debugging. Your owning player's IFRootLayout must be created and ticking as in the quick start.
5. On your IFButton, use its named **TooltipAnchor** and **Set Tooltip Data** as before. Leave the anchor's **Use Navigation Hint Host** enabled. Without a registered host, it uses the existing popup behavior.

Expected result: after Tooltip Delay, keyboard/controller focus shows the host body. Meaningful pointer input switches to the normal popup body when the pointer is over the control; keyboard navigation restores the host. Both use the same Tooltip Data and preserve source focus. This uses the centralized input-presentation service, not a detector in every widget.

Blueprint workflow (text source; interactive diagram in the web guide)

```
flowchart LR
  A[Set Tooltip Data] --> B[Local player request and delay]
  B --> C{Input presentation}
  C -->|Keyboard / Gamepad| D[Registered screen host and chosen IFEntry]
  C -->|Pointer / explicit Touch gesture| E[Engine Menu Anchor popup and chosen IFEntry]
  E[Notify Data Changed] --> D
  F --> E
```

Get Tooltip Entry returns either body. **Is Tooltip Shown** includes both presentations; **Is Tooltip Hosted** distinguishes a docked body. The inherited **Is Open** reports only the engine popup. Update custom visibility checks accordingly.

Reading priority and lifetime

With Yield To Notifications enabled, a navigation description waits while this player has queued results and a registered notification presenter. A visible hosted description is released/collapsed, and returns after result reading and the normal tooltip delay. Pointer popups remain deliberate hover information. Disable the option if your layout reserves separate places for both. The policy does not delay gameplay or move CommonUI focus.

Only the current top screen's registered host is eligible. Modals, transitions, root removal and explicit **Clear Tooltips** close current/pending content. **Unregister Navigation Hint Host** removes the registration and releases its hosted body; pending requests may fall back to the normal popup. IFScreen unregisters its own host during Native Destruct. Registering an invalid/abstract/non-entry class, another screen's container or externally occupied content returns false and preserves the existing host. Changing a valid body class releases the previous entry subscription. Your own entry should make

cleanup idempotent.

The host uses ordinary UMG layout and the owning player's existing DPI/UI scale. There is no extra width wrapper around a hosted body. Give it enough room for your largest text, translations and safe area. It is passive and cannot take focus or scrolling; long documents belong in a screen or panel.

The saved examples

Both HUDs use **NavigationHintFit (Overlay) → NavigationHintHost (InterfaceForge Surface)**, Fill/Center, padding12horizontal/8vertical. RTS Canvas anchors span Y=.28→.46; RPG spans .36→.54, with24-unit side margins. Result notifications use those reserved lanes first. Their saved Construct graphs call the public registration node with the same shared WBP_ForgeContextHint.

The independent Sakura-derived center tint, icon layering and cooldown clock are unchanged. Native IFButton mouse enter/leave tracks pointer presence even when CommonUI suppresses hover for an interaction-disabled button. Its **On Appearance Updated** Hovered argument therefore includes that presence; CommonUI still blocks activation. There is no replacement gameplay validation.

Current runtime verification and rendered evidence are recorded in HUD hint polish. Physical-controller checks are deferred at the user's request; keyboard/pointer synthetic interactions do not certify hardware behavior. Automatic touch gestures, long localization, other layouts and final release acceptance remain separate work.

MENUS AND NAVIGATION

Custom buttons and confirmations

The standalone sample includes `WBP_ForgeActionButton` and `WBP_ForgeConfirmation` under `/Game/UI/Blueprint`. Their Designer trees and event graphs are editable. No Game Master widget tree or artwork was used.

A custom button in Blueprint

Create a Widget Blueprint with parent **IFButton**. You can use any layout. For automatic label typography and visible focus, name an optional Text Block **LabelWidget**, and an optional Border **FocusBorder**. Each binding is independent; both are optional. `Set Label` updates the bound label. Theme, enlarged text, hover, navigation focus and enabled state update the bound controls. Set **Label Alignment** in class defaults.

For other artwork implement **Event On Appearance Updated**. It supplies Theme, Presentation, Navigation Focused, Hovered and Enabled. The sample reads `Theme → Accent` and calls `Set Brush Color` on its narrow AccentMark. The base supplies CommonUI click/back routing, input presentation, sounds and focus styling. You can omit the optional controls and implement all visuals in this event.

CommonUI's standard On Button Base Clicked event remains your action hook. A hover does not imply keyboard focus. Use Navigation Focused to distinguish keyboard/controller navigation from pointer hover. Don't duplicate input detectors in each button.

A custom confirmation

Create a Widget Blueprint with parent **IFDialog**. Add an InterfaceForge Surface, IFText labels and your own IFButton Blueprint controls. Implement these graphs:

Blueprint workflow (text source; interactive diagram in the web guide)

```
flowchart LR
    A[Event Initialize Screen] --> B[Parent: Initialize Screen]
    B --> C[Get Dialog Data]
    C --> D[Set Text: Title and Message]
    C --> E[Set Label: Confirm and Cancel]
    E[Event On Initialized] --> G[Set Initial Focus: Cancel button]
    H[Cancel: On Button Base Clicked] --> I[Reply: Cancelled]
    J[Confirm: On Button Base Clicked] --> K[Reply: Confirmed]
```

Call the parent initialization before reading dialog data. It runs each time a pooled dialog receives new data; Construct alone is insufficient. The base cancels on Back and emits one completion result. Your Blueprint calls `Reply`; it doesn't broadcast completion itself.

On the player's **IFRootLayout**, set **Default Dialog Class** to your custom Blueprint. Construct an **IFDialogData** object, set its text and bind **On Completed**, then call **Push Dialog**. A missing data object returns None. With a valid class/data pair, the dialog enters the Modal layer. `Push Screen` remains available for explicit per-call classes.

The default native dialog uses a 560-unit preferred width, minimum desired height 280 and maximum desired height 480 inside a Scale Box (Scale To Fit, Down Only). Short confirmations remain compact; long messages can grow within the cap. Title and actions remain outside a Fill-sized message Scroll Box. Native actions use a wrapping row. The sample WBP_ForgeConfirmation applies the same bounded-message pattern with normal Designer widgets: place DialogMessage inside **Scroll Box**, set its Vertical Box slot to Fill, keep confirmation buttons below it, and allow focus scrolling. Set **Min Desired Height** and **Max Desired Height** on the containing Size Box instead of forcing one height. Text changes still use the existing **Initialize Screen** → **Set Text** graph. Long messages must not push exit actions beyond the viewport. The fit choice can reduce a card when space is insufficient; choose your own reflow strategy in Blueprint where required.

Expected: initial Cancel focus; Enter/controller accept activates the chosen button; Escape/configured Back cancels; opener focus returns. A new prompt must replace previous text. Confirmation results belong to the calling game adapter, not to InterfaceForge.

Observed verification

UE5.6.1 Win64 DX11 editor-game, automated mouse/keyboard: source build and authored asset compilation passed; menu Quit prompt populated; Escape cancelled; the same class accepted different expedition-return data; Ember updated surface, label/marker and focus; Enter activated custom Cancel; Right moved to custom Confirm; Enter confirmed and returned to Main. Screenshot: `Evidence/Visual/BlueprintDialog-CustomFocus-UE56.jpg`. Archived BlueprintSlice also passed the cooked custom dialog's cancel/restored focus/reopen/Right/confirm/return flow; see Evidence/PACKAGED_BLUEPRINT_SLICE.md. Physical controller/touch, extreme long localized text and the later responsive-wrapper recook remain open.

The later bounded-message revision was inspected live at 800×600 with 150% UI and text scales: both Blueprint/native dialogs reached the final paragraph with long action labels visible. Blueprint Escape and native Tab → Right → Enter returned one result each, restored the preview settings and Main focus. The short Quit prompt still displayed correctly and Cancel restored Quit focus. Evidence/LONG_DIALOGS.md records captures, logs and remaining translation/device/package gates.

MENUS AND NAVIGATION

Loading screens and safe returns

Use **Push Screen Async** when a screen class is a soft reference. InterfaceForge uses Unreal's streamable manager and CommonUI stacks. It presents an honest asset-loading fraction, restores the previous interface on cancellation/failure, and prevents a late result from pushing into an ended player flow. It has no game-specific controller, inventory or Mass dependency.

The playable Outpost and Emberwatch interfaces use original layouts and artwork. Game Master's design stays exclusive to that game. Its asynchronous navigation workflow informed this feature; no reference code, widget tree, material or design was copied.

Working Blueprint recipe

Set up **CommonGameViewportClient**, **IFNavigationInputData** and keyboard **Trigger Click** as described in INSTALLATION.md. Create a root through the owning player's **IFLocalPlayerSubsystem** → **Create Root**.

In an IFScreen Widget Blueprint:

1. From a button's **On Button Base Clicked**, call **Get Root Layout**.
2. Add **Push Screen Async** (category InterfaceForge → Loading). Connect Root. Set Layer to Menu, HUD or Modal. Choose your IFScreen **soft class reference** in Screen Class. Supply its initial Data and a player-facing Title. Scope may be the controller, a retained flow object, or this screen; it is weak and optional.
3. **Ready** supplies the initialized Screen. Perform any game-owned commit here. **Failed** supplies no screen and a readable Reason: show a dialog/notification and let the player retry or stay. **Cancelled** supplies no screen: leave the previous flow in place.
4. The exposed **Request** output supports **Cancel Loading**, **Get Load State**, **Get Load Progress**, **Get Loading Title**, **Get Loading Detail** and **Get Loading Screen**. Do not implement one polling timer per widget.

Blueprint workflow (text source; interactive diagram in the web guide)

```
flowchart LR
    Click[On Button Base Clicked] --> Load[Push Screen Async]
    Root[Get Root Layout] --> Load
    Class[Soft IFScreen class + initial Data] --> Load
    Load -->|Ready: Screen| Commit[Commit game-owned flow]
    Load -->|Failed: Reason| Retry[Dialog: Try again / Stay here]
    Load -->|Cancelled| Stay[Previous screen and focus]
    Cancel[Cancel button / CommonUI Back] --> Request[Cancel Loading]
    Request --> Stay
```

Expected result: the previous screen remains beneath a modal loading presentation. Its Cancel button gets navigation focus. A successful load pushes the supplied class and removes only its loading screen; failure/cancellation returns to the previous screen. Fast loads proceed after the required CommonUI transition, with no minimum duration or fabricated progress.

Actual saved graph: `/Game/UI/Blueprint/WBP_AshenMainMenu`, SettingsChoice's click → **Push Screen Async**, Layer Menu, soft class `WBP_AshenSettings`, Title “Opening settings”, Scope self. Failed → **IFNotificationSubsystem** → **Post Notification**, using Reason as Message. The saved settings screen opens its normal transaction on activation. Cancel settings restores preferences and the same menu. This graph compiles and is exercised through synthetic Slate input; see the verification record for device limits.

The menu's Begin action uses the public **Request UI Action** contract. Its separate Outpost/Pilgrimage adapter calls the same public async factory with the soft HUD class, and starts its demonstration state only on Ready. Failed uses the saved confirmation dialog with “Try again” / “Stay here”. This backend owns the flow commit and game data; none of it is required by the plugin. The direct **Begin Expedition / Begin Journey** methods are synchronous fixture/convenience paths; playable Begin navigation uses **Request Begin Expedition / Request Begin Journey**.

Replace the loading presentation

Create a Widget Blueprint derived from **IFLoadingScreen** and set it as your IFRootLayout Blueprint's **Default Loading Class**. No plugin source edit is needed. The saved `WBP_ForgeLoading` has an original centered, fitted surface, theme-aware typography/progress and the same custom `WBP_ForgeActionButton` as the menus. `WBP_ForgeRoot` and `WBP_EmberRoot` select it in their defaults.

For automatic binding, add optional named controls and enable **Is Variable**:

Name	Type	Behavior
LoadingTitle	TextBlock or IFText	Public request title
LoadingDetail	TextBlock or IFText	Current loading/failure detail
LoadingProgress	IFProgressWidget	Engine asset fraction; choose Linear, Segmented or another public style
CancelButton	IFButton or a Blueprint child	Cancel Loading and initial navigation focus

Use IFText/IFSurface/theme-aware progress for project typography, colors and scale. Fit/scroll the designer with ordinary ScaleBox, SizeBox, SafeZone and layout widgets. Do not hard-code a second user scale. The saved custom body and native fallback use fit-down behavior so enlarged text retains a reachable Cancel action.

For a different body, override **On Loading Updated**, read **Get Loading Request** and update your own controls. The parent event updates the optional named controls; call parent if retaining them. Your custom Cancel button calls **Cancel Loading**. CommonUI Back uses the same cancellation. If overriding **Initialize Screen**, call parent before accessing the request. The node also exposes On Loading Updated; it is a change signal, not a terminal completion.

Leaving Default Loading Class empty uses the native themed body. A custom ordinary IFScreen can instead compose a loading view with the exposed request; IFLoadingScreen is a convenience that retains Back and pointer-modal handling.

Lifetime, history and modal behavior

Only one request runs per root. A competing public request fails with a reason and leaves the current request untouched. Examples also reject repeated Begin actions while a request is pending.

The root retains the request; it registers with the game instance until one terminal output. Scope is weak; keep the scope alive elsewhere for the intended lifetime. Initial Data is retained until completion; a destroyed data actor cancels the request. The root and scope must remain valid when committing. Clearing the target layer or Modal, replacing those stacks, removing the loading screen through **Remove Screen**, or **Remove Root** cancels pending work. During clear/teardown, reentrant new requests are rejected.

Remove Screen removes the exact supplied screen without clearing unrelated history. A newer dialog can cover the loader; committing waits until the loader regains the settled foreground. Cancelling removes the loader and preserves the unrelated dialog. Close that dialog normally to return to the original menu. Do not call Deactivate Widget directly to cancel a custom loader: use Cancel Loading, CommonUI Back, or Remove Screen so the request is also ended.

Terminal status is set before removing presentation or broadcasting outputs, so repeated Cancel/Activate cannot send another completion. Loading/progress/lifetime checks use UI/core time and work while game time is paused. CommonUI owns focus restoration, routing and stack transitions; the normal input-presentation service also applies to loading screens/custom buttons.

Progress and limits

Progress measures the streamable class and its asset dependencies. Unreal can count failed assets as finished; **Ready/Failed**, not a value of 1, determines success. This is not map-travel, server-connection, shader compilation, download, or gameplay-generation progress. Keep those operations in their appropriate engine/game systems and supply a separate presentation adapter. This slice does not implement engine MoviePlayer pre-world loading or conceal blocking level travel behind a fake progress bar.

Soft references must be included by your cooking policy. The initial package omitted the example HUDs: native C++ path defaults did not establish cooked asset dependencies. The sample now provides `/Game/UI/DA_ForgeInterfaceAssets`, an engine **Primary Asset Label** with an explicit list of its23 owned Blueprint classes (22widgets and one world-presenter actor), **Cook Rule = Always Cook**, Priority1 and Apply Recursively. Its Asset Manager scan is enabled for the sample's `/Game/UI` directory with **Is Editor Only = false**; the label's **Is Runtime Label** is true. This policy belongs to the standalone example, not the UI plugin. It adds no broad project-wide cook directory or rule for a buyer's game.

In your project, add your own soft screens to an Asset Manager label/rules policy, or use an appropriate existing packaging policy. Configure **Project Settings → Asset Manager → Primary Asset Types to Scan** so labels intended to pull assets into a package are scanned with Is Editor Only=false. In the label's **Explicit Blueprints**, choose your screen classes and set Always Cook. Inspect your staged assets and run the packaged flow. A soft path resolving in the editor is insufficient. Unreal's asset management, cook rule and primary asset label documentation describes these engine controls.

If an asset is omitted, report Failed and give the player a return/retry route; do not start gameplay before Ready. Packaged outcomes and actual tested engine versions belong in Evidence/LOADING_RUNTIME.md. Physical controller/touch, split-screen and slow-storage stress remain distinct gates.

Unreal foundations: async action lifetime and streamable handle progress/cancellation. The implementation uses the installed engine APIs and respects CommonUI's transition lifecycle.

MENUS AND NAVIGATION

Navigate between example worlds

From Ashen Reach's main menu, **Visit Emberwatch** opens the RPG world. Its main menu offers **Visit Ashen Reach**. These are playable menu actions, separate from Begin and Return to menu. Current UE5.6 Shipping checks operate both menu routes: RTS → RPG → RTS before the RTS workflow, and RTS → RPG before the RPG workflow. Other versions and broader travel policies retain their separate gates.

The saved `WBP_AshenMainMenu` uses the same public widget and provider features as a developer-created screen. Its quiet `OtherWorldChoice` button calls **Request UI Action**, Action Name `VisitOtherWorld`. Each example provider validates that its gameplay flow has not started and that another travel request is not already pending.

The providers return an optional **IFPresentationData** named `FrontEndOther`. Its **Title** supplies Visit Emberwatch or Visit Ashen Reach. On activation, the editable Blueprint **Read Menu Text** function resolves that model with a fallback, then **Set Label** updates the button. Read Menu Text has execution pins in this sample: connect its execution output before using its returned text. The existing arrival-animation graph continues afterward.

Own the root lifecycle around travel

An InterfaceForge root belongs to its owning local player/controller context. Before the example provider calls Unreal's **Open Level**, it uses **Get IFLocalPlayerSubsystem** → **Remove Root**. This cancels its screen loads, clears presenters and releases input observation before the previous world is destroyed. The new local PlayerController's Begin Play uses **Create Root** → **Push Screen** with the new world's data source.

Blueprint workflow (text source; interactive diagram in the web guide)

```
flowchart LR
    A[OtherWorldChoice: On Button Base Clicked] --> B[Request UI Action: VisitOtherWorld]
    B --> C[Provider validates frontend and travel state]
    C --> D[Remove Root]
    D --> E[Open Level]
    E --> F[New local controller: Create Root]
    F --> G[Push Screen with new provider]
```

Keep level selection, travel validation and loading policy in your project or provider. The UI core supplies screen/root features without imposing a game's travel architecture. Its soft-screen loading component does not replace MoviePlayer or a project's map-loading solution.

The packaged Shipping example starts in Ashen Reach. Use the actual menu to reach Emberwatch; a command-line map request was ignored by the tested Shipping application. Shipping verification records24primary/reload cases and36menu travel trails across UE5.6.1/5.7.4/5.8.3, including both directions, on its pre-localization-refresh revision. The fixture is explicitly opt-in; normal play does not spawn it.

The main menu uses a bounded action Scroll Box with **Scroll When Focus Changes = Instant Scroll** and **Navigation Destination = Center**. Enlarged presentation may scroll the four actions; navigation must reveal the focused action without reducing text size. World-choice menu regression evidence verifies actual Settings → Visit → Quit → Visit → Settings navigation and fully visible focused Quit in both genres/sizes on UE5.6.1 source. Refreshed labels have separate source/Development localization proof. Physical controller and fresh final candidate/package acceptance remain separate gates.

CONTROLS AND SETTINGS

Blueprint key remapping

The current four-tab settings example embeds `IFBindingEntry` rows directly under `IFSettingsMenu`, sharing the parent **Apply/Cancel** session. Follow settings composition for that workflow. The standalone screen below demonstrates the lower-level remapping APIs and owns its own separate transaction; do not nest it inside a shared settings session.

`WBP_AshenControls` is an IFScreen Blueprint with two **InterfaceForge Key Selector** controls (keyboard/mouse and controller), an IFActionPrompt, status text and Apply/Cancel/Restore defaults buttons. IFKeySelector is Unreal's InputKeySelector with theme and text-size presentation; it does not own mappings. Developers may use the ordinary engine selector or their own capture interface with the same remapping APIs.

Few-node first-slot recipe

Get **IFRemappingSubsystem** through **Get Local Player Subsystem** using the owning player. On **On Activated**, call **Begin Edit** and store its Boolean result as OwnsRemapSession. Show Failure Reason if it fails. Editing is allowed only while this screen owns the session.

Blueprint workflow (text source; interactive diagram in the web guide)

```
flowchart LR
    A[Selector: On Key Selected] --> B[Break Input Chord: Key]
    B --> C[Stage Registered Key]
    C --> D{Return Value}
    D -->|False| E[Failure Reason → status text]
    E --> F[Refresh selector from registered preview key]
    D -->|True| G[Has Pending Changes → unsaved status]
```

Stage Registered Key receives Mapping Name = IFExampleInteract, Family = Keyboard or Gamepad, and Key. It resolves the actual registered slot/hardware identifier. It rejects invalid/analog/incorrect-family keys, reserved navigation keys and conflicting keys under the configured policy. No guessed DefaultKeyboardMouse hardware name is needed. For multiple slots or hardware-specific interfaces, use **Get Registered Mapping** → **Set Members in FMapPlayerKeyArgs** → **Stage Key Mapping** instead.

To refresh: **Get Registered Preview Key** → **Make Input Chord** → **Set Selected Key**. Surround programmatic updates with your RefreshingKeys Boolean; do not stage mappings from the selector's refresh event. The Blueprint example gates On Key Selected while refreshing. Keyboard capture disables gamepad/modifier chords; the controller row permits gamepad capture and validation rejects wrong-family keyboard selections. This example maps single digital keys, not modifier combinations.

Apply calls **Apply**, branches on its result, then clears ownership and deactivates on success. Failure leaves the draft open. Cancel/Back clears ownership, calls **Cancel** and deactivates. **Event Destruct** also cancels an owned abandoned session. Restore defaults calls **Stage Registered Default** for each displayed family, refreshes selectors and still requires Apply. The IFActionPrompt displays the applied engine mapping, so it changes when Apply rebuilds controls, not while a draft is merely staged.

Key profiles use Unreal's **EnhancedInputUserSettings**. Enable those settings and register a player-mappable Input Action/Mapping Context before opening controls. The sample registers IFExampleInteract for keyboard E and Gamepad Face Button Left; saved remaps replace those defaults. Gameplay routing stays in the example controller. UI acceptance/back remain the CommonUI input-data actions and reserved keys; the example does not remap its only exit.

UE5.6's engine **Save Settings** has no returned I/O result. Remapping Apply can report validation/profile/mapping failures; it cannot currently report a disk-save failure through that engine call. Persistence must be verified on target platforms. UI preference Apply separately checks SaveGameToSlot's result. This limitation should remain explicit in a release that uses the stock Enhanced Input saving API.

Evidence boundary

UE5.6.1 build and Blueprint authoring passed, 0 commandlet errors/warnings. Initial compilation repairs addressed a local name shadow and an unexported editor helper method; no engine files changed. First real-RHI opening found a style-lifetime crash: UE5.6's InputKeySelector style setters hand the caller's style address to Slate. IFKeySelector now retains resolved styles as widget-owned fields; rebuilt live opening/reopening/capture passed. Staging G showed an unsaved binding while the applied prompt remained F; Back cancelled it and reopening restored F. Apply G changed the Blueprint HUD prompt; clean restart retained G. The opt-in engine-event fixture resolved the current registered G binding and triggered the actual Enhanced Input SelectNext action (Scout 1 → Scout 2). This is engine integration evidence, not physical-device proof. Packaged BlueprintSlice also captured/applied G and staged default E while the applied prompt stayed G. No physical controller is currently reported by XInput 1.4 (four disconnected slots); this does not enumerate every possible non-XInput/touch device. Physical controller/touch acceptance stays open.

Early native-window G/E pulses did not trigger the gameplay action: diagnostics showed down/up reaching the controller on one frame,2ms apart. The later120ms Slate-event fixture did run after Begin, resolving the persisted G mapping; the controller received down on frame243, Enhanced Input dispatched SelectNext on244, and release arrived on304. Scout1→2 was observed. Input-route evidence records both attempts. This proves that software Slate→viewport→controller→Enhanced Input route at its recorded revision, not physical input or every newer packaged revision.

Current glyph revision: the example opts into original neutral CommonInput face-position art; remap Apply refreshes IFActionPrompt in place, and empty Action Label shows only a glyph/readable key. RPG header keeps the cue visible beside scrolling prose with a.62shared divider. Current29Blueprint source/archive runtime proof is5.6; the newer28-class sample proof above predates it. Glyph authoring recipe and exact runtime/limitations. Physical controller tests remain explicitly deferred.

CONTROLS AND SETTINGS

Remap-aware controller glyphs

InterfaceForge uses Unreal's CommonInput controller-data assets. It does not keep a second key-to-icon manager. The original **BP_IFGenericController** is a normal Blueprint derived from **CommonInputBaseControllerData**, with four neutral face-position symbols. No controller-brand artwork or reference-game packages are included. Physical controller tests are explicitly deferred because the user has no controller available.

Opt in through CommonInput

1. Enable CommonUI/CommonInput/EnhancedInput as described in installation.
2. In **Project Settings → Game → Common Input Settings → Platform Input → Windows**, set **Default Gamepad Name = Generic** and add **BP_IFGenericController** to **Controller Data**. Configure other target platforms separately.
3. Register your player-mappable Enhanced Input context/user settings, then place an **IFActionPrompt** in your Widget Blueprint. Set **Mapping Name** to your registered row and **Action Label** to the player-facing action, such as Interact.

The sample opts in only in its own DefaultGame.ini:

```
[CommonInputPlatformSettings_Windows CommonInputPlatformSettings]
DefaultGamepadName=Generic
+ControllerData=/InterfaceForge/UIForge/Input/BP_IFGenericController.BP_IFGenericController_C
```

The selected gamepad name must match the controller-data class. Installed5.6 otherwise selected Windows; a class named Generic alone did not resolve its brushes. This is project configuration, not a core plugin override. Hardware identification/other controller names need your own CommonInput data and target-device validation. See Epic's CommonUI controller-data setup and controller-data API.

Original default art and customization

Engine key	Material instance	ActivePosition
Gamepad Face Button Top	MI_IFControllerNorth	0
Gamepad Face Button Right	MI_IFControllerEast	1
Gamepad Face Button Bottom	MI_IFControllerSouth	2
Gamepad Face Button Left	MI_IFControllerWest	3

M_IFControllerFace is a translucent UI material drawing a four-circle cluster, with the relevant position filled. **ActivePosition** is its scalar parameter; the supplied instances choose0-3. Brush Image Size is28×28. Other circles are subdued outlines. These are neutral positions, not claims about a particular brand's printed button labels. Native prompt glyphs retain their desired aspect beside enlarged text.

Duplicate **BP_IFGenericController** into your project. In Class Defaults, change **Input Brush Data Map** keys/brushes, image sizes or tint, and select your copied class in the platform settings. Brushes can use your own UI materials or licensed textures. Choose Input Type/Gamepad Name appropriate to your project. You can use additional CommonInput classes for keyboard/mouse and other controller families. The default set supplies only the four face positions; unsupported keys retain readable text fallback.

Applied mappings and minimal cues

The prompt resolves the current **applied** key from the registered mapping row. **Begin Edit** → **Stage Registered Key** does not change that key or glyph. **Cancel** discards the draft; successful **Apply** refreshes the existing prompt through EnhancedInput's **On Settings Applied** event. Family/key-profile changes also refresh it. No per-widget mapping polling is needed. Controls recipe gives actual stage/apply/cancel nodes and validation.

Leave **Action Label** empty when the surrounding UI already identifies the action. The native body then shows only the glyph, or only the readable key when a glyph is unavailable. It does not leave an empty separator or reserve a hidden text column. The original RPG example uses this minimal cue beside its context reading region; its Interact/Pause controls retain their original height. Prose remains separately scrollable. The RTS footer keeps the label Select next scout beside its glyph/key.

Make a custom prompt body entirely in Blueprint

Create a Widget Blueprint with parent **IFActionPrompt**, add your own Image/Text widgets, and implement **On Prompt Updated**:

Blueprint workflow (text source; interactive diagram in the web guide)

```
flowchart TD
    E[On Prompt Updated] --> B{Has Resolved Glyph}
    B -->|True| T[Resolved Glyph → Set Brush on your Image]
    T --> L[Action Label → Set Text; show Image]
    B -->|False| T[Prompt Text → Set Text; collapse Image]
```

Resolved Key, **Resolved Glyph**, **Has Resolved Glyph**, **Prompt Text** and **Action Label** are public Blueprint properties. A Designer root replaces the native body; the base keeps resolution/subscription lifecycle and calls your event. Style your own Image with **Set Color and Opacity** or normal theme tokens. **Refresh Prompt** is available after an intentional live change to Mapping Name/Action Label. Do not manually set the applied key or glyph in response to staging; let the engine's applied mapping supply it.

The saved **WBP_ForgeAppliedPrompt** proves this custom-body workflow in the RTS footer. Its **PromptBody** Horizontal Box contains **CueGlyph** (Image) and a bounded reading region for **CueCaption** (IFText). **On Prompt Updated** uses the branches above, then **Get Owning Player** → **IFLocalPlayerSubsystem** → **Get Theme** → **Accent** → **Set Color and Opacity** on CueGlyph. IFActionPrompt refreshes this event when the local-player theme changes, as well as when input or an applied mapping changes. Replace the body in Blueprint while keeping the public resolution lifecycle. RPG uses the minimal native prompt, so the custom-body proof applies to RTS.

For longer action/key text, the saved RTS body uses **CueReadingBounds** (Size Box, **Max Desired Height =84**) → **CueReading** (InterfaceForge Scroll Box) → CueCaption. The bounded reading region has a Fill Horizontal Box slot; CueGlyph retains its own Auto slot. CueReading is focusable, clips to bounds and uses Instant Scroll on focus changes. PromptBody is **Not Hit-Testable (Self Only)** so its reading child can receive wheel input. The whole Action Label/Prompt Text is retained; the region scrolls instead of growing until it removes the adjacent resource reading area. Adjust the ordinary Size Box limit and slot alignment in Blueprint for your typography/layout. The native prompt has no global height restriction; custom bodies choose their own layout policy.

The expanded Latin-text fixture found and repaired a zero-height resource viewport at maximum recorded scale. Actual wheel input now reveals the caption while preserving its applied-key text. This tests wrapping/overflow, not a shipped German translation or a culture switch. Responsive text evidence records the exact revision and remaining gates.

For CommonUI navigation-action icons, use **CommonActionWidget** with your configured CommonUI row. The saved main/pause identity bands now demonstrate icons and actual-key fallback through the menu cue recipe. **IFActionPrompt** continues to resolve registered Enhanced Input gameplay rows.

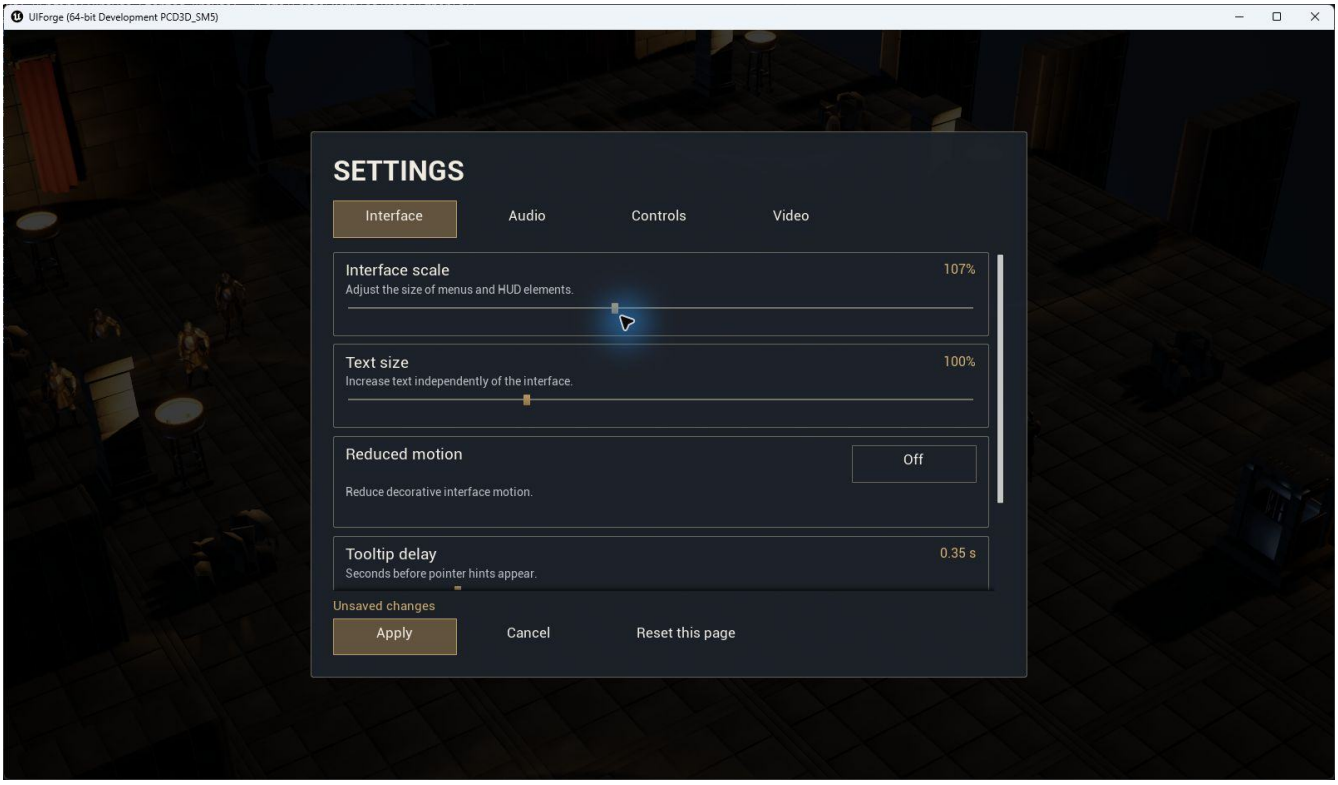
Expected: controller intent selects the configured face-position glyph; applying a supported remap changes that glyph in place; meaningful mouse/keyboard input restores the appropriate applied key/brush or readable fallback while modal focus remains valid. Runtime and packaged scope, actual failed shader/config/remap/culling runs and remaining device/layout gates are recorded in controller glyph evidence.

CONTROLS AND SETTINGS

Build a settings menu in Blueprint

Follow the settings tutorial with editor images: exact locations, real Designer/graph images and expected results. This page supplies the detailed contract/reference.

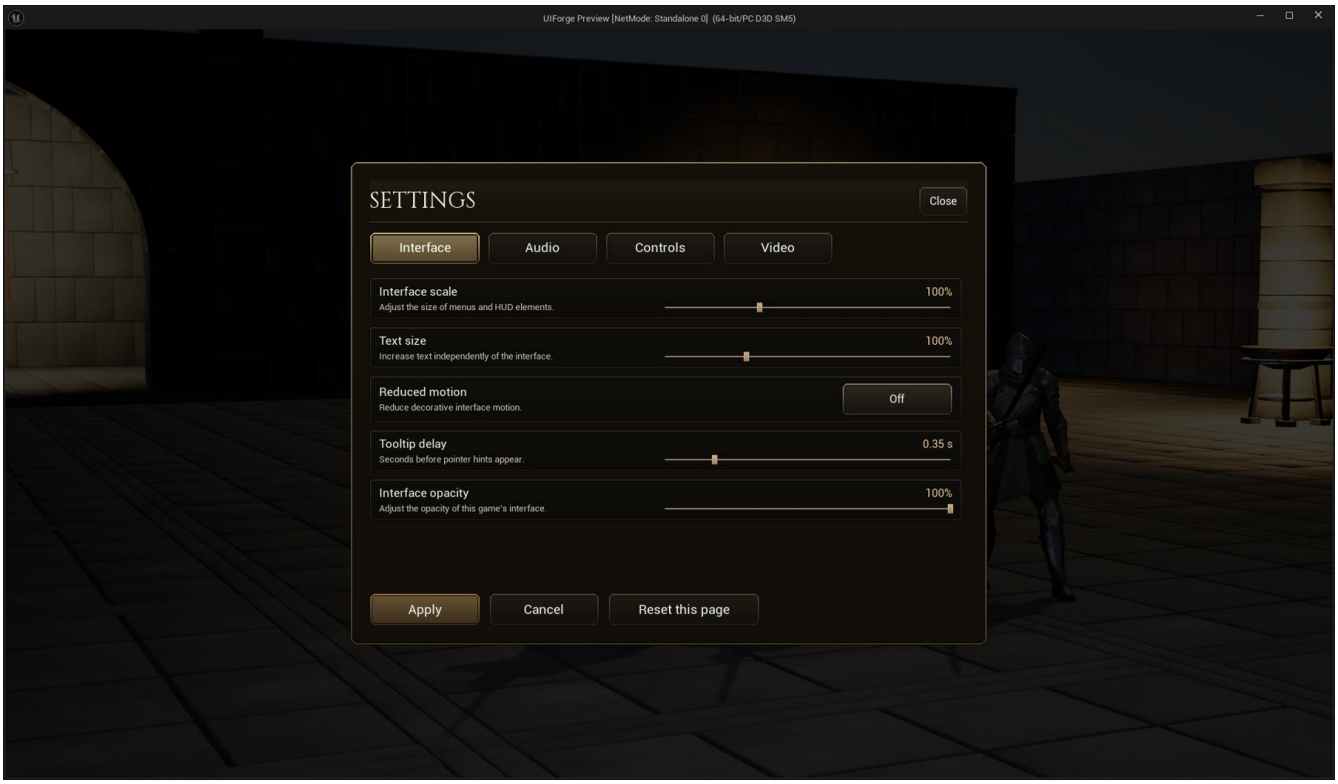
Verified on UE 5.8.3 / Win64 within the recorded functional passes. The saved four-tab menu runs from main and pause menus, including a separately installed Blueprint-only host. Shared Cancel/Apply, remapping-aware prompts, new-game-instance persistence, validation and a saved Blueprint custom provider have runtime evidence. The older Development package loads its themed menu and previews scaling; Escape cancels its draft. That package predates the latest presentation and Designer-preview corrections. Physical-controller verification remains deferred.



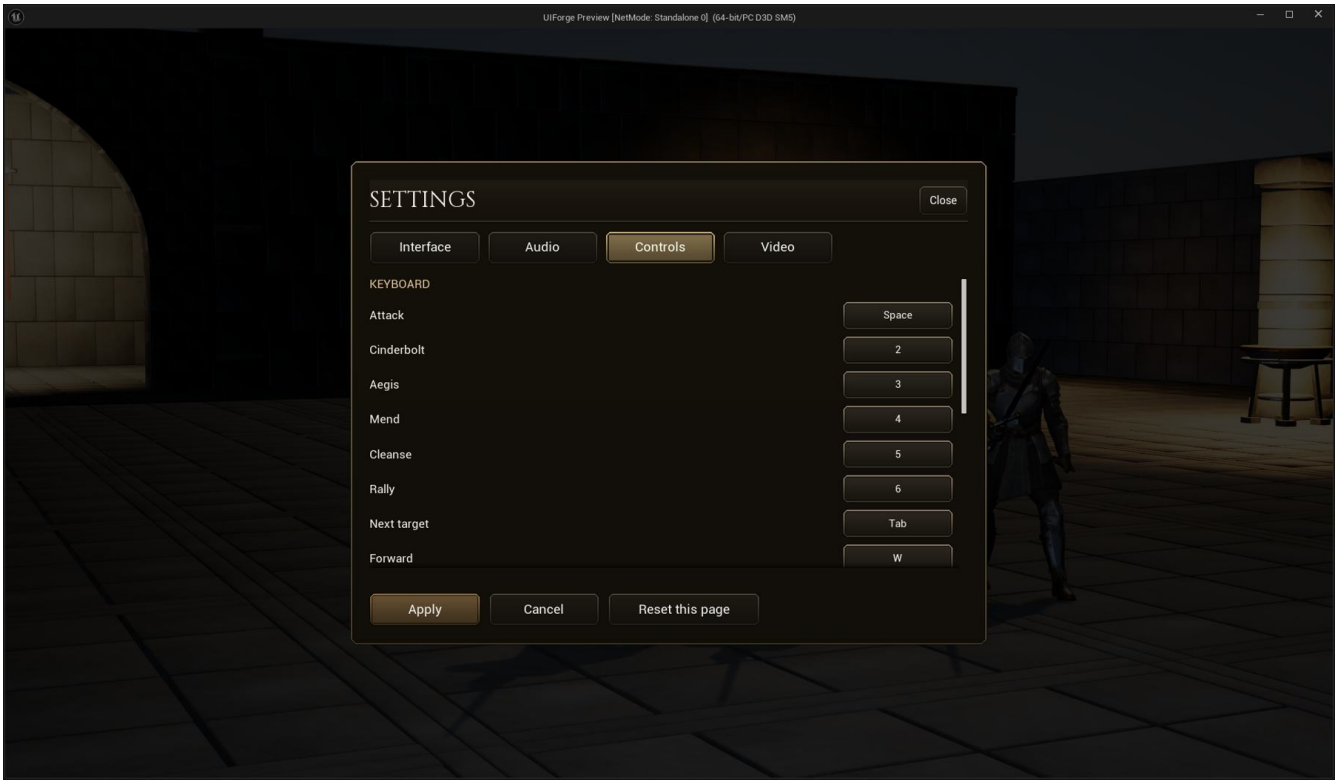
Historical packaged settings menu with an unsaved scaling preview, before the current presentation correction

Current RPG/RTS presentation

Both examples use the same public IFSettingsMenu/IFSettingRow session contracts. The current replaceable **WBP_IFRPGSettingRow** is a normal Designer Blueprint with compact label/help and value/control columns, a themed On/Off button, and validation text that collapses when empty. Interface, Audio, Controls and Video retain one shared Apply/Cancel/defaults session. Binding rows similarly reserve space for status only when needed. Their labels and page order are editable assets, not a game-specific C++ menu.



Current four-tab settings in Unreal Engine



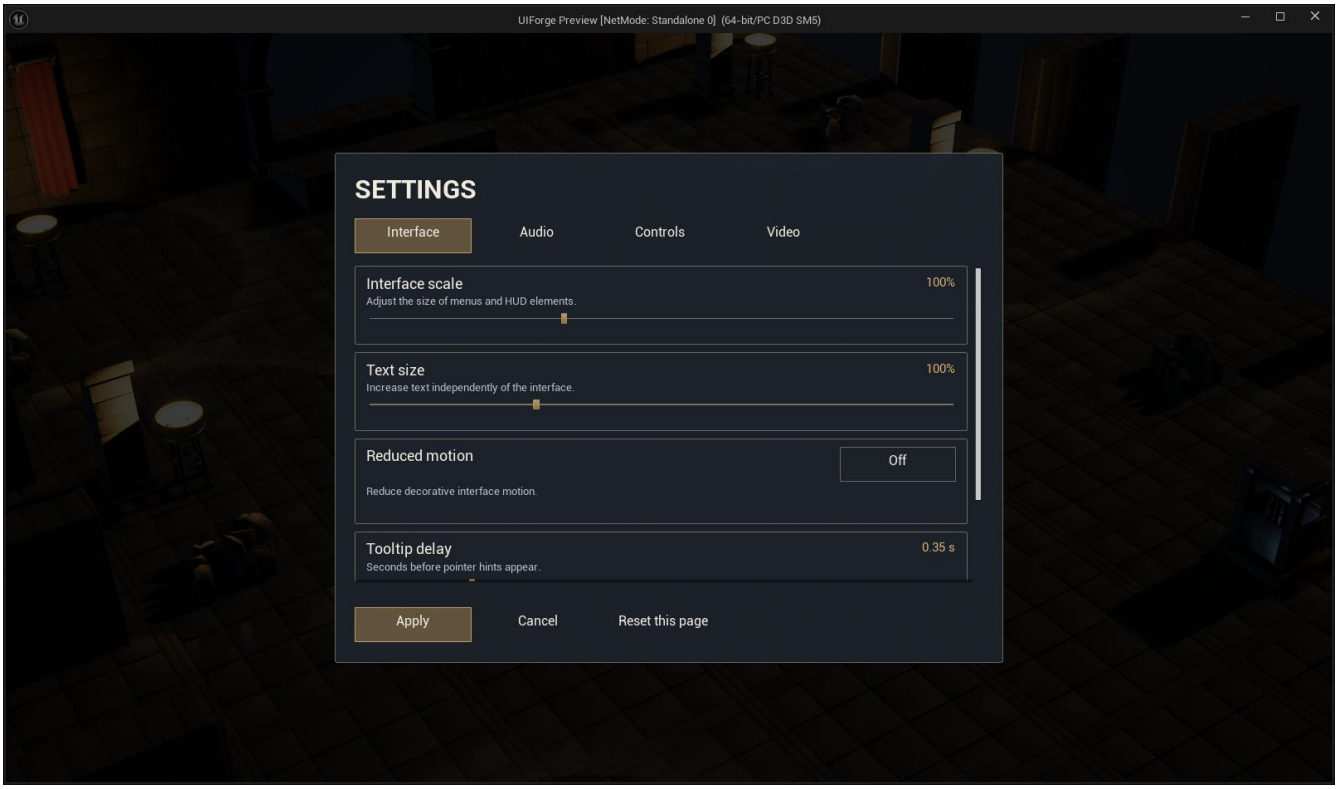
Current Controls page with real Enhanced Input key selectors

Author the appearance

1. Create a Widget Blueprint with parent **IFSettingsMenu**. Add your background, title, tabs, content and footer in UMG Designer; the parent does not construct a widget tree.
2. Add a **Widget Switcher** named `SettingsPages`. Put one Scroll Box/page layout inside for each category. In Class Defaults, set **Page Ids** in the same order, such as `Interface`, `Audio`, `Controls`, `Video`.

3. Place **IFSettingsTabButton** widgets, set each **Label** and **Page Index**, and style them like other IFButton widgets. The host connects the tab behavior automatically. A custom tab implementation can instead call **Select Settings Page**. **On Settings Page Changed** exposes the selected index/ID for Blueprint animation or focus rules.
4. Place IFButton controls named `ApplyChoice`, `CancelChoice` and `DefaultsChoice`. Name a Text Block `SettingsStatus`. DefaultsChoice means **Reset this page**, not the whole menu; label it accordingly. For an explicit global reset, call **Restore All Defaults** on a separately labelled control.
5. Set **Initial Focus Name** to your first tab's Designer name (for the example, `Tab0`) and **Fallback Focus Name** to `CancelChoice`. Open this class through the existing **Push Screen** node on the Menu layer from main menu or pause. Back uses the same cancel path.

The included editable example is `/Game/ForgeExamples/UI/Settings/WBP_IFSettingsMenu`. The two reusable row assets are `WBP_IFSettingRow` and `WBP_IFSettingRowCompact` in that folder. They are saved Designer trees using public plugin widgets; the example editor authoring utility is not a runtime dependency.



The same saved four-tab settings menu above paused gameplay

Define and place a setting

Create a Data Asset of class **IFSettingDefinition**. Assign a stable **Setting Id**, matching **Page Id**, localized **Label/Help**, **Binding**, **Control**, **Minimum/Maximum**, **Step** and **Default Value**. Choice definitions also need localized **Options**, each with its stable numeric value. Toggle definitions use range zero to one and exact values zero/one. Binding ranges cannot exceed the underlying engine setting's supported range.

Create a Widget Blueprint with parent **IFSettingRow**. Add only the controls your presentation needs:

Optional Designer name	Type	Behavior
LabelText / HelpText / ValueText / StatusText	Text Block or IFText	Definition label/help, current numeric value and validation feedback
ValueSlider	Slider	Range/step and reversible draft edits
ValueToggle	Check Box	Boolean draft value

Optional Designer name	Type	Behavior
ValueToggleButton	IFButton	Optional themed On/Off button with normal CommonUI focus and touch-sized hit area; takes priority over ValueToggle
ValueChoice	Combo Box String	Options mapped back to stable values

Place the row on a page and choose its **Definition**. The menu discovers definitions and initializes row sessions across all pages; switching tabs retains drafts. For a custom control, implement **On Setting Refreshed** to display its value and call **Submit Setting Value** to edit it. Do not duplicate engine setters in each row. **On Setting Submitted** reports success or a validation reason. Initialization/unsubscription belongs to the row lifecycle.

Display Multiplier, Value Suffix and Fractional Digits change formatting without changing engine storage: e.g. a stored audio value of 0.8 can display as 80%. Keep numeric text unwrapped and give its column enough width for the longest localized value.

Builtin bindings currently use the existing UI preferences/audio mix service and Unreal GameUserSettings: UI/text scale, interface/game volume, tooltip delay, reduced motion, VSync, frame limit, normalized resolution scale, shadow/view-distance quality. Video values remain staged until Apply; current builtin rows do not change display resolution or window mode. Those changes require Unreal's confirm/revert flow rather than calling Apply Resolution Settings from a slider.

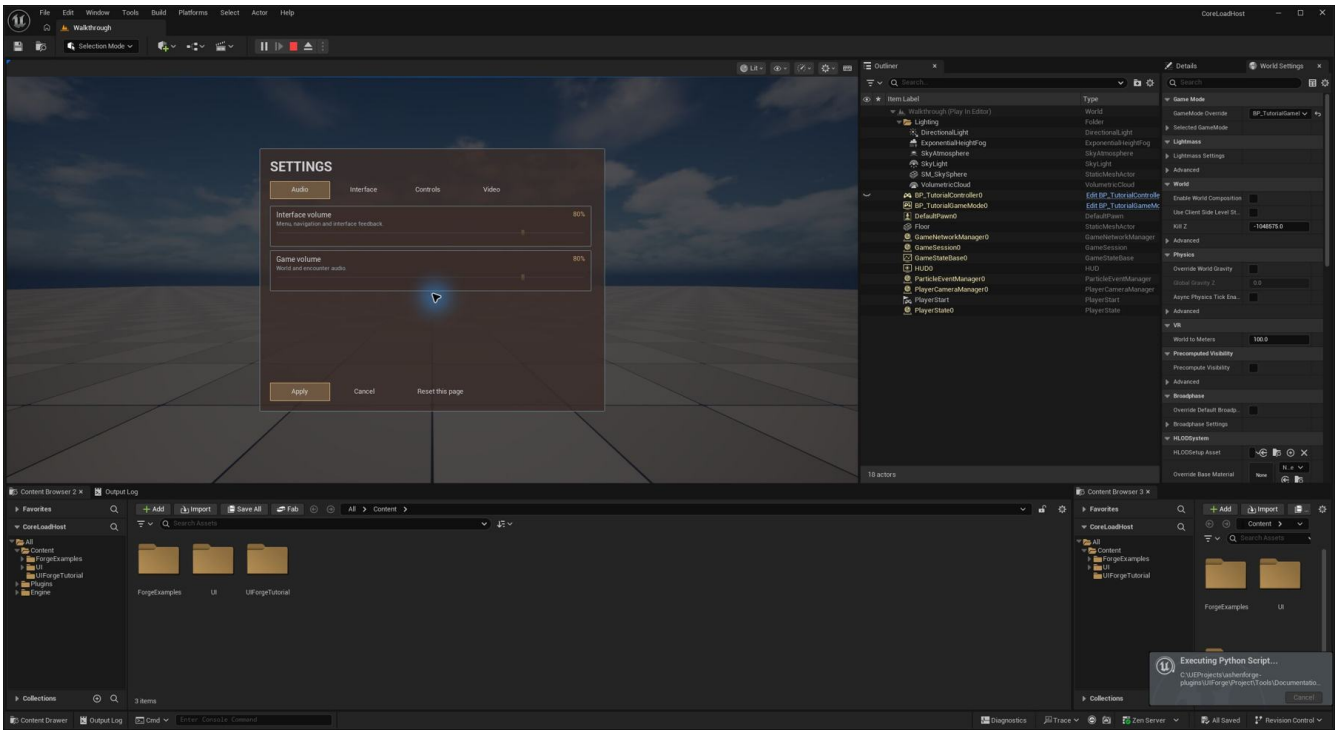
Controls share the same edit session

Enable **Include Input Bindings** on the menu. Add IFBindingEntry Blueprint rows to the Controls page, choosing **Mapping Name**, input **Family** and label for mappings already registered through Enhanced Input. Do not embed a separate IFRemappingScreen with its own Apply button: the parent owns a single preference/remapping session. Existing IFKeySelector behavior captures keys and uses the project's CommonInput glyph data.

Set **Binding Page Id** to the relevant page. Page defaults reset displayed binding rows when that page is selected. Apply commits staged keys together with other settings; Cancel discards them. Reserved keys and conflicts still use the existing remapping service. Missing mapping/context registration is an error to fix in game setup, not a reason to invent an independent key store.

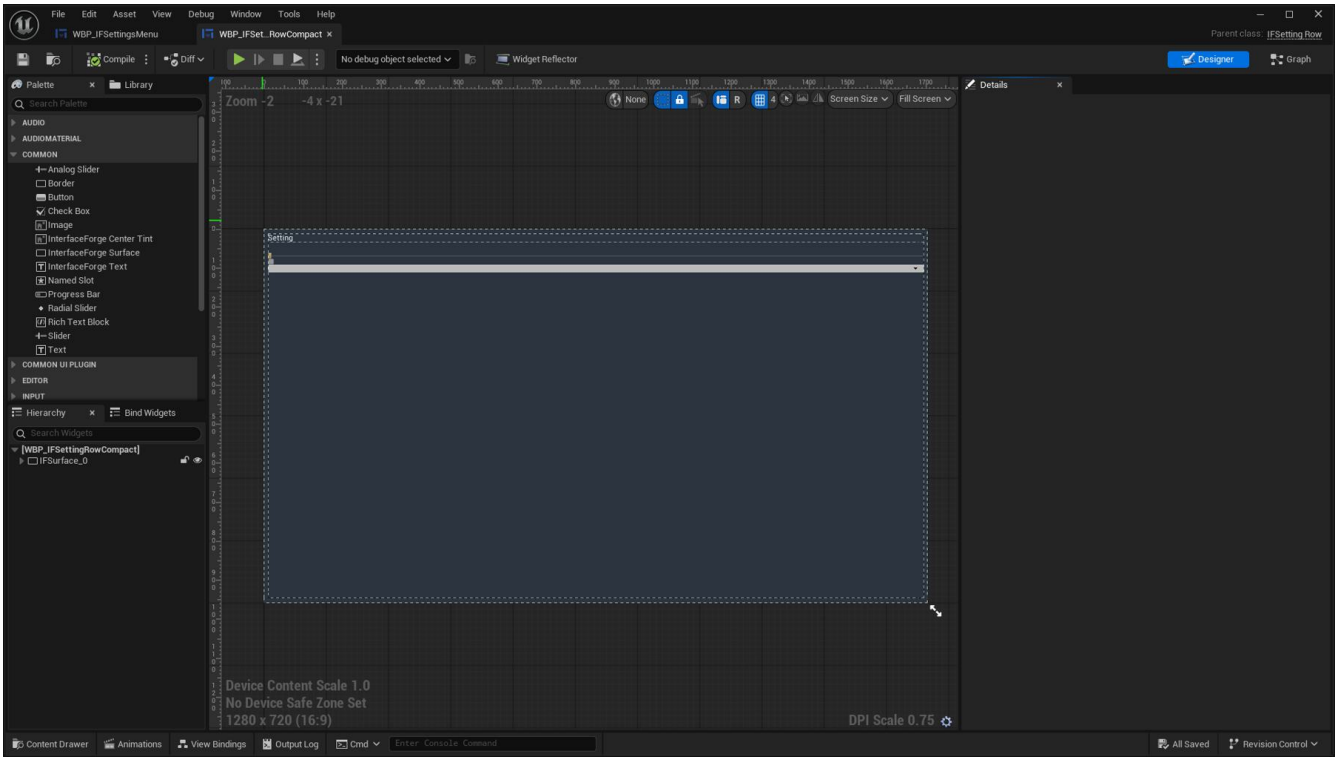
Add a project-specific provider

The example's Interface opacity row uses **WBP_IFSettingRowCompact**, while routine rows use **WBP_IFSettingRow**. Both inherit IFSettingRow and share the same session contract. The independent host runs this replacement and its custom provider. Replace a row in Designer, keep its Definition, then move pages and update Page Ids/Tab Page Index together; neither change requires plugin C++.



Independent host: Audio moved before Interface and theme replaced through public properties

This runtime proof uses UMG's **Get Child At** → **Clear Children** → **Add Child**, reordered Page Ids, **Set Label**, **Select Settings Page** and the local subsystem's **Set Theme**. In Designer, arrange the switcher pages directly instead. Keep each definition's Page Id stable: changing order does not rename category identities or create another editing session.



Actual Designer view of the replacement setting row

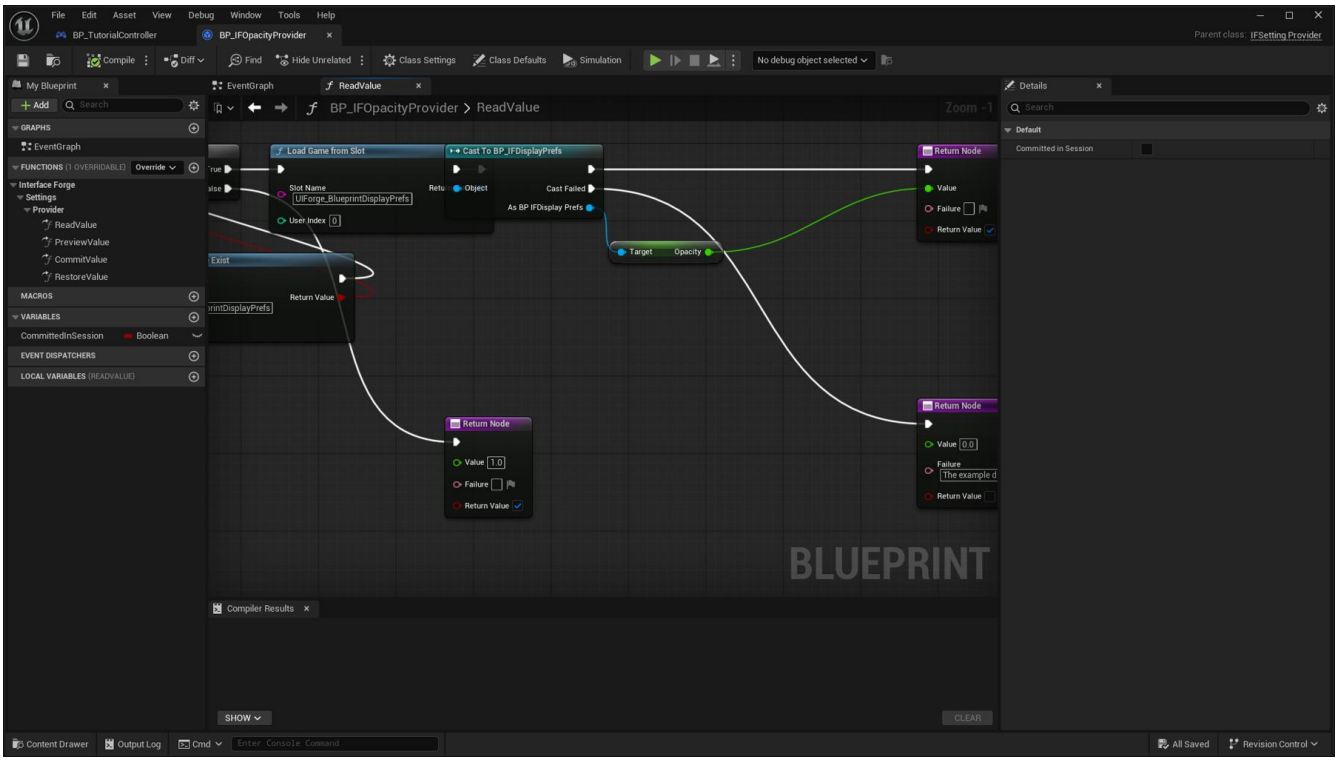
Create a Blueprint derived from **IFSettingProvider**, select Custom binding on the definition and choose its **Provider Class**. Implement the real functions:

- **Read Value**: read the game's committed setting and return Value/Success, or Failure.
- **Validate Value**: reject unsupported values before any effect.

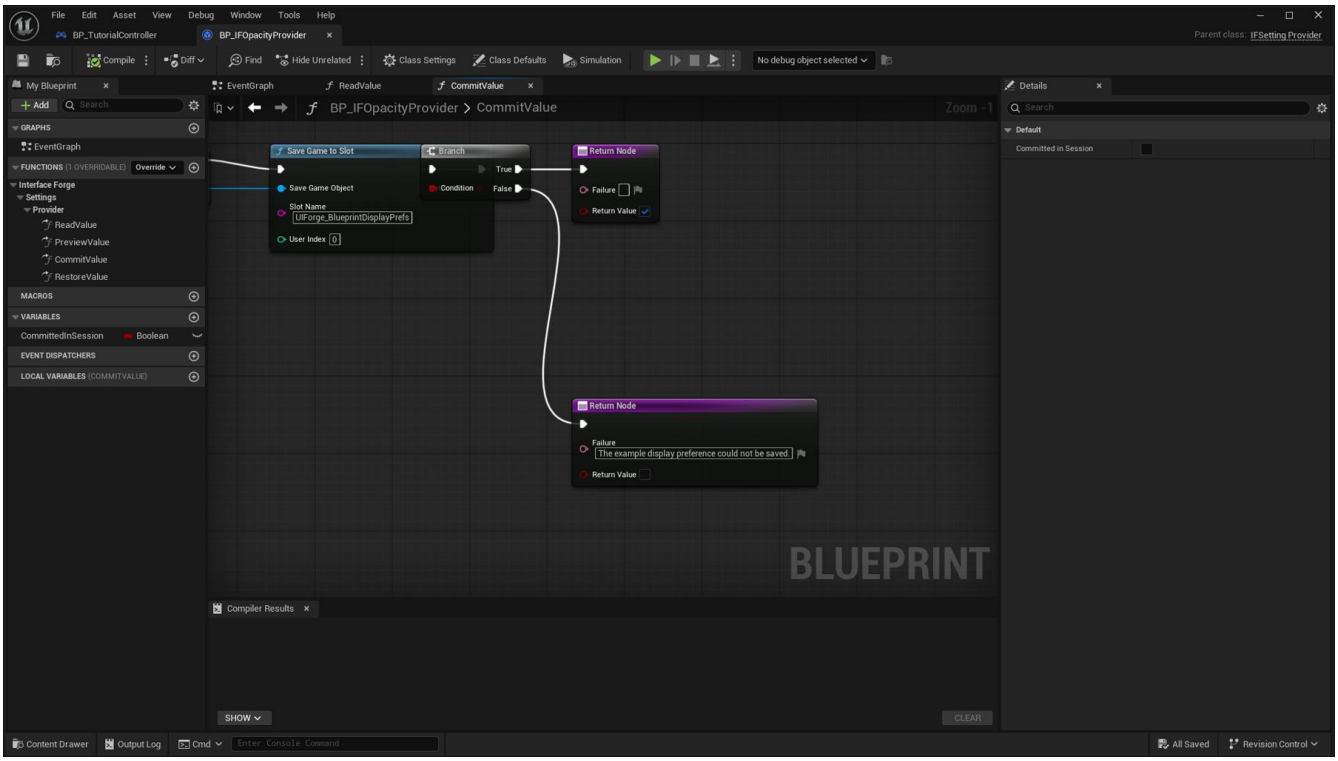
- **Preview Value**: apply a reversible temporary effect if the definition enables preview.
- **Commit Value**: apply and persist through the game's own store; return failure if persistence fails.
- **Restore Value**: restore the original value after Cancel or a failed Apply. This must also undo a partial custom commit; do not change unrelated settings.

The owning Player Controller and stable Setting Id are supplied to every function. The menu instantiates a provider per custom definition. Its public **Editing Session** exposes **Get Setting Value**, **Set Setting Value**, **Has Pending Changes**, **Restore Defaults**, **Apply Session** and **Cancel Session** for advanced Blueprint layouts. **On Session Changed** updates presentation without per-widget population polling.

The included custom definition `DA_Setting_InterfaceOpacity` selects `BP_IFOpacityProvider`. Its Read Value graph loads `BP_IFDisplayPrefs` from the game's SaveGame slot; Preview Value changes the owning local root's render opacity; Commit Value saves; Restore Value undoes preview or a partial commit. The example controller invokes the same public Read/Validate/Preview contract at startup through its map-owned Startup Settings definitions. In your game's Blueprint startup, construct your provider after Create Root and invoke those functions with the owning controller to present the saved setting. Keep that object in a variable if it owns subscriptions.



Actual Blueprint Read Value detail: load the game-owned display preferences and return opacity



Actual Blueprint Commit Value detail using the game-owned SaveGame and reporting failures

Runtime checks prove that this saved Blueprint provider changes the real root, Cancel restores it without rewriting its save, Apply saves, and a new provider instance reloads the value. The shared menu test also stages a keyboard binding and scale across tabs, cancels both with Back, then applies both and loads them in a new game instance. These are development checks, not packaged or physical-controller certification.

Transactions and limitations

For a custom dropdown option, return a **Widget Blueprint/UserWidget** containing your text or icon in its Designer tree. ComboBoxString keeps a Slate widget; an unattached plain Text Block returned from a generator can lose its UObject/style during garbage collection. The default themed option supplies a UserWidget lifetime owner automatically. Custom Blueprint generators remain supported and are not replaced by the default.

Back dismisses an open settings dropdown before cancelling the menu. This also applies when the Back input changes from pointer/keyboard to controller presentation. If Slate clears focus while removing the popup window, the row restores focus to its choice control while its screen is still on top. The small closure watch runs only while a dropdown is open; closed settings rows do not poll.

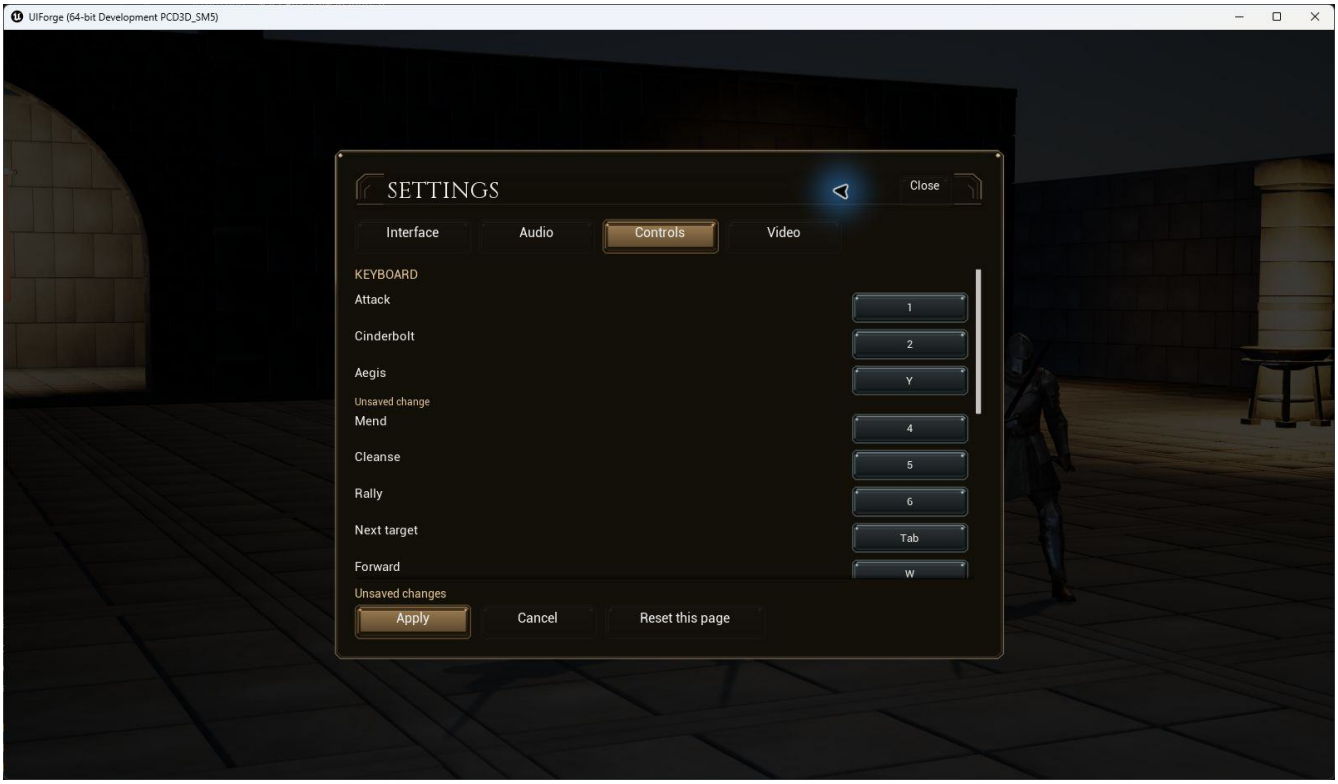
The menu validates drafts before applying. Cancel restores preference preview and discards key drafts. Custom commits have rollback hooks and failures remain visible with drafts retained. This is not an atomic write across SaveGame, Enhanced Input and user-defined persistence. Enhanced Input's Save Settings and Unreal's GameUserSettings persistence expose no success result to this layer, so a disk-write guarantee for those systems is not advertised. Do not use a provider that cannot restore its own partial changes.

Focus/navigation remains based on CommonUI activation ownership; individual rows are ordinary user widgets. See Epic's CommonUI design guidelines and GameUserSettings API for the engine responsibilities. Physical controller testing remains deferred by the user.

For a row added after the menu opens, register its definition in **Additional Definitions** before opening the screen. In Blueprint, use **Create Widget** with your row class and owning player, **Set Definition**, **Add Child** to the page, then **Initialize Setting** with the menu's **Editing Session**. Designer rows are discovered and initialized automatically. A runtime-created row was verified through this public workflow, including dropdown collection, controller Back and focus restoration.

Current packaged remapping proof

In the matching UE5.8.3 Win64Development package, selecting the Aegis key and typingY shows the staged value below. Apply updates the HUD/book prompt. After exiting and restarting the entire process,3 leaves the skill idle andY consumes12Resolve and starts12-second recovery. The test uses an isolated user directory; source user saves remain unchanged. This is real mouse/keyboard input, separate from software controller and settings-session fixtures.



Actual packaged key-capture draft

CONTROLS AND SETTINGS

Authoring settings in Blueprint

For new menus, use **Blueprint settings composition**. It documents the current `IFSettingsMenu`, definition/provider/session and reusable-row workflow, including a four-tab saved example and one shared Apply/Cancel for preferences and input bindings. The low-level subsystem recipe below describes the historical `WBP_AshenSettings` screen; its separate Controls transaction is not the current product menu's behavior.

`WBP_AshenSettings` is a saved IFScreen Blueprint. Its Designer uses ordinary Sliders, IFText, IFSurface, WBP_ForgeActionButton and an InterfaceForge Scroll Box. Apply/Cancel stay outside the scroll area. **Defaults** is a quiet draft action inside the reading area, alongside the other editable controls; it calls the same **Restore Defaults** node. This leaves space to edit when a save error is visible at enlarged scale. The example uses public IFSettingsSubsystem contracts, with no parallel settings save file or custom slider runtime class.

Session and controls

Get **IFSettingsSubsystem** with **Get Game Instance Subsystem**. On **On Activated**, call **Begin Edit** and retain its result in your screen's OwnsSettingsSession Boolean. When returning from a child screen, retain an existing owned session. If Begin Edit returns false, display a busy message and reject edits; another screen owns the draft.

Each slider uses the same small workflow:

Blueprint workflow (text source; interactive diagram in the web guide)

```
flowchart LR
    A[Slider: On Value Changed] --> B{Owns Settings Session?}
    B -->|Yes| C[Set Scalar Preference: field, Value, Preview=true]
    C --> D[On Preferences Updated]
    D --> E[Get Scalar Preference]
    E --> F[Slider: Set Value]
    E --> G[As Percent → value label: Set Text]
```

Fields: **UIScale**, **TextScale**, **UIVolume**, **GameVolume**, **TooltipDelay**. The first four have working controls in the example. UIScale ranges 0.75-1.5, TextScale 0.8-1.5, volumes 0-1; a keyboard/controller step is 0.05. **Set Scalar Preference** edits only that field and uses the same validation and transaction as **Set Draft**. It returns false outside an edit session or for non-finite input. Finite values clamp to the supported range. For batch edits use **Get Preferences** → **Set Members in FIFUIPreferences** → **Set Draft** instead.

The motion button calls **Get Reduced Motion** → **NOT Boolean** → **Set Reduced Motion**. Its label reflects the current draft. **On Theme Updated** applies Theme Outline/Accent through **Set Slider Bar Color / Set Slider Handle Color**; text, surfaces and buttons follow the same theme independently.

Finishing edits

Player action	Public Blueprint flow	Expected result
Apply	Apply → Branch Return Value; success clears ownership and Deactivate Widget	Save succeeds before draft becomes committed; returns to opener
Save failure	Apply Failure Reason → status Set Text	Screen stays open with its draft available
Cancel or Back	Clear ownership → Cancel → Deactivate Widget	Reverts preview to committed values; restores opener focus
Restore defaults	Restore Defaults, Preview=true	Changes draft and presentation; Apply still required to persist
Forced removal	Event Destruct; if owned, clear ownership and Cancel	Abandoned drafts do not survive a destroyed screen

Ownership is a screen convention around the subsystem's single game-instance edit session. Clearing your Boolean before Cancel prevents reentrant preference notifications from treating the screen as an active editor. UI/audio preferences are game-instance-wide; Enhanced Input key profiles remain local-player data.

Unreal's device DPI curve is applied once, with UIScale multiplying ApplicationScale. TextScale changes semantic text size separately. Reduced motion controls the existing CommonUI stack transition duration. UI/game audio use the configured distinct SoundClasses and SoundMix; **Play Sound 2D** with the example's game-class preview sound exercises that route. Configure your own **IFAudioRouting** data asset through **Set Audio Routing**.

Controls opens **WBP_AshenControls** using public **Push Screen**. See BLUEPRINT_CONTROLS.md. Key persistence remains Unreal's EnhancedInputUserSettings. Key changes have their own Apply/Cancel transaction: they are not atomically committed with UI/audio preferences. Document this distinction in your game if exposing both on one page. Graphics should use UGameUserSettings rather than this preference struct.

Connect audio and a save profile

Create an **IFAudioRouting** data asset. Assign your engine Sound Mix, one UI Sound Class and one separate game Sound Class. Use distinct sibling classes, avoiding a master/parent class whose gain also affects the other group. Assign the UI class to the theme's Focus Sound/Confirm Sound assets, and the game class to your gameplay sounds. Call **Set Audio Routing** once during game-instance/player interface setup. After travel, **Refresh Presentation** reapplies the presented preferences to the new world. Unreal still owns mixing, playback and sound-class propagation.

The saved **Test audio** button uses **On Clicked** → **Play Sound 2D** with the original IF_GamePreview sound assigned to the game class. IFButton's normal click feedback uses the theme's Confirm Sound assigned to the UI class. Moving **UI feedback** should change the confirmation without changing the game preview; **Game audio** should do the opposite. Setting one slider to0 mutes that class. Cancel/Back restores the committed gains; reopening displays those values. Replace both sound assets/classes through your own theme/routing asset.

For a named local preference profile, call **Set Storage Profile**(Slot,UserIndex) before opening the settings edit session. The node rejects empty/path-like names, negative users and changes during an active edit. It loads defaults if the selected slot has no valid supported save. Apply writes that profile through Unreal's SaveGame system. A failed write returns false and Failure Reason while retaining the draft, so keep the screen open and permit retry or Cancel. Do not display success merely because the button was clicked. Platform-specific user-index/storage behavior remains Unreal-owned.

Evidence and limits

The saved screen also includes English/Türkçe choices. Language preview, Cancel/cleanup, Apply and project Defaults use public Unreal **Get/Set Current Language**, **Get/Set Current Locale**, and **Set Current Language and Locale** nodes inside the same edit-ownership gates. Culture config is committed only after a successful UI preference Apply; these are separate file operations. Follow the language-setting recipe for the actual wiring, target generation, expected results and engine startup/save-I/O limitations.

Current settings/audio evidence records actual mixer PCM, saved Blueprint controls/actions, unique test-slot OS save failure/retry and clean-process reload. The source/runtime fixture uses normal1280×720 and enlarged800×600/150%UI+text, retains the player save bytes, and independently analyzes exported waveforms. GUI failure feedback keeps Apply/Cancel visible and at least120logical units of editing space; recorded enlarged height170.4. Defaults is accessible inside the normal scroll/navigation region.

This proves engine output signals and the tested transactions, not physical speaker/listening, controller/touch/thumb dragging or arbitrary platforms. Synthetic Slate keys and explicit slider events are identified in the evidence. Graphics remain Unreal GameUserSettings-owned; remaps have their separate EnhancedInput transaction and stock Save Settings provides no disk-I/O return. Older settings/restart/scaling results retain their recorded revision scope in RESPONSIVE_LAYOUTS.md and prior Evidence.

CONTROLS AND SETTINGS

Language settings and translated interfaces

The example's saved `WBP_AshenSettings` screen includes English and Türkçe choices. Choosing either previews the language in the existing screen. Apply keeps it; Cancel, Back, and external deactivation restore the language present when the edit began. Defaults previews English, the example project's chosen default. The same compiled translations cover the RTS and RPG menus, confirmation dialogs, settings, HUD data, and notifications.

InterfaceForge uses Unreal's Text and culture systems. The game owns its localization target and supported cultures. Language is a global engine setting, so two local players cannot independently preview different languages with these nodes.

Reproduce the Blueprint workflow

Open `Content/UI/Blueprint/WBP_AshenSettings`. Use **Get Game Instance Subsystem** to obtain **IFSettingsSubsystem**.

In its Event Graph, search for **IF Language /** to find seven labeled sections: English preview, Turkish preview, incoming culture capture, Cancel rollback, owned-deactivation rollback, successful Apply commit, and Defaults preview. The two language click chains and two rollback chains have separate positions. Each preview is **On Button Base Clicked** → **Branch (OwnsSettingsSession)** → **Set Current Language and Locale**. These are ordinary editable Blueprint nodes; the editor-only layout helper adds comments and positions without replacing the runtime workflow. The following diagram names the actual nodes rather than relying on an unverified editor screenshot.

Blueprint workflow (text source; interactive diagram in the web guide)

```
flowchart LR
    E[EnglishChoice: On Button Base Clicked] --> EB{OwnsSettingsSession}
    EB -->|True| EN[Set Current Language and Locale: en, Save=false]
    T[TurkishChoice: On Button Base Clicked] --> TB{OwnsSettingsSession}
    TB -->|True| TR[Set Current Language and Locale: tr, Save=false]
```

1. On **On Activated**, call **Begin Edit**. Only after it succeeds, capture **Get Current Language** and **Get Current Locale** into String variables. Retain the existing `OwnsSettingsSession` gate when returning from a child screen.
2. On a language button's click, branch on `OwnsSettingsSession`, then call **Set Current Language and Locale** with `en` or `tr`, **Save to Config = false**. Supported Text widgets update without replacing the widget tree. Use a culture-invariant autonym for language choices so users can recognize their language.
3. On Cancel and owned **On Deactivated** cleanup, call **Set Current Language** and **Set Current Locale** using the captured values, both with **Save to Config = false**, then the existing **Cancel** preference operation. Cleanup is gated by edit ownership.
4. On Apply, call InterfaceForge **Apply**. Branch on success. Only the success branch calls **Get Current Language** → **Set Current Language** and **Get Current Locale** → **Set Current Locale**, each with **Save to Config = true**, then clears ownership and deactivates the screen. A failed UI save retains the translated preview and visible error message.
5. On Defaults, call **Restore Defaults**, then **Set Current Language and Locale**, culture `en`, **Save to Config = false**. Cancel still restores the previously committed culture.

Blueprint workflow (text source; interactive diagram in the web guide)

```
flowchart LR
    A[Begin Edit succeeds] --> B[Capture language and locale]
    B --> C[Choose language: preview without saving]
    C --> D{Apply succeeds?}
    D -->|Yes| E[Save current language and locale to engine config]
    D -->|No| F[Keep draft and translated error visible]
    C --> G[Cancel or owned cleanup]
    G --> H[Restore captured language and locale]
```

Expected result: select Türkçe and see **Uygula** on Apply. Cancel restores the original language. Apply followed by a fresh packaged launch restores Turkish. Changing text scale still changes the same widgets; the longer RPG pause identity wraps beside its navigation cues.

The engine culture setters report whether the culture was accepted. Their Boolean result does **not** report a successful disk flush. InterfaceForge's UI preference save and Unreal's culture config are separate persistence operations. This example does not claim an atomic transaction across the two files. A platform that needs verified culture file I/O can add that policy to its own settings screen.

Translate your own screen and entries

Use Text variables for player-facing labels, tooltips, model fields, and formatted messages. Keep Text through **Format Text**, rather than converting it to String and back. Use named placeholders such as `{Count}` and preserve them in translations. Entries receive localized Text through the same public presentation/data-source contracts; they do not need a localization subclass or polling timer.

Create a project localization target in Unreal's **Localization Dashboard**. Gather both your widget packages and native runtime source, export PO files, translate, import, and compile. The example supplies the equivalent configuration in `Config/Localization/ForgeExamples_Gather.ini` and `ForgeExamples_Compile.ini`; its en/tr catalogs are in `Content/Localization/ForgeExamples`. The target name and directory name must agree with the generated resource names.

For commandlet authoring from the example directory:

```
& '<Engine>/Engine/Binaries/Win64/UnrealEditor-Cmd.exe' '<Project>/ForgeExamples.uproject' -run=GatherText '-config=Config/Localization/ForgeExamples_Gather.ini' -unattended -NullRHI
# Translate/exported PO, then import and compile:
& '<Engine>/Engine/Binaries/Win64/UnrealEditor-Cmd.exe' '<Project>/ForgeExamples.uproject' -run=GatherText '-config=Config/Localization/ForgeExamples_Compile.ini' -unattended -NullRHI
```

`Tools/translate_example_tr.py` authors the example's original Turkish catalog and rejects missing translations or altered placeholders. Buyers translate their own PO content; this tool is not a general translation service.

Register the project's target in **DefaultGame.ini** under `[Internationalization]`, using `+LocalizationPaths=%GAMEDIR%Content/Localization/<Target>`. Select supported cultures under **Project Settings** → **Packaging** → **Localizations to Package**, and include suitable **Internationalization Support**. The example stages `en` and `tr` with `InternationalizationPreset=All`. Check the packaged staging manifest for the locmeta and each culture's locres; having a PO in the editor is insufficient.

Layout, fonts, and troubleshooting

The sample uses engine Roboto and tests Turkish Latin extended glyphs. Font replacement remains a normal theme choice. Supply an appropriate composite font and fallback coverage for additional scripts. CJK, right-to-left layout, localized voice/art, and platform font rendering are not certified by these Turkish tests.

Enable **Auto Wrap Text** for long headings, give reading bodies a scroll area, and keep Apply/Cancel outside it. Do not shrink accessibility text to hide clipping. The example's pause `MenuPublisher` uses standard Text Block wrapping inside a Fill slot, while the action cue rail retains its separate automatic width.

If text stays English, check gather coverage, stable Text identities, imported translations, compiled locres, target registration, and staged cultures. If the decimal/percentage format is unexpected, check **Get Current Locale** independently of language. PIE preview alone does not prove runtime locale formatting or persistence.

UE5.6.1 editor-game startup with the active Turkish catalog reports five errors from Epic's `FStructuredLogFormatTest`. The failure is reproduced in a minimal host with InterfaceForge disabled; the earlier host without a valid Turkish catalog silently fell back to English and was not a valid Turkish control. These logs are retained and strict source-reload verification stays failed. The tested Development package skips these engine startup smoke tests and its real culture persistence/restart checks pass. The exact newer-version and final-revision results belong in the runtime evidence, not an assumed compatibility claim.

Official foundations: Managing the Active Culture at Runtime, Localization Tools, and Localization Overview.

PRESENTATION

Author screen motion in Blueprint

Open **WBP_WatchpostSurvey** and select **SurveyArrive** in its Animations panel. This is an ordinary editable UMG timeline, not a plugin-owned animation format. It animates **SurveySurface** → **Render Opacity** from0.35 at0s, through0.86 at0.12s, to1 at0.24s. The short reveal introduces the contextual card without delaying navigation or action execution. The full-screen scrim and gameplay data are separate from that animated surface. Game Master's animations/design are not included.

The same screen also contains **SurveyHeadingArrive**, targeting **SurveyHeading** → **Render Opacity**:0.82 at0s,0.94 at0.16s,1 at0.32s. This subtle independent reveal demonstrates two editable timelines with one accessibility/lifecycle policy. Their different durations are intentional. Both targets have a normal opacity baseline of1; their playback does not gate input or mutate gameplay data.

In the playable example, Begin expedition → Pause → Field journal → Watchpost survey uses this saved Blueprint. Observe ridge remains the real local adapter action; no explanatory labels were added to the interface. Duplicate the screen and change the timeline keys/target or choose your own animation variable in its graph. Select the copied class on **Push Screen**. No InterfaceForge source edits are needed.

Actual graph

1. **Event On Activated** → **Stop Animation** (SurveyArrive) → **Set Render Opacity** (SurveySurface,1) → Stop Animation(SurveyHeadingArrive) → Set Render Opacity(SurveyHeading,1) → **Branch** (ReduceSurveyMotion). False → **Play Animation**(SurveyArrive) → Play Animation(SurveyHeadingArrive). Each uses Start At Time0, Num Loops To Play1, Forward, Playback Speed1, **Restore State true**. Ordinary execution pins start both in the same frame; they do not wait for completion. True leaves both targets fully visible.
2. **Event On Preferences Updated** → **Break FIFUIPreferences** → set ReduceSurveyMotion from **Reduced Motion**. Branch on that value. True → stop/reset both named animations/targets using the same pairs. False does not replay an already visible screen. This event supplies the presented preferences, including initial settings and reversible live previews.
3. **Event On Deactivated** → stop/reset both pairs → the existing **Release Entry** on SurveyRecord. The entry unbinds its data immediately. Do not wait for decoration to finish before allowing Back. Stop the animations you own, preserving unrelated child behavior.

Blueprint workflow (text source; interactive diagram in the web guide)

```
flowchart LR
    A[On Activated] --> S[Stop own animation / opacity 1]
    S --> R{Reduced Motion?}
    R -->|No| P[Play SurveyArrive then SurveyHeadingArrive / both Restore State]
    R -->|Yes| V[Visible without motion]
    C[Presented preferences change] -->|Reduced Motion enabled| S
    D[On Deactivated] --> E[Stop / reset / Release Entry]
```

Why Restore State matters

The visible baseline is1 before playback. This animation's final state is also1. Restore State ensures an interrupted timeline returns to that baseline after the engine finishes its queued sequence update. Installed UE5.6 **FWidgetAnimationState::Stop** resets its playback cursor to zero and queues evaluation before stopping; without restoration, a same-graph Set Render Opacity can be overwritten by that zero-frame update, leaving the card dimmed. The actual runtime test exposed and repaired this issue.

The public engine Play Animation API supplies Restore State; use the setting appropriate to your own final state. The linked API reference is5.5; this recipe's exact behavior and node pins were inspected and tested in installed5.6.1. An animation intentionally ending in a different retained state needs its own completion/reset policy. This recipe does not claim that every timeline can use the survey's settings unchanged.

When gameplay is paused

Keep menu motion on normal UMG **Play Animation**, with the widget's **Tick Frequency=Auto**. Do not drive decorative UI with a gameplay timer or manually call Native Tick. Your gameplay adapter owns **Set Game Paused** and resume policy; InterfaceForge does not impose global world pause on multiplayer or other games. The small playable adapters pause their own demonstration state. A separate runtime fixture pauses the actual Unreal world and operates this same saved screen.

Expected: both timelines advance while world time is frozen; live Reduced Motion stops both at opacity1, Observe ridge remains usable, Back releases the model immediately, and reopening begins a fresh pair. Restore State and explicit resets prevent queued engine evaluations from leaving an old fade on a reopened screen. A gameplay-owned cooldown may intentionally use game time and stop during pause, independently of menu presentation.

Customization and lifecycle

Keep decorative motion independent of your gameplay adapter and stable model. CommonUI still owns screen activation, stacks, focus and Back. Its root Designer transition presets are documented in Blueprint customization. UMG timelines can complement those presets without replacing action routing.

Animate a heading, accent or image instead of the entire surface if that suits your layout. Keep opacity baseline/reset explicit. Avoid a delayed cleanup callback that might affect a later pooled activation. Stop only the animation you own; unrelated child animations have their own lifecycle. For an entry's own animation, start/update it through **On Entry Initialized** / **On Entry Refreshed** and stop/release through **On Entry Released**, since recycling does not require Destruct.

Reduced Motion should suppress nonessential motion and leave all controls readable and usable. The survey uses the real **On Preferences Updated** event rather than an independent preference variable or per-frame settings poll. It does not replay motion when the setting is disabled on an existing screen.

Verification and limits

Current paused/concurrent evidence records source/exact packaged checks on5.6.1/5.7.4/5.8.3. The fixture freezes actual world time, observes both timeline times and changing independent opacities, enables Reduced Motion during playback, uses synthetic Slate actions/Back, reopens the screen, waits for both natural completions, and removes the root during playback. Capture readback can stall rendering, so completion is based on UMG's playback state rather than a fixed wall-clock guess. Earlier animation evidence preserves the original renderer/queued-stop repairs and their narrower revision.

Current evidence is Win64 DX11 with synthetic Slate input and programmatic preference/root/pause setup. Two named timelines on independent targets are tested; arbitrary concurrent timelines, custom exit choreography, physical controllers/touch, other platforms and complete product visual polish remain separate gates. The original sample scene remains a development scene.

PRESENTATION

Author button and resource artwork in Blueprint

Customize with the illustrated Blueprint guide: exact locations, real Designer/graph images and expected results. This page supplies the detailed contract/reference.

Development revision, 2026-10-27. These public appearance features are implemented in the competitive quality pass. They do not certify that every example window has finished its visual review.

Named button skins

Open a Forge Theme data asset. In **Button Styles**, add the keys **Standard**, **Primary**, **Quiet** and **Destructive**. Each value is a **Button Skin** structure. Set its **Normal**, **Emphasized**, **Pressed**, **Selected** and **Unavailable** brushes, text colors and **Feedback Seconds**. Brushes accept your textures/materials and ordinary Slate nine-slice margins. The skin's tint is the state color; use a neutral image when you want one asset to support multiple palettes. The example's original editable sources are in ContentSource/UI/Emberglass.

On a **Forge Button / IFButton** or **IFCommandButton** choose **Button Role**. Its role automatically resolves the matching theme key. Alternatively set **Skin Name** to any other key you authored. Set **Theme Override** to restyle one control. Missing keys use the original semantic button appearance, so existing projects need no migration.

For a live Blueprint change, use **Set Button Role**, or set **Skin Name** and call **Refresh Appearance**. Expected result: the same action, CommonUI navigation and tooltip contract remains active while the art changes. Use **Set Action Availability** with a reason to block an action without removing navigation focus.

State priority is unavailable → pressed → hover/navigation focus → selected → normal. Pointer hover does not compete with controller/keyboard focus. **Get Visual State** reads the resolved state. Implement **On Visual State Changed** in your button Blueprint to animate a custom decoration. Use **On Appearance Updated** for the theme, input method, navigation focus, pointer hover and enabled information.

To create your own tree, make a Widget Blueprint derived from IFCommandButton. Add a Border named **FocusBorder**, a Text Block named **LabelWidget**, and optionally **AvailabilityWidget** and **TooltipAnchor**. Those names opt into the built-in presentation. You may omit them and use the appearance events to drive your own tree. Keep ornamental images hit-test invisible. No plugin source edit or per-variation C++ subclass is needed.

The built-in feedback interpolates tint/text using a short ease-out. Brush resources/geometry switch immediately; this is not cross-fading arbitrary materials. It never translates or resizes the hit target. Repeated state changes resume from the current displayed color. Feedback stops when settled and is removed when the widget is destructed. **Reduced Motion** in the same settings transaction snaps changes. If your Blueprint adds its own animation, honor **Get Presented Preferences** → **Reduced Motion** as well.

Resource bars using your own art

Place **Forge Progress** in your Widget Blueprint. Choose **Style = Linear**, set a useful **Desired Size**, then enable **Use Linear Brushes**. Set **Linear Fill Brush** and **Linear Track Brush**. Supply a neutral fill to tint through **Fill Color**; **Track Color** multiplies the track art. White retains the artwork's original color. Texture alpha is supported.

For project-wide changes, add two keys to the theme's **Surface Styles**, then select those in **Linear Fill Style** and **Linear Track Style**. Leave the keys empty to use local brushes. Switching to a theme without a key falls back to the local brush rather than retaining art from the previous theme.

At runtime call **Set Linear Brushes** to replace local art and clear the two style keys. This enables brush rendering; **Style** must still be Linear. **Configure Progress** can select Linear, desired size and reverse direction without replacing the widget. Call **Set Fraction** with a normalized value. Expected results:

Fraction / direction	Result
0	Track only
0.5 / normal	Left half of full-sized fill artwork
0.5 / reverse	Right half of full-sized fill artwork
1	Entire fill

The fill is drawn at its full geometry and clipped. End caps/highlights do not squeeze as health falls. Nine-slice margins preserve outer details across sizes; the fill boundary clips the image and does not invent a new end-cap at every fraction. Idle values invalidate on change instead of requiring Blueprint Tick. Engine DPI and widget transforms apply normally.

For automatic model binding, keep the control name **FractionBar** inside an **IFReadoutEntry**. It receives **Presentation Data → Set Fraction** updates through the normal entry contract. Your gameplay code owns health and authority; this control displays a value.

This art path applies only to Linear. Radial, segmented, charges and square-perimeter modes retain their procedural renderer. **IFCenterTint** and Sakura's reproduced center-origin cooldown material remain independent and unchanged. Do not replace an icon cooldown with a resource-bar skin.

Shared themes and window headers

The standalone core assets **/InterfaceForge/UIForge/Themes/DA_ForgeEmber** and **DA_ForgeMoonsteel** provide warm and cool palettes using the same small neutral art resources. Call **Get Local Player Subsystem (IFLocalPlayerSubsystem) → Set Theme** with either asset. No example game, character mesh or external font is required. The RPG's **DA_IFEmberglass / DA_IFMoonsteel** variants use the same core artwork with the example's licensed heading font. Editable originals and hashes are in **Plugins/InterfaceForge/SourceAssets/UI**.

Choose **Window**, **Header**, **Inset**, **Row**, **Combat** or **Panel** on an **InterfaceForge Surface**. These are names in **Surface Styles**, not special widget classes. A typical window is a Surface containing a Vertical Box: header Horizontal Box, flexible Scroll Box, then footer Horizontal Box. Put the heading on a Fill slot, a Close button on Auto, and set that IFCommandButton's **Request Screen Back**. Keep footer actions on Auto slots with a Fill spacer between secondary and primary actions. These layouts remain editable in the Widget Blueprint Designer.

On a theme, **Dialog Surface Style = Window** and **Dialog Header Surface Style = Header** opt the native confirmation into that family. Empty header style preserves the old text-only heading. Build an **IFDialogData**, set **Title**, **Message**, **Confirm Label**, **Cancel Label**, and call **Push Dialog** on the root. **Destructive Action** selects the Destructive button role; it does not authorize or execute gameplay. The dialog still returns exactly one result, focuses Cancel initially and supports Back. Your result handler owns the action.

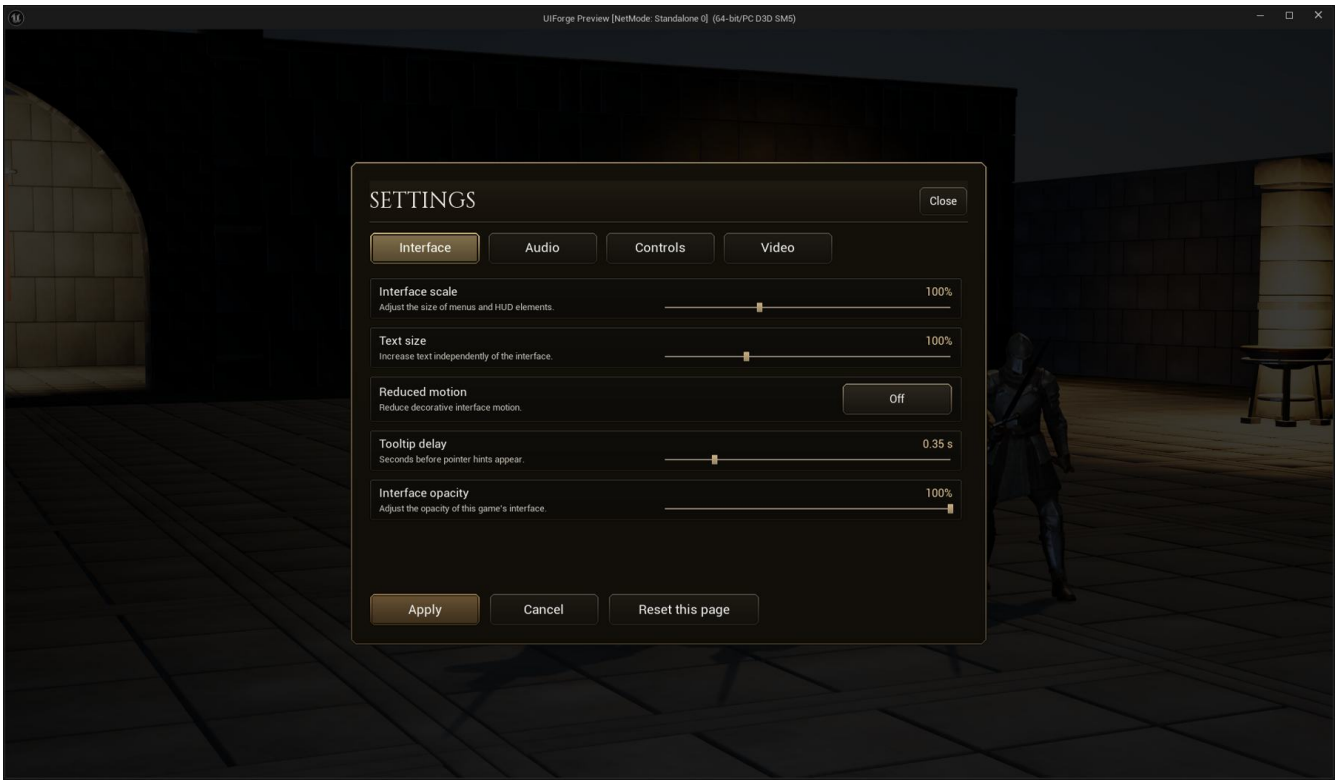
InterfaceForge Key Selector and the standard setting-row choice control follow **Button Styles → Standard**. Removing that key restores their semantic rounded fallback. Set the setting row's **Follow Theme Colors** off to retain a fully custom Slider/Combo style. Custom option generators are preserved.



The same ability window using the cool project theme

Compact settings composition

/Game/ForgeExamples/UI/RPG/WBP_IFRPGSettingRow is an ordinary Blueprint child of **IFSettingRow**, showing a different layout over the same definition/session contract. Its left column has **LabelText / HelpText**; the right has **ValueText / ValueSlider / ValueToggleButton / ValueChoice**. **StatusText** appears only when a reason is present. No fixed row height clips wrapped help or enlarged text. Copy this layout into your own Blueprint, set each instance's **Definition**, and place rows on the pages of an **IFSettingsMenu**. Built-in bindings and custom providers share Apply/Cancel/Defaults. See Blueprint settings composition for the complete transaction and integration recipe.



Current compact Designer-authored settings layout

Choose a primary menu action in Blueprint

The RPG and RTS main menus implement **Event Initialize Screen → Parent: Initialize Screen → Get UIModel (Continue) → Cast to IFPresentationData → Branch (Enabled)**. True calls **Set Button Role (Primary)** on Continue and **Set Button Role (Quiet)** on New. False assigns the reverse. The game adapter supplies save availability; the UI does not read or create save files. The existing **Initial Focus Name = Choice0 / Fallback Focus Name = Choice1** chooses a usable navigation target. This graph reruns for each supplied screen context, including a pooled menu returning from gameplay.

The two branches are runtime-tested using temporary availability on the same stable model, restored within the test call; no save is removed to manufacture the no-save case.

Verification record

The first appearance runtime checks cover role skin resolution, unavailable/selected states, interrupted finite feedback, reduced motion, missing-key fallback and idle completion. Native main/confirmation views were visually inspected. Resource rendering now passes zero/half/reverse/full and local theme-key fallback, including actual captured pixel checks. Native standalone menu/confirmation/cast/damage/pause/settings also passes. Current13-check Blueprint composition/source proof, measured responsive/enlarged layouts, and27-observation native cooked-package interaction/restart pass are recorded in the active milestone ledger. No physical controller/touch claim is made.

PRESENTATION

Responsive Blueprint layouts

InterfaceForge keeps Unreal's DPI curve and applies one user UIScale through the settings subsystem. TextScale changes semantic type independently. These settings cannot repair an unconstrained Designer tree: choose sensible bounds, wrapping and scrolling for each screen. The root uses Unreal SafeZone around the HUD/Menu/Modal layers.

Current expanded-text revision retains full FText in bounded reading regions. The RTS custom prompt now gives its caption an editable Size Box/InterfaceForge Scroll Box limit instead of allowing a long label to consume the whole resource pane. The RPG context entry accepts an independent longer-title model through Initialize Entry. Tests use expanded Latin text; they do not establish full localization, RTL/shaping/font coverage or touch-device acceptance. See prompt recipe and Evidence/EXPANDED_TEXT_RUNTIME.md. No global text shrink/truncation policy is imposed by the core.

The original main/pause samples use ordinary UMG widgets: a full-stretch Canvas slot with 24 units of margin contains **Scale Box**, Stretch = Scale To Fit, Stretch Direction = Down Only. Its **Size Box** supplies preferred bounds of500×480 for Main and460×420 for Pause. The Scale Box's child slot aligns Main to the right and Pause to the center. The InterfaceForge Surface has24-unit padding around MenuColumn. This fit policy can reduce the card below requested user scale when space is insufficient; it is a sample choice, not a global accessibility guarantee. Replace it in Blueprint if your product requires reflow instead.

The publisher (IFText Base Size10) stays outside scrolling. MenuTitleBounds is100 units tall on Main,64 on Pause; its inner Scale Box is Down Only and aligns the unwrapped identity left/center. Title Base Size is28/22; explicit line breaks are retained. This bounded decorative identity prevents a large title from pushing actions off-screen. For long localized game titles, consider a different line break, more title space or a separate arrangement, and verify the result instead of relying on automatic shrinking.

Main's introduction uses IFText Base Size12 with Auto Wrap Text, in a64-unit MenuDetailBounds and independently clipped **MenuReading** Scroll Box. **MenuActions** occupies the remaining vertical Fill space and contains the original public buttons, with8-unit gaps and no trailing gap. Set **Scroll When Focus Changes = Instant Scroll**, **Navigation Destination = Center** and Clip To Bounds Always. Restored/navigated focus then reveals an action if customization makes scrolling necessary. Do not put identity, prose and all actions in one scroll region: the previous tree hid most actions and clipped the publisher at maximum scale. The current recorded English strings fit completely at1280×720/defaults and800×600/150% UI and text; arbitrary translations and minimum touch targets still require verification. Actual interaction and visual evidence.

The journal uses the same fitting wrapper, but keeps a bounded **List View** in its vertical Fill slot. Avoid wrapping a virtualized list in an unconstrained outer Scroll Box. It can defeat virtualization and grow the entire collection.

Settings/Controls use a stretch surface, a vertical column with fixed header, a Fill-sized Scroll Box body, status and a **Wrap Box** footer. Set Inner Slot Padding = (12,8) and disable Explicit Wrap Size to follow the actual allotted width. Apply/Cancel/Restore Defaults wrap into additional rows when necessary. The body gives up space before these exit actions. Do not put a fixed horizontal button row on a narrow page: the live 800×600/150% scale test exposed clipped defaults before this repair.

Themes and IFText/IFButton change fonts/colors without copying these trees. Set Auto Wrap Text on prose and verify long localized labels. Use the containing screen's input-presentation event to choose alternate arrangements when appropriate; retain the same commands, focus and model contracts. Slate keyboard navigation still traverses the actual controls.

The RTS HUD uses an original viewport-relative dock, rather than two independently anchored fixed-width cards. Its header Canvas slot stretches from (0,0) to (1,0.28), with margins (24,24,24,0). The bottom dock stretches from (0,0.46) to (1,1), with margins (24,0,24,24). Inside a Horizontal Box the resource and command cards use Fill weights 0.55/0.45 and a 16-unit gap. Clear their old Width Overrides. Their existing readouts/buttons remain intact; wrap each card's column in a bounded Scroll Box with Instant Scroll on focus change. The header pairs a 110-unit preferred-height objective card with a top-aligned Pause button. These are standard Designer controls, not a private example layout class.

At normal 800×600 scale all four orders, production and prompt fit. At maximum UI/text scale, controls and longer status messages scroll in separate panels; keyboard focus reveals orders. Navigation Destination = Center leaves space around a focused command when its context gains a line. If your data can change by more than this allowance, use **Scroll Widget Into View** after the update or choose a stable context region. Do not assume a focus-change setting covers every later data mutation.

Responsive visual checks are recorded in Evidence/RESPONSIVE_CARDS.md. Live 800×600 review passed settings footer visibility/Apply at 150% UI scale plus 150% text and mouse scrolling to all Main actions. Clean restart retained the enlarged preferences; keyboard Down revealed Settings automatically and Enter opened it. Restore Defaults remained operable. Other resolutions, HUD reflow at extremes, dialog long text, minimum touch targets and safe areas on actual devices remain acceptance gates. Engine-safe-area support alone is not a tested mobile claim.

The RPG's two header cards use a different ordinary UMG arrangement: **Canvas → Overlay (PartyHeaderFit / QuestHeaderFit) → InterfaceForge Surface → clipped Scroll Box → Vertical Box → existing entries**. Keep the former surface's Canvas anchors and offsets on the Overlay (.36 bottom anchor); set the Surface's Overlay slot to Horizontal **Fill**, Vertical **Top**, and padding14. The enclosing dock provides a maximum available region while a short surface follows its content. Avoid a Fill vertical surface here: it reserves the entire region even when there are only two vitality lines or one objective. No new widget subclass or per-frame geometry calculation is needed.

At actual1280×720/defaults, party and quest surfaces measure113 and91 layout units high inside235.2-unit docks. At actual800×600/150% UI and text, they measure130.6 and137.4 inside148.8-unit docks. These are root layout units, not physical pixels. Source and fresh cooked checks verify complete initial English titles, quest detail and party bars within the surfaces and viewport. Header evidence includes rendered captures and interaction scope. Longer or later-state objectives can still require the retained scrolling; this check does not guarantee every translation or certify the rest of the HUD's layout.

The original RPG bottom action dock uses **Canvas → ActionDockFit (Overlay) → ActionDockColumn (Vertical Box)**. Keep anchors(0,.54)→(.65,1) and margins(24,0,12,24) on the Overlay. Set the column's Overlay slot Fill/Bottom. Pin StatusReadout in an Auto first slot, put the existing clipped Scroll Box in a Fill second slot, and keep its ActionColumn/Wrap Box/cards. A short dock follows its content; longer wrapped cards remain bounded. Cards are160wide/Min Desired Height168/no fixed height. This releases unused scene area without shrinking the original80×80 icon or changing actions.

Enable the shared readout's **Inline Detail** only for the short status instance, and set its two IFText Base Size values to12 followed by Refresh Appearance. Readout recipe documents the ordinary Stack Box/Pre Construct/slot/wrapping graph and default stacked behavior. An inline title has no automatic wrapping; use stacked mode for longer prose. Enlarged recovering Ward now fits its whole icon/tint/countdown, reason and pinned status. Approach's longer reason still grows a217.2-unit card and reveals through scrolling; no promise that every part of a tall card remains visible simultaneously. Current action evidence records strict source/cooked bounds and saved directional-navigation checks. Tooltip overlap and general scene/HUD polish remain separate work.

The latest HUD composition reserves a content-fitted navigation description within a .18-height scene lane. Both genres share the same IFEntry body and public screen host registration; pointer uses the normal popup. Result messages have reading priority. Blueprint hint hosts describes the ordinary Fill/Center Overlay/Surface arrangement and current width/text limits.

The RTS passive card follows its content at the bottom of the original .55-weight dock: **Horizontal Box → RTSPassiveFit (Overlay) → Size Box → Surface → RTSOverviewColumn**. The column gives its resource/notice Scroll Box a Fill slot, then pins QueueReadout and IFActionPrompt in Auto slots. Keep14-unit Surface padding, Queue Base Size12 and ordinary Refresh Appearance. Commands use6-unit gaps and no trailing gap. Default-size labels fit; enlarged controls/prose still scroll. Production and the resolved action prompt remain visible. These modifications preserve data/action graphs and add no gameplay system.

PRESENTATION

Safe areas and responsive Blueprint layouts

Use **Create Root** with **IFRootLayout** for the default root. Its engine **Safe Zone** wraps HUD, Menu and Modal stacks and the notification host. Unreal supplies the safe margins; InterfaceForge keeps screen ownership, focus and navigation in CommonUI. UI scaling and text scaling remain separate user preferences.

For a Designer-authored IFRootLayout subclass, put a **Safe Zone** at the root, with an **Overlay** containing your CommonUI stacks. A custom Designer tree replaces the native tree; it must include its own Safe Zone. `WBP_ForgeFlowRoot` demonstrates this. **Event On Initialized** → **Register Layer** registers HUD/Menu/Modal stacks; see Blueprint customization. Keep Pad Left/Right/Top/Bottom enabled for edges you protect; leave the content slot's Safe Area Scale at1unless your platform design requires otherwise. The same public engine properties remain editable in Blueprint.

Anchor HUD docks within the safe content. Use **Scale Box** (Scale To Fit / Down Only) for bounded menu cards, **Scroll Box** for long prose, and separate pinned Apply/Cancel/Interact/Back controls from scrolling text. Reusable semantic text and theme components retain the user's enlarged text; resolve clipping through layout before reducing text size.

The original RPG context panel uses a Canvas slot with minimum anchor(.58,.48), maximum(1,1). The neighboring ActionDockFit ends atX.58. These Designer settings provide more room for the interaction heading under enlarged text/insets, without replacing its entry, fonts, input prompt or graphs. Change those anchors/layout classes normally in your own Blueprint. The action Wrap Box can form a scrollable column at narrow/enlarged sizes; focused actions scroll into view. Ward remains operable, including recovery/countdown and the same center tint. RPG composition.

Decorative full-screen art or a backdrop can be outside the Safe Zone in a custom root, while its interactive stacks remain inside. Safe Zone is a layout constraint, not a replacement for localization, font coverage, overflow handling or accessibility testing. Long descriptions may scroll; keep key action labels and initial-focus controls readable.

Safe-area runtime evidence identifies actual engines, viewport sizes, source/package interactions, simulated inset profiles and limits. Windowed Win64 simulation is not physical mobile/console notch certification. Physical controller/touch checks remain deferred. The internal QA preview overrides are test setup, not required Blueprint authoring or project settings.

PRESENTATION

Theme-aware reading panels

Use **InterfaceForge Scroll Box** in the Widget Blueprint palette for menus, settings and HUD reading regions. It inherits ordinary UMG ScrollBox: your children, slots, orientation, focus targets, wheel policy and scroll events remain normal Unreal workflows. Its scrollbar adopts the owning local player's theme automatically; it adds no input detector or per-frame polling.

Quick start

1. Place **InterfaceForge Scroll Box** in a constrained panel, such as a Fill slot in a Vertical Box. Put your content inside it. Give the viewport finite space; content that expands its parent cannot overflow.
2. Set **Scroll When Focus Changes** = **Instant Scroll** and an appropriate **Navigation Destination**, such as Center, for a list of focusable buttons. For prose, enable **Is Focusable** if players should focus the reading region. Preserve your own explicit navigation targets.
3. Call **Set Theme** on **IFLocalPlayerSubsystem**. Existing InterfaceForge Scroll Boxes update along with the rest of the interface. No reconstruction or copied widget tree is needed.

The theme uses **Muted Text** for the normal thumb, **Accent** for hover/drag and a subdued **Outline** track. **Scroll Bar Thickness** defaults to6Slate units and clamps to4-32 at runtime. The viewport's DPI and UI scale apply once; text enlargement does not separately multiply this thickness. Raise it for your intended presentation/accessibility needs. The styled box removes the default edge shadows so enlarged prose stays clear; the scrollbar indicates overflow.

Local overrides and standard ScrollBoxes

Assign **Theme Override** on one InterfaceForge Scroll Box, then call **Refresh Appearance** after a runtime edit. That panel retains its local colors/thickness when the global theme changes. Clear the override and refresh to return to the current player theme. Retain a reference to any theme you create dynamically, as with other UObject assets.

An existing ordinary **ScrollBox** can instead call **Apply Theme to Scroll Box** with that box and your **IFTheme**. This is a one-shot visual operation; repeat it from your **On Theme Changed** binding for live updates. It does not change children, offsets, scrollbar visibility, orientation, wheel/touch policy or navigation. It changes bar colors/thickness and clears edge-shadow brushes. Use ordinary ScrollBox and edit its Designer **Style / Bar Style** properties when you want complete custom brushes/shadows; automatic InterfaceForge styling intentionally reapplies its tokens.

Blueprint workflow (text source; interactive diagram in the web guide)

```
flowchart LR
    A[Standard ScrollBox reference] --> S[Apply Theme to Scroll Box]
    T[IFLocalPlayerSubsystem → Get Theme] --> S
    E[Your On Theme Changed binding] --> S
```

Use Unreal's **Scroll Widget Into View**, **Scroll to Start**, **Scroll to End**, **Set Scroll Offset** and **On User Scrolled** as usual. For gamepad-focused prose, configure the inherited **Analog Mouse Wheel Key** according to your project's CommonUI policy. Touch behavior remains Unreal's ScrollBox behavior; target-device validation is still required. The style does not hide the mouse cursor or decide which input family is active.

ScrollBox is intended for bounded collections and does not virtualize; use the journal's ListView/IFEntry lifecycle for large collections. Epic's ScrollBox API documents its existing navigation, scrolling and style controls. Avoid constructing thousands of rows in a ScrollBox.

Examples and verification scope

The original main/pause/settings/controls/dialog and RTS/RPG HUD Blueprint trees use this public component. Their existing data/action graphs and children remain in place. The internal own-project conversion helper is not part of the normal developer workflow: use the palette or public styling node in your project. Reference projects are untouched.

Offscreen Win64 DX11 checks exercise actual HUD wheel scrolling, live global theme colors/thickness, local override across a theme broadcast, existing modal/device/remap flows and default/enlarged layout. A temporary long-prose fixture creates measurable overflow and is removed before returning to the menu; it is not saved into the playable UI. Physical controller/touch/thumb dragging and arbitrary localization/aspect ratios remain separate gates. The actual outcomes and archive revision are recorded in Evidence/SCROLL_THEME_RUNTIME.md.

PRESENTATION

Blueprint readout layout choices

Use the saved example **WBP_HUDReadout** when the same Title/Detail model needs either stacked prose or a compact status line. It remains an **InterfaceForge Entry**, using **Initialize Entry**, **On Entry Refreshed** and **On Entry Released**. No plugin source change or separate subclass is needed for this choice. This is an editable example Blueprint, not a required native plugin widget.

Use the option

1. Place WBP_HUDReadout in your screen in Designer.
2. Select that instance. Under **Presentation**, leave **Inline Detail** off for a quest or longer contextual description. Turn it on for a short status caption.
3. Initialize it with your presentation model through **Initialize Entry**. Its existing refreshed graph reads Title and Detail; your data source remains responsible for those values.
4. Size the enclosing region for your content. Test your largest UI/text settings and translated strings. Expected result: the same entry class renders two stacked text regions by default, or Title followed by Detail in one row when enabled.

For a longer title or description, prefer the stacked layout in a bounded **InterfaceForge Scroll Box**. Keep essential actions outside that reading region. A title need not shrink or lose its FText to remain reachable; preserve the full text and let the region scroll. Scrolling recipe describes the normal engine controls.

When demonstrating another data source, create your own presentation object in Blueprint, set its Title/Detail, then call **Initialize Entry** on the saved entry. Restore/rebind the original object when the demonstration ends. Avoid overwriting fields that an active gameplay adapter owns and updates. The expanded RPG fixture uses an independent object through this same public contract; no gameplay rules or plugin source edits are needed.

The RPG **StatusReadout** uses Inline Detail. Its screen's **Event Construct** sets both **Readout Title** and **Readout Detail** → **Base Size = 12** → **Refresh Appearance**. The quest and RTS instances keep the default stacked layout and their existing sizes. Theme and text-size updates still use the configured Base Size.

Actual saved graph

InlineDetail is an instance-editable Boolean with default false. **Event Pre Construct** → **Branch (InlineDetail)** chooses the following paths:

Property/node	False: stacked	True: inline
ReadoutFlow → Set Orientation	Vertical	Horizontal
ReadoutTitle → Set Auto Wrap Text	True	False
Title → Slot as Stack Box Slot → Set Size	Auto	Auto
Title slot → Set Padding	Bottom 6	Right 12
Detail → Slot as Stack Box Slot → Set Size	Auto	Fill, weight 1
Detail slot → Set Padding	Bottom 8	0

The root **ReadoutFlow** is a normal **Stack Box**. Set Orientation, Set Padding and Set Size are Blueprint property setters on the Stack Box/Stack Box Slot. Detail retains automatic wrapping. Existing initialization, data refresh and cleanup graphs are unchanged by this layout choice.

Blueprint workflow (text source; interactive diagram in the web guide)

```
flowchart LR
```

```
A[Event Pre Construct] --> B{InlineDetail}
B -->|False| C[Vertical: title above detail]
B -->|True| D[Horizontal: title beside flexible detail]
E[On Entry Refreshed] --> F[Read model Title and Detail]
F --> G[Set Text on the same two widgets]
```

The option is applied in Pre Construct. Configure the Designer instance before it is constructed. There is no new automatic runtime toggle/reflow event; mutating the Boolean alone after construction is not a documented live-update workflow. You can extend this example's graph with a reusable refresh event if your game needs one.

Inline titles deliberately do not wrap. A long title can consume the row; use stacked mode for prose or provide more width. This choice prevents cached title wrapping from accidentally producing a second status line and clipping the cooldown icon below it at the tested enlarged settings. It is not a guarantee for arbitrary translations.

Responsive RPG use

Canvas → ActionDockFit (Overlay) → ActionDockColumn (Vertical Box) contains the pinned StatusReadout in an Auto slot, followed by the action Scroll Box in a Fill slot. The column's Overlay slot is Fill/Bottom. The old ActionColumn and Wrap Box remain inside the scroll. Status updates therefore remain visible while navigation scrolls the action cards.

The original action entries use Width Override 160 and Min Desired Height 168, with no Height Override. Longer feedback grows the card. The existing 80×80 icon, 62×62 center tint and countdown layers remain intact. **Navigation Destination = Bottom or Right**, **Instant Scroll** and the existing Blueprint reveal graph keep longer reasons reachable. At enlarged settings a tall Approach card may scroll its upper icon away while its complete reason is revealed; the focused recovering Ward card's whole icon, tint and countdown fit together.

Runtime record and RPG recipe cover the tested saved graph and limitations. The current shared stacked readout also passes the packaged RTS command/selection/production regression. Physical devices, long localization, broader tooltip placement and newer-engine sample revisions remain unverified here.

PRESENTATION

Blueprint contextual tooltips

Tooltips present short contextual information without moving CommonUI focus away from an action. The standalone RTS and RPG action entries pass their own presentation models to a saved custom tooltip. All classes and nodes below are public InterfaceForge features; command execution stays in the example adapters.

Add a tooltip to your own button

1. Create a Widget Blueprint with parent **IFButton**. Put **InterfaceForge Tooltip Anchor** at the root of its Designer tree and name it **TooltipAnchor**. Place your existing button content inside the anchor. Optional **FocusBorder**, **LabelWidget**, and **AvailabilityWidget** names keep the normal button presentation contracts.
2. Set the anchor's **Tooltip Entry Class** to your tooltip Widget Blueprint. It must implement **IFEntry**. You can leave the native **IFTooltipCard** default for a title, description and unavailable reason. **Placement** defaults to Menu Left; the engine also provides above/below/right placements. **Preferred Width** is in player UI units, before the engine's DPI/user scale.
3. Create or receive an **IFPresentationData** object. Set **Title**, **Detail**, **bEnabled** and **Unavailable Reason**, then call **Set Tooltip Data** on the button. Its normal hover/focus callbacks request or hide the tooltip through the owning local player.

Expected result: meaningful pointer hover or keyboard/controller navigation focus opens the tooltip after the UI preference's **Tooltip Delay** (default 0.35 seconds). A parked mouse pointer does not compete with navigation focus. Unavailable actions remain focusable and can explain their restriction. No gameplay action is executed by opening a tooltip.

The native IFButton already contains an anchor. Existing custom button trees need the named component; Set Tooltip Data safely does nothing if it is absent. Clearing data with **Set Tooltip Data** → **None** closes the tooltip and drops the button's data reference.

Author the content entirely in Blueprint

WBP_ForgeTooltip in `/Game/UI/Blueprint` has parent **IFEntryWidget**. Its Designer uses an **InterfaceForge Surface** and three **InterfaceForge Text** widgets: HintTitle (Accent, 13), HintDetail (Body, 11), HintReason (Caption, 10). These follow the current local-player theme and text-size preferences.

Its real graph is:

Blueprint workflow (text source; interactive diagram in the web guide)

```
flowchart LR
  R[On Entry Refreshed] --> C[Cast Entry Data to IFPresentationData]
  C --> T[Set Text: HintTitle from Title]
  T --> D[Set Text: HintDetail from Detail]
  D --> U[Set Text: HintReason from Unavailable Reason]
  E[On Entry Released] --> X[Set all three texts empty]
```

Change your model's fields together and call **Notify Data Changed** once. IFEntryWidget refreshes its subscribed content without rebuilding the button or tooltip. Call **Set Tooltip Data** to replace the entire source object. For your own data type, implement IFEntry directly or use IFEntryWidget and bind/unbind your events in **On Entry Initialized / On Entry Released**. A custom UObject is supported; generic-model subscription is only a convenience.

The tooltip is passive and Hit Test Invisible. Keep its body concise. Put long documents, interactive actions or scrollable details in a contextual screen/dialog; tooltip content does not take keyboard focus or accept scrolling. Its width is bounded to the lesser of Preferred Width (120-600) and 60% of the current player's root width. The engine fits the resulting popup to the window. This is not a guarantee that arbitrarily long prose or a custom fixed-size tree will fit vertically; test your actual translated content and use a panel when needed.

Plain custom widgets and touch gestures

A plain Widget Blueprint can wrap its content in **InterfaceForge Tooltip Anchor** without deriving from IFButton. Use the anchor's **Set Tooltip Data**, **Request Tooltip** and **Hide Tooltip** nodes from your own hover, navigation or gesture events. **Request Tooltip** → **Use Delay false** supports an explicit touch long-press or another deliberate gesture. InterfaceForge does not implement gesture recognition or consume touch presses on your behalf. Automatic touch activation and physical touch-device behavior remain unverified.

Use these InterfaceForge request/hide nodes instead of the inherited low-level Menu Anchor Open/Toggle Open functions so the local-player delay and competing-tooltip policy run. Request Tooltip returns false for missing data, invalid/abstract/non-entry classes, another player's anchor, a missing root/top screen, or a screen transition/covered screen.

Lifetime, screens and customization

Create the owning player's root using **Get Local Player Subsystem (IFLocalPlayerSubsystem)** → **Create Root**. A custom IFRootLayout Blueprint registers its HUD/Menu/Modal CommonUI stacks through **Register Layer**. The inherited root tick advances one tooltip delay per player even when the game world is paused. No timer or population scan is added to each entry.

Only one pending/open tooltip exists per local player. Switching sources closes the old popup and calls **Release Entry** before creating the new content. Modal pushes, screen transitions, layer clears and root removal close active/pending tooltips. Covered IFScreen content cannot reopen a tooltip. Custom widgets with no enclosing IFScreen are scoped to the current top screen when requested; hide them when your own context ends.

Set **Tooltip Entry Class** in the Designer to replace the whole body. Change IFText roles/Base Size, surface tokens, spacing and the local-player theme to customize it. The anchor adds a width bound around the entry tree, uses the engine's Menu Anchor placement and opens in the existing player window without taking focus. It does not copy Game Master's visual design or use reference assets.

Source and cooked runtime evidence, exact engine/device scope, captures and retained failed checks are recorded in `Evidence/TOOLTIP_RUNTIME.md`. Physical controller/touch, split-screen and localized long-text acceptance are still separate gates.

Engine foundation

Epic documents Menu Anchor's placement, Fit in Window, content creation callbacks and application-stack lifetime options in Using Menu Anchors. InterfaceForge uses that engine placement/lifetime foundation with a passive current-window popup and a local-player policy. It avoids continuously positioning an independent canvas from potentially stale Get Cached Geometry values. Cached geometry in the runtime fixture measures the previous rendered frame for verification; it is not the placement implementation.

Input-aware placement

The anchor's **Use Navigation Placement** defaults off. Enable it and choose **Navigation Placement** for keyboard/controller focus; normal **Placement** remains the pointer/touch choice. **Get Resolved Placement** reads the current local-player intent. After changing options while open, call **Refresh Tooltip Placement**; this preserves the entry and source focus. IFButton's existing input callbacks also refresh an active request. This adds no widget-specific device detector.

The samples choose **Above** for pointer and **Centered Above** for navigation, Preferred Width400. Their saved tooltip uses padding10,4-unit gaps, **Wrap Text At350** and Auto Wrap Text; the final Set Text of HintReason branches through **Text Is Empty** → **Set Visibility** (Collapsed / Hit Test Invisible). Tune those properties for your own width and translations. These values were verified at two sizes, not every layout.

InterfaceForge buttons now default **Apply Alpha On Disable** off. Semantic unavailable colors still apply, CommonUI still blocks activation and preserves navigation, and the tooltip avoids a second inherited0.5fade. Existing projects may explicitly retain their chosen CommonUI property. Keep it off for readable tooltip/cooldown layers.

Current placement evidence covers18source/cooked results and strict independent core builds on5.6/5.7/5.8. Passive readout/notification overlap, physical devices and broad polish remain open.

Reserved navigation descriptions

An IFScreen can now call **Register Navigation Hint Host** for an empty owned Content Widget and any IFEntry body. With **Use Navigation Hint Host** enabled on the anchor, keyboard/controller focus uses that body while pointer hover retains its popup. **Is Tooltip Shown** includes both; **Is Open** remains popup-only. Navigation hint recipe documents actual Designer/entry/registration nodes, result reading priority and lifetime. The saved RTS/RPG examples use this to keep focus descriptions out of their essential HUD panels.

PRESENTATION

Blueprint result notifications

A short notification draws attention to a completed action without taking focus from the player. InterfaceForge supplies a bounded local-player queue and an ordinary replaceable `IFEntry` body. Your gameplay system decides when an action succeeds. The UI queue does not execute or validate gameplay commands.

The queue uses Unreal's local-player subsystem lifetime, matching its owning player rather than a single global HUD. See Epic's ULocalPlayerSubsystem API and subsystem Blueprint access. InterfaceForge's queue and reading-time behavior below are its own public contracts, verified against the installed engines.

The Outpost example posts **Road secured / 40 supplies recovered.** only after its patrol completes. Emberwatch posts **Party restored / Ember Ward is active.** after Ward succeeds and **Watch restored** when Kindle reaches the hearth. These are brief results; the persistent objective, resource and status models remain available in the HUD.

First working result

1. Create the owning player's root with **Get Local Player Subsystem** → `IFLocalPlayerSubsystem` → **Create Root**. Use `IFRootLayout` or a Widget Blueprint derived from it with an empty Designer tree to retain the native layers.
2. From the same PlayerController, use **Get Local Player Subsystem** and select `IFNotificationSubsystem`.
3. On your successful-action event, call **Post Notification**, supplying **Title** and **Message**. The return value is a notification GUID; save it if the action needs to dismiss or update this result.
4. Play. The default passive card appears above the HUD/menu and below dialogs, then disappears after four seconds of visible UI time. It does not move focus, accept clicks or create a separate window.

Blueprint workflow (text source; interactive diagram in the web guide)

```
flowchart LR
    A[Your action succeeds] --> B[Get Local Player Subsystem: IFNotificationSubsystem]
    B --> C[Post Notification]
    C --> D[GUID for dismissal]
    C --> E[Public root presents IFEntry body]
```

Expand the advanced pins for **Duration**, **Key**, **Scope**, and **Entry Class**. Duration must be finite and positive; accepted durations are clamped to 0.25-60 seconds. Both empty texts, missing custom data, an invalid owner or an explicit class that does not implement `IFEntry` return an invalid GUID. Do not treat an invalid GUID as a successfully queued message.

Replace the appearance in Blueprint

Create a Widget Blueprint based on **IFEntryWidget**. Add your own Designer tree using **IFSurface**, **IFText**, images or other UMG controls. The example's saved `WBP_ForgeNotification` uses an accent title and body text in a compact horizontal ribbon.

In **On Entry Refreshed**:

`Entry Data` → **Cast To IFPresentationData** → `Title` / `Detail` → **SetText** on your title and message controls.

In **On Entry Released**, clear presentation text and stop any timers or animation work you started. `IFEntryWidget` binds `Notify Data Changed` and releases its old model subscription through the same lifecycle used by journal and command entries. A later **Notify Data Changed** on the live model invokes refresh without creating another notification.

Choose your body class in your root Blueprint's **Default Notification Entry Class**, or override **Entry Class** on an individual post. The saved `WBP_ForgeRoot` and `WBP_EmberRoot` use the same body with different **Notification Dock** values. Their owning example controllers expose **Root Class**. No plugin source edit is needed to replace either root or the body.

To change the live layer's **Notification Entry Class**, call **Refresh Notification Presentation** afterward. It releases the current body and reconstructs it on the next UI tick with the same model, GUID and remaining reading time. **Register**

Entry Host performs this refresh automatically when the layer is already registered. A per-post Entry Class override continues to take precedence over the layer default.

For custom presentation data, call **Post Notification Data** with your UObject and an entry class that understands it. An ordinary Widget Blueprint may implement **IFEntry** directly instead of inheriting `IFEntryWidget`; it must implement **Initialize Entry** and **Release Entry**, including its own binding cleanup. The built-in card expects `IFPresentationData`, so select your compatible class when posting another model type.

Keep the body passive. Put interactive choices in a screen or dialog, where CommonUI can route accept/back and restore focus. Do not put essential actions inside a notification that intentionally ignores hit tests.

Updates, capacity and owner lifetime

Post Notification with a nonempty **Key** and the same **Scope** coalesces an existing record: it keeps that GUID and queue position, replaces its data/entry and resets reading time. `Scope=None` matches only another unscoped record. Empty keys always create distinct records. The old visible body receives **Release Entry** before replacement; coalescing does not emit **On Notification Ended**.

The default capacity is eight, including the visible record. **Set Queue Capacity** clamps to 1-32 and trims pending records. On overflow, the visible result is preserved and the oldest pending one ends with **Overflow**. At capacity one, a new record cannot displace an already visible result and its post returns an invalid GUID. Before any record is presented, the oldest queued one may be discarded. This finite queue is for attention, not a persistent event history.

Use a gameplay adapter, quest or screen object as **Scope**. The scope reference is weak and does not retain that owner. The queue retains its **Data** until the record ends; custom models should use weak gameplay references when needed rather than retain actors through the model. Call **Dismiss Scope** when an expedition or interaction ends even if its provider object remains alive. The examples do this on return to the main menu. Other owners' notifications survive that scope cleanup.

Dismiss Notification returns true only when its GUID was present. Subscribe to **On Notification Ended** for **Expired**, **Dismissed**, **Scope Ended**, **Overflow**, **Invalid Entry**, or **Root Removed**. Records are removed before the event; dismissing the same GUID again returns false. **Clear Notifications** ends all this player's records with **Root Removed** and rejects posts made from those end callbacks, preventing a flow cleanup from creating another stale message. The same guard applies during capacity trimming. An overflow callback during an ordinary post may fill the freed slot; the incoming post then returns invalid instead of recursively evicting callback-created records.

Reading time, dialogs and motion

The public root drives notification time through its UI tick, including while the game world is paused. A modal dialog or an active screen-stack transition collapses the notification layer and freezes visible reading time. Pending records do not count down before presentation. Weak-scope cleanup still runs while a modal suppresses display. If there is no registered presenter, records wait rather than silently expire.

The default fade uses the current theme's **Transition Seconds**, capped to one second. The presented **Reduced Motion** preference removes that fade. **Get Remaining Seconds**, **Get Active Notification Id**, **Get Notification Data**, and **Get Notification Count** expose the current state for your own extensions. Duration is not derived from world gameplay time.

Positioning and custom roots

Notification Dock is a normalized anchor within the root's safe area; the native layer aligns its body to that point horizontally centered and vertically downward. **Notification Width** is clamped to 120-600 layout units and at most 80% of the allocated player width. These defaults are copied into the layer during root construction. To change the live presenter, use **Get Notification Layer** and set its **Dock** or **Preferred Width**. Inspect the result at your largest UI/text preferences.

Reserve a lane in your own HUD rather than covering critical information. Outpost uses dock Y=0.28 and begins its lower panels at Y=0.46. Emberwatch uses Y=0.36 and begins its action/context areas at Y=0.54. Both root assets use width 600. A complete quest playthrough exposed a wrapped result overlapping the previous context position at Y=0.48 at maximum text/UI scale; the saved context anchor now leaves additional clearance. The current clearance verification is recorded in the

acceptance record. Arbitrarily long or localized messages need your own wrapping/layout policy; use a contextual screen or journal for long explanations. A viewport bounds check cannot certify that an overlay leaves other information readable.

A root with a custom Designer tree owns its layout. Add an **IFNotificationLayer** above HUD/menu content and below the modal stack, then call **Register Notification Layer** on the **IFRootLayout**, passing that layer. Its owning player must match the root. The root retains the presenter, handles modal/transition suppression and clears/unregisters it when removed. Passing None explicitly unregisters it; a removed root rejects re-registration. Do not bypass the root with subsystem-only registration when relying on its cleanup. The saved WBP_ForgeFlowRoot demonstrates this actual public Blueprint graph; see Blueprint customization.

In a layer with a custom Designer tree, add a content container and call **Register Entry Host**; alternatively name the container `EntryHost`. An optional SizeBox named `NoticeBounds` receives the native width/dock behavior when it is a Canvas child. Otherwise your Designer tree owns sizing/placement.

Troubleshooting and verification

No result: check that the post returned a valid GUID, the correct local player owns the presenter, a root is alive, the body implements `IFEntry`, and a custom layer has an entry host. A dialog or transition intentionally postpones presentation. Stale message after returning: pass a Scope and call **Dismiss Scope** as your flow ends. Incorrect model text: check the body class and refresh graph, including whether the cast succeeded. Overflow: coalesce repeated status reports or change capacity; use a separate history model when every event must be retained.

Notification runtime evidence records actual source/cooked results and limits. Tests use Win64 UE5.6.1 DX11 rendering, synthetic Slate command/back input and programmatic lifecycle checks. They do not prove physical controller/touch behavior, split-screen, arbitrary localization, audibility or complete product readiness.

PRESENTATION

Progress presentation in Blueprint

Use **Forge Progress** for ordinary linear tracks, radial outlines, square/rectangular perimeter outlines, segmented fills and whole charge cells. These are procedural UI styles. Sakura's requested cooldown uses the separate **InterfaceForge Center Tint** above its icon; keep that center-origin tint and its layering when reproducing that presentation.

Bind game-owned progress

Add Forge Progress to your widget, choose **Style**, **Desired Size**, **Thickness**, **Segments**, **Reverse**, **Start Offset** and theme options in the Designer. On your provider's changed event, call **Set Fraction** with a normalized0-1 value. The convenience IFEntryWidget already wires **Notify Data Changed** → **On Entry Refreshed**. A manual **Set Fraction** stops any previous UI-owned cooldown on this widget. For a decorative overlay above a clickable sibling, set its visibility to **Not Hit-Testable (Self & All Children)** (`HitTestInvisible`) so it cannot cover that sibling's input target.

The default RTS/RPG readouts use real provider-driven linear tracks. **Pause** → **Field journal** → **Watchpost survey** opens a Blueprint-only provider and custom entry. **Observe ridge** advances one of four observations; its count, fraction, availability and completed prose change through the same saved public graphs.

Customize the same saved entry

Open **WBP_WatchpostEntry** under `/Game/UI/Blueprint`. Set its **Progress Presentation** property in **Watchpost | Presentation** to Linear, Radial, Square Perimeter, Segmented or Charges. **Pre Construct** applies this property, including Designer previews. In this entry, use that property instead of changing only RecordProgress's Style; Pre Construct intentionally reapplies the entry's configured presentation.

Call **Apply Progress Presentation** on a live entry to change styles. This is an editable Blueprint custom event supplied by the example, using public plugin/engine nodes:

Blueprint workflow (text source; interactive diagram in the web guide)

```
flowchart LR
    A[Apply Progress Presentation: Presentation Style] --> S[Set Progress Presentation]
    S --> B{Radial or Square Perimeter?}
    B -->|Yes| D[Configure Progress: 96x96 / 4 segments]
    B -->|No| L[Configure Progress: 440x8 / 4 segments]
    D --> C[Cast RecordProgress Slot to VerticalBoxSlot / Set Horizontal Alignment: Center]
    L --> F[Set Horizontal Alignment: Fill]
    E[Pre Construct] -->|Progress Presentation| A
```

Its **On Entry Refreshed** → **Set Fraction** and observation button graph are unchanged. The same source/model supports every style; no new subclass or gameplay implementation is needed. The helper also prevents a Vertical Box's fill alignment from stretching a96×96 dial. In your own layout, use the appropriate slot/alignment or Size Box.

Actual source interaction after one observation (UE5.6.1,1280×720):



One observation shown by the same Blueprint entry with radial progress

The same entry at enlarged UI/text scale with a reversed outline and another origin (UE5.8.3 source), and completed charge cells (UE5.7.4 Development package), are in enlarged reverse and completed survey. These are original game interfaces; simulated input produced the recorded interactions.

Public runtime controls

Node / property	Result
Configure Progress	Sets style, desired size, segment capacity, direction and origin; invalid style, nonpositive/nonfinite size or nonfinite origin returns false and preserves the previous configuration. Valid segment counts clamp1-64 and origin clamps0-1. It invalidates layout/paint and preserves the current reading/clock.
Set Fraction	Finite values clamp0-1; nonfinite values display0. Stops a previous local clock.
Get Displayed Fraction	The actual manual or current timed reading.
Get Displayed Charges	Whole filled cells, quantized down using Segments as capacity. A tiny1e-5 cell-unit tolerance prevents floating-point count underruns. It owns no charges, item state or action rules.
Reverse	Fills from the right for line/cell styles, counterclockwise for radial/perimeter styles.
Start Offset	Turns/perimeter fraction: radial0 starts at the top; square0 at top-left. Only outline styles use this origin.
Use Theme Colors / Theme Override → Refresh Appearance	Follows Accent/Outline or a local theme; disable theme colors to retain your explicit Fill/Track colors.

For a charge display with three filled cells out of five, set Style=Charges and Segments=5, then **Set Fraction** to3÷5. The renderer and Get Displayed Charges return three whole cells, including normalized fractions that land a few floating-point units below an integer. A partly filled charge does not become a whole cell. Validate action availability and actual charges in your gameplay provider.

Timed UI presentation and countdown text

Start Cooldown(Duration Seconds) starts at1 and decreases to0 against an absolute game-time endpoint. Game-world pause freezes it; changing presentation does not restart it. Zero/negative/nonfinite duration completes immediately. **Clear Cooldown** clears its reading. **Get Remaining Seconds** → **Ceil** → **To Text (Integer)** → **Set Text** supplies a countdown label through your existing shared UI update path. Hide the label/tint when the remaining value reaches0.

Use this clock for a small transient UI timer when the widget owns its timing. For gameplay cooldowns, use the game's authoritative time/data and notify entries: rebuilding a UI widget must not restart an ability. The RPG example demonstrates provider-driven fraction/countdown fields; its center tint never starts a gameplay timer. Avoid independent per-widget population scans or creating one timer for every recycled list entry.

Current verification passes60source/exact Development-package progress runs acrossUE5.6.1/5.7.4/5.8.3, plus24shared RTS/RPG flow regressions and12real Ward/tint/world-entry regressions. All30saved BPs compile cleanly on each engine, and revised core builds/loads independently. Runtime record gives commands, hashes and limits. Physical controller testing remains deferred. The player-facing survey contains no instructional style selector or test-only timer controls; this scoped milestone is not whole-product release acceptance.

PRESENTATION

Center-origin tint

Add **InterfaceForge Center Tint** above an icon in an Overlay. Keep the icon and tint the same size; make countdown text the last layer. The tint ignores pointer hit tests so the action button beneath it keeps receiving interaction.

Call **Set Remaining Fraction** with `RemainingSeconds / TotalSeconds`, clamped to 0-1. At 1 the entire interior uses Remaining Opacity. At .75 the upper-right quarter uses Passed Opacity; at .5 the right half; at .25 only the upper-left quarter remains strongly tinted. At 0 the whole interior uses Passed Opacity. Hide the tint on completion if the completed state should have no overlay.

This renders the independently authored center-origin opening established by Sakura's inspected behavior. It does not start an ability or invent a gameplay cooldown. Bind an external source's changed event or timeline to the fraction. No per-widget entity scan or private gameplay dependency is required.

Tint, **Remaining Opacity**, **Passed Opacity**, **Start Angle** and **Reverse** are editable in the Designer. Defaults are white, .75, .15, 0 and false. Start Angle uses turns clockwise from the top: .25 starts at the right. Reverse changes opening direction. Call **Refresh Tint** after changing these properties in Blueprint.

A plain Unreal Image is equally valid: choose `/InterfaceForge/UIForge/Materials/M_IFCenterTint` for its Brush, use **Get Dynamic Material**, then **Set Scalar Parameter Value** for `Percent`, `RemainingTimeOpacity`, `PassedTimeOpacity`, `StartAngle` and `Reverse` (0/1); **Set Vector Parameter Value** for `Color`. `Percent` means normalized *remaining* fraction. Each widget uses its own material instance.

The material is newly authored from the behavioral specification and contains no copied reference packages, icon art, frame art or game design. The initial UE5.6.1 Win64 DX11 review covered five intermediate shapes and a continuous eight-second opening, including simulated mouse/keyboard Replay. Its verification image is an internal fixture. The original playable Emberwatch icon/frame, real provider clock/countdown and shared menu/HUD integration are documented in RPG authoring and RPG runtime evidence; later WorldLinks/SafeArea/Localization records retain their exact revisions. Physical controls and broader platforms remain unverified. Generic progress styles are separate presentation tools and do not replace this tint.

DATA AND REUSABLE ENTRIES

Presenting external gameplay data

Add **IFDataSource** to your gameplay adapter Blueprint. Implement **Get UI Model** and **Request UI Action**. The UI never needs to cast to your game's controller, inventory or ability component.

Create a stable **IFPresentationData** object for each visible model with **Construct Object from Class**. Set `StableKey`, `Title`, `Detail`, `Quantity`, `Fraction`, `Icon` and `Enabled` as appropriate. Return the same object from **Get UI Model**. After changing its fields, call **Notify Data Changed** once. Bind a screen or entry to **On Data Changed**, update its visuals and unbind on release/destruction.

Pass the adapter as `Data` to **Push Screen**. **Initialize Screen** receives it before activation; the screen stores it as `Screen Data`. Call **Get UI Model (Message)** with your agreed model name. Invoke **Request UI Action (Message)** for interaction. The adapter decides whether an action is valid and returns whether it accepted the request. Treat refusal as a normal result, and present feedback from the adapter.

Bind context-specific models in **Initialize Screen**, after calling its parent implementation. CommonUI may construct a pooled widget's Slate tree before it supplies the new context. **Construct** alone is therefore insufficient for a context subscription. Unbind the old context before assigning the new one, and unbind again during destruction/release. This also prevents one reused screen observing two players or entities.

The example project implements `Resources`, `Selected`, `Objective`, `Queue` and `Notice`, with `SelectNext`, `Patrol`, `Rally` and `Recruit` requests. Those names and the small outpost simulation belong to the example, not the plugin. Existing game systems can use different names and their own UObject models. Multiplayer adapters must send requests through their existing authoritative gameplay path; InterfaceForge does not grant authority or implement item/combat rules.

Use **InterfaceForge Text** for labels that should follow the local player's theme and text-size preference. Set `Role` and `Base Size`; select a `Theme Override` only for a local exception. **IF Button** exposes `Label Alignment`; its theme supplies padding, corner radius, font, colors and audio. **Request Back** and the root's **Go Back** invoke CommonUI's actual back handler, including Blueprint overrides and settings cancellation.

The saved Blueprint-only Watchpost source, screen and entry now have scoped source/cooked action/model/theme/focus/lifecycle proof. Follow Blueprint customization for the actual graphs and verification record for limits. The sample's FrontEnd identity model also replaces MainTitle/MainDetail through ordinary public screen nodes; menus/HUD documents that shared tree.

Menu presentation extends that saved workflow to optional Begin/Resume labels and confirmation wording, with missing/empty-model fallbacks and same-instance source reset. Menus explicitly refresh configuration text when asked; continuously changing entries still use Notify Data Changed and the entry lifecycle above.

For a remapping row, call **Get Registered Mapping** on `IFRemappingSubsystem` with your mapping name and Pointer/Keyboard/Gamepad family. It returns the first registered slot in that family and the engine's actual hardware identifier. Feed its result to **Get Preview Key**, **Stage Key Mapping** and **Stage Default**. Do not assume `DefaultKeyboardAndMouse` is the identifier of a registered mapping: engine contexts can register the neutral identifier instead. Touch capture is not implemented by this helper. Advanced multiple-slot/hardware selection can use the engine key profile's mapping rows directly.

DATA AND REUSABLE ENTRIES

Blueprint journal and reusable list entries

The playable Field journal uses `WBP_FieldJournal` (IFScreen), an ordinary engine List View named `JournalList`, and `WBP_JournalEntry` (IFEntryWidget). Open it from the pause menu. Its rows show the active example's presentation models; closing restores the underlying pause screen. These are original InterfaceForge layouts, not Game Master's exclusive design.

Populate a screen

Initialize Screen calls the parent with **Data**, then **Clear List Items**. **Get UI Model** retrieves Resources, Selected, Objective, Queue and Notice from the supplied **IFDataSource**; **Add Item** adds those model UObjects to JournalList. UI authoring does not implement gameplay rules. Your data source can return different IFPresentationData objects using the same contract.

`WBP_AshenPause` connects JournalChoice's **On Button Base Clicked** to **Push Screen** (Menu layer, WBP_FieldJournal, Screen Data). The screen's **On Initialized** sets **Initial Focus** to CloseJournal. That button's **On Button Base Clicked** calls **Request Back**, preserving CommonUI stack and focus restoration. The current visible label is Return to pause menu.

Create or replace an entry

Create a Widget Blueprint deriving from **IFEntryWidget** and design its visual tree. Set it as the List View's **Entry Widget Class**. Implement **On Entry Refreshed**: read **Entry Data**, **Cast to IFPresentationData**, then set your labels from **Title** and **Detail** through **Set Text**. This is the saved journal entry's actual graph. The core invokes refresh on assignment and when the model calls **Notify Data Changed**. **On Entry Initialized** also receives the assigned object if you need separate initialization work. The core entry unbinds the old model on reassignment/release and subscribes the assigned presentation model. For another object type, use your own binding and matching cleanup.

The saved journal entry uses **On Entry Selection Changed** to set title **Role** to Accent when selected or Body otherwise, change EntrySurface **Outline Width** to 2 or 1, and call **Refresh Appearance** on both. **On Entry Released** clears Title/Detail and restores the unselected role/outline. Reset your own timers, animations, tooltips and custom state as well. A pooled entry belongs to its currently assigned model; never treat a row pointer as the permanent view of an item.

For the IFEntryWidget base, select **On Entry Released** from **InterfaceForge | Entry**. Unreal also exposes a similarly named **UserListEntry** event; that engine event runs after the base has cleared its context. Use the InterfaceForge hook when cleanup needs **Entry Data**. Do not wire both cleanup paths or call Initialize Entry from inside cleanup; assign the next context through the normal list/source workflow after release. A completely custom User Widget instead wires the engine UserListEntry event to the IFEntry interface implementation, as described in the quick-start recipes.

Initialization with **None** data is valid for a static/default entry. The lifecycle fix tracks initialization independently of model presence, so Blueprint cleanup runs once and repeated release does not repeat it. Current source/cooked proof confirms None → release → real-model reassignment and pooled-row cleanup on UE5.6.1/5.7.4/5.8.3, alongside full flows/customization/cards and32cleanBPs per engine. Earlier IF3/BI artifacts predate this fix; current EB installation evidence is recorded separately.

Change themes

Mineral and Ember buttons use **On Button Base Clicked** → **Set Theme**, targeting **Get IFLocalPlayerSubsystem** from **Get Owning Player**. The Theme pin references DA_ForgeMineral or DA_ForgeEmber. The same screen and entries remain; theme changes also reach the underlying HUD and pause screen. This example's theme choice is session state, not a persisted preference.

Duplicate the sample screen/entry, change their Designer layout and event graphs, select the copied Entry Widget Class and copied Push Screen class, and supply another IFDataSource as Data. No plugin source edit is required. Watchpost

survey constructs BP_WatchpostSource and pushes WBP_WatchpostSurvey through public features; see Blueprint customization.

Keep large collections responsive

Use **Set List Items** to assign an array of stable model UObjects once. Retain each model while changing its fields, then **Notify Data Changed** once per changed model. Visible entries update immediately; offscreen models have no realized entry subscription and show their latest data when assigned to a recycled row. Avoid Set List Items or Regenerate All Entries for every text/cooldown change.

Keep the List View in a bounded Fill slot instead of creating thousands of children in a Scroll Box. **Get Displayed Entry Widgets** returns current realized views, not the complete model array. **Scroll Index Into View** or **Navigate To Index** reaches a distant item. **Set Selected Item** changes model selection immediately; row events may follow on the next Slate update. **Clear List Items** releases subscriptions before replacing the source.

Group each model's field changes before notifying it. The performance fixture deliberately notifies the entire collection to measure a producer burst; it does not recommend whole-population notifications every frame. Filtering, sorting and relevance are data-source decisions. Preserve model identity to retain selection. The UI plugin does not scan an entity population from each widget.

Collection measurements cover the saved journal at 1,000 and 10,000 models, normal/enlarged UI, 16 bulk-update batches, selection/far scroll/stale-source isolation/clear/reset/repopulate/close. Only 3/2 rows are displayed; the observed pool is 6/4 objects. All model UObjects remain retained. Timings describe this host/configuration; whole-scene frame observations include other rendering/work and are not isolated widget benchmarks or cross-device guarantees.

Style the standard List View

In an **IFScreen** Blueprint, add **Event On Theme Updated** → **Apply Theme to List View**. Connect your List View variable to **List View** and the event's **Theme** to **Theme**. IFScreen sends the current theme when constructed and again when **Set Theme** changes the owning player's theme. WBP_FieldJournal contains this actual graph. The thumb, active thumb, track and thickness follow the screen theme while entries and selection remain intact.

For an ordinary User Widget, call the node from **Event Construct** using your chosen theme and again from your theme-change handler. The node applies once; it does not automatically subscribe a standard List View. For a local override, supply the override theme on every call instead of the event theme. Applying local styling does not change the player theme. A later global-event call can replace it if your graph supplies the global theme.

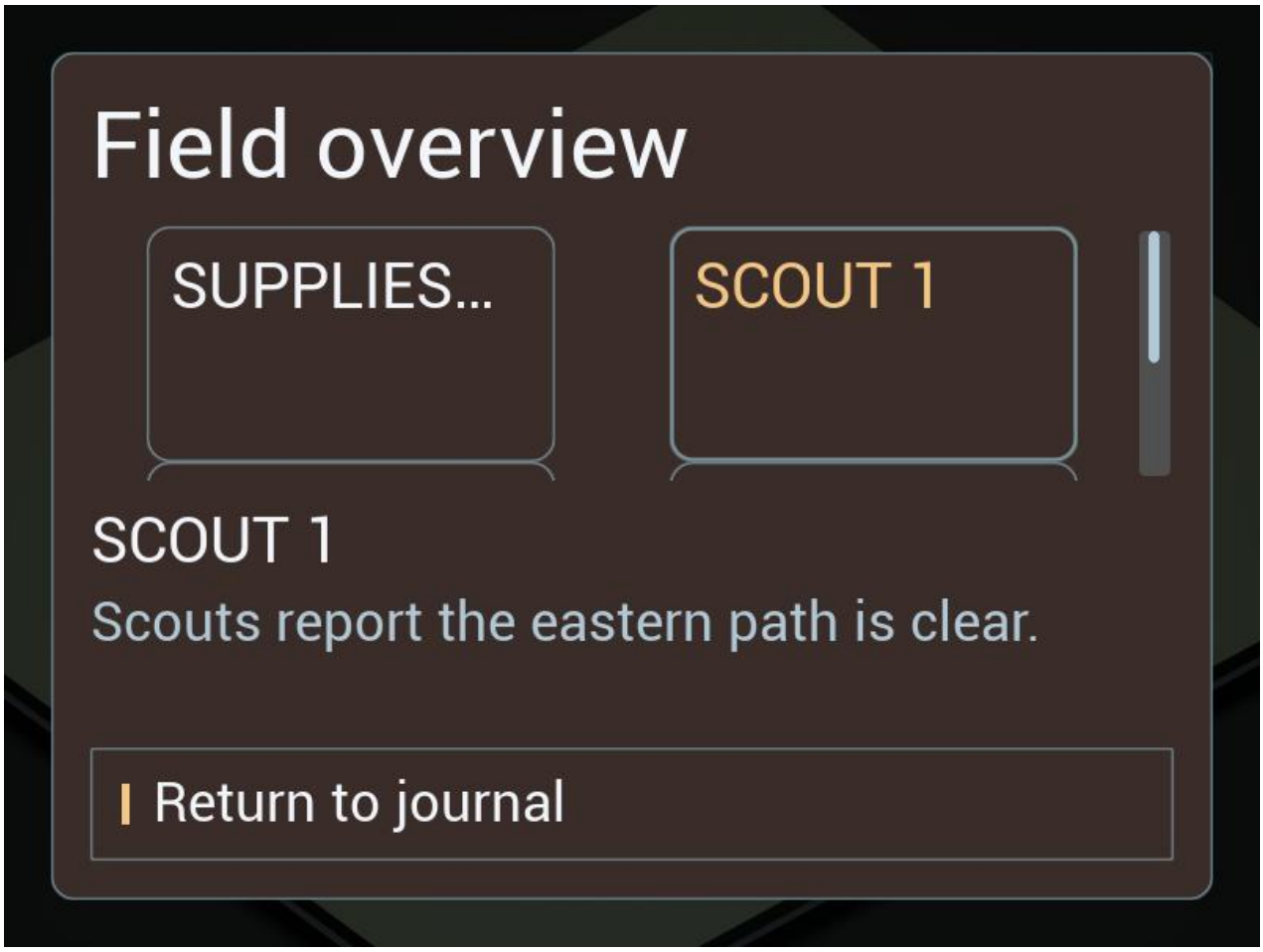
Theme **Muted Text** colors the resting thumb; **Accent** colors hover/drag; **Outline** at 35% alpha colors the track. **Scroll Bar Thickness** uses 4-32 Slate units, with 6 for a nonfinite value. DPI/player UI scaling applies normally. The node accepts engine Tile View and Tree View subclasses; their entry classes, data and navigation remain engine-owned. It returns false for a missing view or unsupported live scrollbar structure. No theme uses InterfaceForge defaults. No Regenerate All Entries is required; both preconstruction configuration and constructed engine views are handled.

Live List View styling/recycling is verified in the saved journal. Field overview adds a saved playable Tile View with live selected details and the same provider contracts; current runtime acceptance is recorded separately. Tree View still has preconstruction styling evidence only. Entry layout decisions explains why these flat RTS/RPG workflows use List/Tile examples and how a project can extend the contract to a hierarchy; no completed Tree workflow is advertised. Physical controller/touch and arbitrary third-party entry/Slate classes remain unverified. See entry lifecycle, customization, and the latest milestone record for exact revision scopes.

DATA AND REUSABLE ENTRIES

Blueprint field overview and tile entries

From either playable HUD, open **Pause** → **Field journal** → **Field overview**. The same saved `WBP_FieldCards` presents the current RTS expedition or RPG party models. Select a card to read its full current description; Return to journal restores the previous screen. This is a player interface, with no inventory or gameplay rules in the UI plugin.



Original enlarged field overview with bounded previews and live selected details

Build the screen

Create a Widget Blueprint deriving from **IFScreen**. Add an ordinary **Tile View**, set **Entry Widget Class** to your **IFEntryWidget** Blueprint, and put it in a bounded Fill slot. `WBP_FieldCards` uses 208 × 120 logical-unit entries; Unreal determines the number of columns from available width. A separate **InterfaceForge Scroll Box** holds the selected detail entry so full text can scroll independently.

Implement **Initialize Screen** and call its parent with **Data**. Call **Clear List Items**, then **Get UI Model** and **Add Item** for the source's model keys. The sample uses Resources, Selected, Objective, Queue and Notice. Its existing adapters translate these to expedition or party data. Supply stable model objects; changing a view does not require recreating gameplay state.

Set **Initial Focus** to the live Tile View, then **Set Selected Item** to the first model and **Initialize Entry** on the detail widget with that model. Bind the Tile View's **On Item Selection Changed** event. Use **Branch** on **Is Selected**, then **Initialize Entry** with **Item** on the true branch. Ignore an old item's deselection event when assigning the new detail. InterfaceForge unbinds the previous model before subscribing to the new one.

Blueprint workflow (text source; interactive diagram in the web guide)

```
flowchart LR
    A[Initialize Screen: call parent] --> B[Clear List Items]
    B --> C[Get UT Model → Add Item]
    C --> D[Initial Focus → Set Selected Item]
    D --> E[Initialize detail Entry]
    F[On Item Selection Changed] --> G{Is Selected?}
    G -->|True| H[Initialize detail Entry with Item]
```

Make a card Blueprint

`WBP_FieldCard` is an independently editable copy of the example's journal entry. **On Entry Refreshed** reads **Entry Data**, casts to **IFPresentationData**, and uses **Set Text**. Its title disables Auto Wrap Text and uses **Overflow Policy = Ellipsis** in a bounded Fill slot. This deliberately provides a compact preview; the selected readout retains the complete title and detail. Text enlargement does not remove the full description.

On Entry Selection Changed updates the title role and surface outline through **Refresh Appearance**. **On Entry Released** clears text and selection styling. These are the same public lifecycle events used by List View entries. Changing the entry class or Designer tree requires no plugin source edit.

Live updates, themes and cleanup

Change fields on the external model and call **Notify Data Changed** once after the edits. The realized card and selected detail update through their existing subscriptions. Avoid rebuilding the tile collection for each text change.

Connect **On Theme Updated** → **Apply Theme to List View**, passing the Tile View and event Theme. The engine Tile View inherits List View; its scrollbar and the entry components use the existing theme system. Local theme overrides remain widget-owned.

Connect **On Deactivated** → **Clear List Items** → **Release Entry** for the detail widget. The sample's Return control calls **Request Back**. The journal's Field overview button calls **Get Root Layout** → **Push Screen**, Menu layer, `WBP_FieldCards`, with the journal's **Screen Data**. Reopening starts from fresh selection and binds the current source.

Keyboard/controller directional navigation follows the actual grid: horizontal movement between columns, vertical movement between rows. Shared input presentation applies to this custom screen. Current expanded source checks route actual Slate pointer down/up events through the card hit area, controller-family changes above a modal, and modal Back → grid focus → journal launch-control focus. These are software input tests; physical controller testing is deferred. Current source checks are recorded separately from packaged, newer-engine and Shipping acceptance; this recipe does not claim those gates are complete.

Current three-engine source/cooked runtime record records24current card reports and24full-flow regressions, actual commands, rejected diagnostics and remaining verification.

The Field overview/world-choice labels and default dialog text are included in the refreshed271-entry Turkish catalog. Refreshed-label runtime evidence verifies24 source/exact Development-package cases across UE5.6.1,5.7.4 and5.8.3, with32 clean Blueprint compilations per engine. The owned runtime language switch starts after English startup; it does not close the known UE5.6 Turkish-editor-startup gate. Earlier L2 and Shipping records retain their original revisions and scope.

DATA AND REUSABLE ENTRIES

Choose an entry layout

InterfaceForge supplies the entry lifecycle and styling contract; Unreal supplies virtualized List View, Tile View and Tree View. A project can design the entry entirely in Blueprint without changing plugin source. Use stable model UObjects, **On Entry Refreshed**, **On Entry Selection Changed** and **On Entry Released**; call **Notify Data Changed** after model updates. See journal entries and field cards.

The shipped workflows use a List View for the journal's detailed rows and a Tile View for compact field summaries. The RTS provider exposes resource, selected group, objective, production and notice models; the RPG provider exposes party, objective, actions and status. Both are flat collections in these examples. A Tree View would add expansion and parent/child semantics without solving a player task in either interface. This release therefore does not advertise a completed hierarchical-tree workflow or provide a decorative tree showcase.

For a project that needs a hierarchy, use an engine Tree View and its **On Get Item Children** binding with project-owned parent/child models. Use an IFEntryWidget Blueprint as the entry class and the same assignment/update/release contract. Keep expansion state and child retrieval in your screen/data source; do not create a new entry subclass for every node type. Supply your own navigation/expansion affordances and validate recycled entries against the assigned model.

Apply Theme to List View accepts the engine's Tile View and Tree View subclasses. The current release evidence verifies live List/Tile workflows; Tree styling has preconstruction coverage only. Runtime Tree expansion, focus restoration and large hierarchical collections are not certified. This is a deliberate example-scope decision based on the actual player workflows, not a claim that a flat-list test validates trees.

Do not use a Scroll Box with thousands of manually created entry children when the engine's virtualized views fit the task. Preserve model identity when filtering/sorting; refresh changed models instead of recreating every entry. The journal's1,000/10,000-model tests and the field-card interaction tests document their own performance/input limits.

DATA AND REUSABLE ENTRIES

World labels through public Blueprint features

InterfaceForge uses Unreal's **Widget Component** for world rendering. **Set World Entry**, **Refresh World Entry**, and **Clear World Entry** connect it to the same IFEntry/data/theme contracts as screen widgets. No actor search, gameplay tick, distance policy or interaction authority is added to core.

Connect an offset marker to its subject

An offset marker may clear a HUD panel but look detached from its actor. Use **Get World Entry Widget Point** and **Update World Entry Link** to draw a caller-owned connection without adding an entity scanner or requiring a special widget subclass.

1. Add a normal **Static Mesh Component** to your label actor, assign a mesh such as the engine Cube, and provide your own material. Disable collision and shadow casting for a passive UI connection. The mesh's local X axis runs along the line; its measured Y/Z bounds determine width. Non-centered mesh bounds are accounted for.
2. Obtain the actual widget to attach to, such as your progress bar. **Get World Entry Widget Point** takes your Widget Component, that widget, and **Alignment=(.5,1)** for bottom-center. It returns **World Position** and a success Boolean. It resolves cached rendered geometry; retry after the first layout/render if false. Widgets in another entry, screen-space/cylindrical components and missing geometry return false. Alignment clamps to0-1.
3. On success, call **Update World Entry Link** with Link=your Static Mesh Component, From=World Position, To=your actor's chosen world anchor, Width=your desired world-unit thickness. Success updates transform but preserves your visibility policy. Invalid/coincident endpoints, missing mesh or unusable mesh bounds hide the connection and return false.
4. Set the connection's visibility using the same flow/range/marker-validity policy as the label. Hide it when the entry is cleared, paused or its anchor ends. You own updates; call from your placement event or selected-marker update policy. Neither node polls actors, installs a tick nor owns gameplay data.

Blueprint workflow (text source; interactive diagram in the web guide)

```
flowchart LR
    W[Rendered bar or custom widget] --> P[Get World Entry Widget Point]
    C[World Widget Component] --> P
    P -->|Success / World Position| L[Update World Entry Link]
    A[Your actor anchor position] --> L
    M[Your Static Mesh Component / material] --> L
    V[Your flow / range / validity policy] --> S[Set Visibility on the link]
    L --> S
```

The saved RPG **BP_ForgeWorldReadout** uses these public nodes through its example placement adapter. **Link Widget Name** defaults to ProductionProgress; **Anchor Link Offset**, **Anchor Link Width**, **Anchor Link Color** and **Show Anchor Link** are editable Blueprint properties. The actor retains weak anchor/player references. The two example markers update at30Hz to follow the small party smoothly; this is not a large-population policy supplied by the plugin. You can call the same nodes entirely in your Actor Blueprint with your own mesh, entry and cadence. Planar world components are supported; other geometry needs your own projection adapter.

Actual endpoint/engine coordinate round-trip, model update, invalid points/tree, range/pause/clear/anchor-lifetime and rendered source/package scope are recorded in Evidence/WORLD_LINK_RUNTIME.md. Physical device, split-screen, large population, occlusion and arbitrary camera/geometry support remain separate gates.

A working label in Blueprint

1. Add a normal **Widget** component to your Actor Blueprint. Choose Space=World, Draw Size appropriate to your tree, and a Pivot such as(.5,1). For a passive label disable **Window Focusable** and **Receive Hardware Input**, and use no collision. Leave Widget Class empty; Set World Entry creates the owning-player-aware body.
2. Create a Widget Blueprint using **IFEntryWidget**, or a UserWidget implementing **IFEntry**. **On Entry Refreshed → Cast Entry Data to IFPresentationData → Set Text(Title) / Set Fraction(Fraction)** updates your label/bar. **On Entry Released** clears custom references and presentation. The convenience base subscribes to Notify Data Changed and unbinds on release. A plain interface implementation owns its subscriptions and must make release safe to repeat.
3. From your actor's initialization, call **Set World Entry** (InterfaceForge | World): Component=your Widget component, Local Player=the actual local PlayerController, Entry Class=your widget, Data=your stable model. It returns the created UserWidget, or None for invalid input. Keep gameplay/model authority in your provider. In a multiplayer game create local presentation from legitimate replicated data; this function implements no network rules.
4. On owner/anchor cleanup call **Clear World Entry**. It releases the old IFEntry binding, removes the body and clears the component's automatic Widget Class so Init Widget cannot recreate stale UI. Invalid Set inputs leave the existing body unchanged; a valid replacement releases it before initializing the new body. Nested Set/Clear mutations during the same component's release/initialization are rejected.

Blueprint workflow (text source; interactive diagram in the web guide)

```
flowchart LR
  A[Actor initialization] --> B[Stable presentation model]
  B --> C[Set World Entry]
  D[Local PlayerController] --> C
  E[Engine Widget Component] --> C
  C --> F[Custom Blueprint IFEntry]
  B -->|Notify Data Changed| F
  G[Actor End Play / anchor destroyed] --> H[Clear World Entry]
```

The helper creates the widget with the explicit PlayerController before Slate construction, then sets the component's Owner Player. IFText/IFSurface/progress bind to that local player's theme. Changing **Set Theme** updates the body without copying/rebuilding the widget tree. Core requires a valid local controller in the same world and rejects dedicated-server creation, abstract/deprecated classes, invalid data and classes without IFEntry.

Inspect the playable example

Emberwatch creates two **BP_ForgeWorldReadout** instances for its party. Open the saved Blueprint's **Event Create Readout** graph: **Display** and **Entry Class** property getters plus Player/Data pins feed the actual **Set World Entry** node. The native example actor supplies placement/lifetime/visibility only, with a convenience fallback using the identical public node. It is not a required plugin subclass: the four steps above also work on an ordinary Actor Blueprint.

WBP_ForgeWorldStatus is a saved custom IFEntryWidget Designer/graph, with a one-line IFText accent title and a themed linear bar. **WorldTitleFit** is an ordinary Scale Box: Scale To Fit, Down Only. Its title has Auto Wrap Text=false. Model Title and Fraction drive the same public nodes; release clears text and fraction. The fit-down title keeps the entire compact name/percentage within the fixed render target when text size increases. Supply a short localized world-label title; use a contextual screen/tooltip for lengthy descriptions rather than shrinking arbitrary paragraphs into a world marker.

The example uses a160×64 world render target, scale1.8, pivot(.5,1), one-sided world rendering, and configurable **World Offset**, **Camera Side Offset**, **Maximum Camera Distance** and **Entry Class**. Its orthographic arrangement moves labels into the gap between HUD cards and raises them above scene props. Replace these example placement choices in Blueprint for your camera/game. Ward updates the shared Hero/Companion models, so both HUD and world labels show restored vitality. Begin reveals them; pause/modal suppresses them; resume restores them; return hides them. A destroyed anchor releases and destroys its example presenter immediately. A separate fixture actor demonstrates the same binding with a standalone model and no RPG provider.

Placement checks in this small example use only each known anchor/camera at a1/30-second actor interval so the two connections follow moving subjects smoothly; flow visibility changes only when pause state changes. It performs no population search in a widget. This is not a scalable thousands-of-markers manager or a10kentity performance result. For

larger populations, select relevant anchors centrally and pool/budget presentations in your adapter.

Rendering, scaling and input choices

World space uses a rendered world surface and normal scene occlusion. **Screen** space uses an actor-anchored screen layer without world occlusion. Draw Size controls the world render target; actor/component scale controls its world extent. Neither is a second viewport user-scale pass. IFText still respects the shared text-size preference. Choose readable physical sizes for your camera distance. Unreal documents Widget Component space, target size, redraw and focus controls.

Default automatic redraw permits live theme/data/text updates. With **Manually Redraw** enabled, call **Refresh World Entry** after model/theme/preference changes; it executes Refresh Entry and **Request Render Update**. This helper does not silently subscribe to arbitrary data sources or schedule redraws. Animate/dynamic labels need an appropriate engine redraw/tick policy. **Draw at Desired Size** can resize the render target frequently; prefer a stable target and responsive tree when practical.

Passive labels do not join the CommonUI menu focus tree. An interactive world terminal needs explicit focus/routing and a project-owned Widget Interaction Component. Set World Entry does not make a passive label a tested VR/touch/controller terminal. Keep screen dialogs/settings in the existing screen stacks.

Each component body has one owning player. For different per-player themes/visibility in split-screen, use separate per-player presenters or a suitable centralized screen-marker adapter. A single world surface should not be advertised as independently styled for every viewer. Engine screen-layer ordering, occlusion, camera changes, split-screen and physical devices require project testing. Runtime record states the actual tested scope.

GAME INTERFACES AND ADAPTERS

Build and customize the RPG interface

The Emberfall revision separates a third-person encounter from the RTS. Its camera and tiny combat adapter live in ForgeExamples. The UI core knows screens, commands, models, themes, input presentation and entry lifecycles; it does not implement an MMO, inventory or character controller.



Actual RPG HUD with top-center target and compact action bar

Open the usable example

In UE5.8.3 open `/Game/ForgeExamples/Maps/Emberfall`, Play and choose **Begin a new journey**. New does not overwrite the existing chronicle; Save does. The current Enhanced Input profile determines your keys. Read the action-bar prompts or open **Menu** → **Settings** → **Controls** rather than assuming the defaults still apply.

Default keyboard actions are1-6, Tab to cycle targets and WASD movement. Hold right mouse to orbit and wheel to zoom. The example uses camera-relative movement. Menus and modals take ownership away from movement/look. In **PIE**, UIForge offers the configured Back action to the active player's CommonUI router before Unreal's Stop Play shortcut. The editor toolbar remains available to stop play; global editor bindings are unchanged. A key selector keeps its release event so Escape cancels capture first.

The visible scene is a small local encounter with a plane-constrained movement adapter. It is not a replacement for Character Movement, jumping, world traversal or multiplayer gameplay in your game.

The editable assets

All paths below are beneath `/Game/ForgeExamples/UI/RPG/`.

Asset	Developer edits
WBP_IFRPGHUD	IFScreen class defaults: Input Policy, Shortcuts, Gameplay Key Mappings, focus and Back Action Name

Asset	Developer edits
WBP_IFRPGLayout / DA_IFRPGLayout	Designer region widgets and data-asset anchors, sizes, offsets and compact behavior
WBP_IFRPGTarget	Name, linear health bar and target details at the top center
WBP_IFRPGVital / WBP_IFRPGParty	Compact player/resources and interactive party entries
WBP_IFRPGAbility	Slot artwork, icon, center-origin cooldown, readable counter and live applied-key prompt
WBP_IFRPGMain / WBP_IFRPGPause / WBP_IFRPGJournal	Ordinary editable Widget Blueprints for the usable windows
WBP_IFRPGSettings	Four-tab IFSettingsMenu; rows remain the public settings/binding controls
DA_IFEmberglass	Fonts, palette, audio, transition settings, surface brushes and default/dialog style keys
BP_IFRPGCamera	Example-only spring-arm distance, lens, sensitivity, invert and follow settings

The RPG map uses original cartographic geometry from the actual courtyard. In the example adapter, **RPG Map World Min / Max** match the artwork: XY min(-2600,-2400), max(2600,3600). U increases with world Y and V with negative X. The RPG does not draw an RTS camera-frustum rectangle. Change the material/texture and bounds together for your own level; live markers remain ordinary IFMapData.

The RPG defaults place the target at top center, player resources above the bottom action strip, party at the upper left and map/objectives at the upper right. They are a layout profile, not a requirement imposed on other games. RTS keeps its separate army interface.

Make your own skill entry in Blueprint

1. Create **Widget Blueprint**, choose **IFReadoutEntry** as its parent and name it WBP_MySkill.
2. In Designer add a64×70 Size Box containing a Canvas Panel. Add **IFCommandButton** named `EntryCommand`, **Image** named `IconImage`, **IFCenterTint** named `CooldownTint`, a Text Block named `DetailText`, and **IFActionPrompt** named `ActionPrompt`. Use a54px icon/tint inside the64px slot; keep decorations hit-test invisible.
3. Set the prompt's **Label Font Size** and disable **Wrap Label** for a keycap. A Designer Scale Box with **Down Only** fits a long applied key on one line. Do not hardcode a number into a Text Block: the prompt listens to Enhanced Input settings/profile changes and CommonInput device changes.
4. On the entry instance set **Model Name**, **Entry Action Name**, **Mapping Name** and **Tooltip Model Name**. For the existing second skill these are Ability2, Ability2, IF_Ability2 and Ability2Tooltip. Enable **Detail Only When Fraction Active**.
5. Add the entry to your own HUD, or call **Get Region Widget** / **Set Region Widget** on an IFHUDLayout to replace a region. To use the existing bar, replace the ability entry classes in its Designer row. No plugin source change is required.
6. Supply a presentation model through **IFDataSource** → **Get UI Model**. Update Title, Detail, Icon, Enabled and Unavailable Reason together, then call **Notify Data Changed**. For a running fraction use **Set Fraction**. The framework displays the model; your game validates and executes **Request UI Action**.

Expected result: the same Blueprint entry responds to your model, click, focus, tooltip, applied key and controller presentation. For fully custom data, implement IFEntry's **Initialize Entry**, **Refresh Entry**, **Set Entry Selected** and **Release Entry** in your Widget Blueprint. Clear subscriptions and temporary state on Release; see entry decisions.

Make keyboard actions work while the HUD has focus

Register your player-mappable Input Mapping Context with **Enhanced Input User Settings**, and add the context to the local player's Enhanced Input subsystem. Enable Enhanced Input user settings as shown in installation.

In the HUD's **IFScreen** → **Shortcuts** array, add a Mapping Name / Action Name pair for each UI-routed game command, for example `IF_Ability2 → Ability2`. The screen resolves the *applied* engine key and calls the data source's **Request UI Action**. Shortcuts only operate on the active top screen, stop during transitions, and avoid editable text/key selectors.

For continuous movement, put the registered mapping names in **Gameplay Key Mappings** and the appropriate stick keys in **Gameplay Analog Keys**. The HUD forwards those inputs to the game while preventing widget navigation from stealing them. The RPG HUD also includes IF_Pause in this array and in Shortcuts, so its applied pause command works with viewport or widget focus. Leave gameplay forwarding empty on settings and other menus; CommonUI owns their Back behavior. Keep mouse capture at **Capture During Mouse Down** for this mixed HUD: Unreal's scene viewport otherwise skips initial left clicks when only right-button capture is allowed.

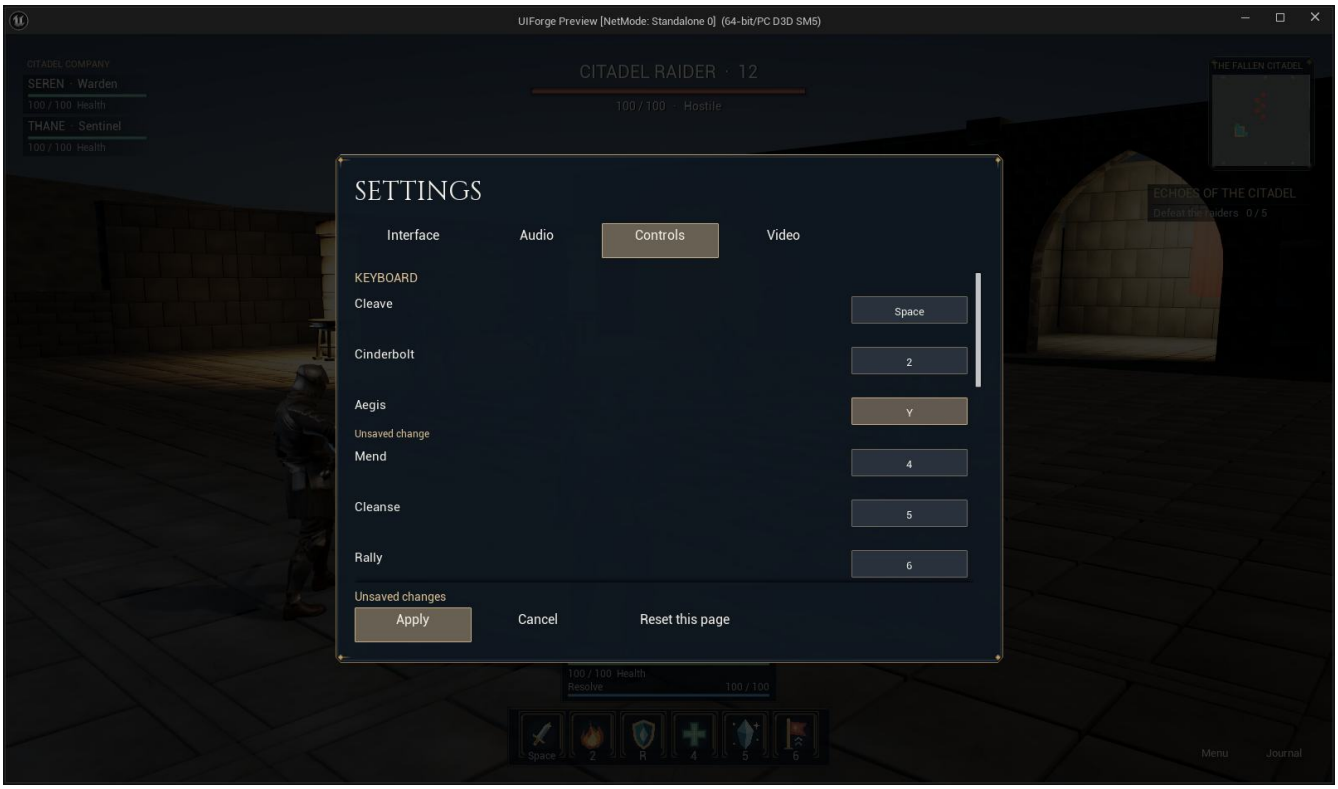
A Game And UI HUD with **Show Keyboard Navigation Focus** disabled returns focus to the viewport after transitions in pointer/keyboard presentation. Focused text/key editors keep their focus. Menus retain their normal CommonUI navigation focus.

Your game's Enhanced Input actions remain its gameplay path when the viewport owns focus. Avoid a second hardcoded key array. Normal Blueprint users edit IFScreen class defaults in the editor; the offline example generator must recompile after changing class defaults so UMG's generated instance template agrees with the CDO.

Build the rebinding settings tab

Use the detailed settings composition tutorial. Place **IFBindingEntry** rows in the Controls page, set **Mapping Name** and input **Family**, and use an **Input Key Selector** named `KeySelector` inside your row Blueprint. IFSettingsMenu creates one editing session for preferences and bindings.

- Choosing a key stages it. **Cancel Settings** discards the draft; gameplay stays on the old key.
- **Apply Settings** validates conflicts, writes the Enhanced Input profile, rebuilds control mappings and refreshes prompts. Test the new key *and* the old one.
- **Restore Page Defaults** stages defaults for Controls; Apply commits them. Reserved navigation keys and conflicts produce a reason instead of silently overriding another action.
- The ready-made first-slot workflow is **Begin Edit** → **Stage Registered Key** → **Validate Pending** → **Apply** on IFRemappingSubsystem. Resolve registered rows rather than inventing hardware identifiers. Advanced context-aware conflict policies are configurable; analog axes, modifier chords and arbitrary new mapping slots are outside this digital-key editor's current contract.



Actual native key-selector draft before Apply

Restyle without duplicating the behavior

On DA_IFEmberglass edit **Surface Styles**: Window is the etched major-window brush, Inset is the smaller framed treatment, and Combat is the quieter HUD/row surface. Set **Default Surface Style** and **Dialog Surface Style**, or select **Surface Style** on an individual IFSurface. Brushes may use your own texture/material and nine-slice margin. Other themes can omit these keys and retain the procedural fallback.

Change Heading Font / Body Font independently. The example uses unmodified Cinzel for major headings and the regular engine body face for combat labels. The font's SIL OFL notice is included with the example. Editable frame/slot paths and deterministic raster sources are in ContentSource/UI/Emberglass; the core does not depend on them. Replace the atlas/Icon Image or UV rectangle to supply your own icons.

The cooldown remains **IFCenterTint** over the icon, using the same center-origin interior material and Percent/Color/RemainingTimeOpacity/PassedTimeOpacity/StartAngle/Reverse controls. It is not the health-bar renderer. See center-origin tint and progress variants.

Verification scope

Current evidence is recorded in the product verification record. The revision includes actual settings-session rebinding, native key selection/Apply/activation, old/new movement checks and profile reload. Controller presentation tests use software Slate events; physical controller, unplug and touch testing remain user-deferred. Do not treat those software tests as hardware certification or these local examples as network-ready gameplay.

Shared window and ability presentation

The current revision uses the public skin system for Window/Header/Row/Combat surfaces, role buttons, art-based Linear resources and theme variants. It preserves the separate IFCenterTint cooldown path. **WBP_IFRPGAbilityBook** is an ordinary IFScreen populated by the same six stable skill models as the hotbar; the full Blueprint recipe explains creating/replacing its cards and gameplay adapter. Load, settings, journal, ability browser and confirmation dialogs share the same header/action conventions. Bottom HUD shortcuts use Auto Height so enlarged labels grow upward from their bottom anchors.

GAME INTERFACES AND ADAPTERS

Build a working ability browser from public UI widgets

The Emberfall **Combat arts** button opens an actual game window showing the six skills used by its action bar. Each card has an icon, the applied key prompt, description, cost/recovery text, and a Use ability action. Using an available art returns to gameplay and executes that same skill. A recovering or otherwise unavailable skill keeps its reason and cannot execute. This example supplies local encounter rules; UIForge supplies presentation, binding, navigation and screen management.

The saved Designer assets live under **/Game/ForgeExamples/UI/RPG**:

Asset	Purpose
WBP_IFRPGAbilityBook	IFScreen: scrim, fitting frame, header, scrollable two-column cards, Return
WBP_IFRPGAbility	IFReadoutEntry: icon, mapped prompt, countdown, original Sakura-style center tint
WBP_IFRPGAbilityCopy	IFReadoutEntry: wrapped title and gameplay description
WBP_IFRPGLayout	HUD profile/regions and Combat arts command

No runtime widget tree is generated by the editor authoring utility. These are normal editable Widget Blueprints. The example's ForgeExamples adapter handles **AbilityBook** and **UseAbility1…UseAbility6**. Those names are example actions, not hard-coded plugin requirements. Your game supplies its own adapter and validation.



Actual saved ability browser running in Unreal

Compose a card in Blueprint

1. Create a Widget Blueprint derived from **IFScreen**. Use a Canvas with a full-screen scrim and a Scale Box set to **Scale To Fit / Down Only**. Inside it, use a bounded Size Box and an **InterfaceForge Surface**, **Surface Style = Window**.

2. In its Vertical Box, place a **Header** surface, a flexible Scroll Box, and a footer. Keep the close/return buttons on Auto slots. Set **Request Screen Back** on IFCommandButton to share CommonUI's Back path.
3. Make a card from **Surface Style = Row**, with a Horizontal Box containing an ability entry and a text entry. Add an Auto-sized Use button below, aligned right. A Fill spacer above it aligns paired cards when descriptions have different heights.
4. On the icon entry set **Model Name = Ability3**, **Mapping Name = IF_Ability3**, **Entry Action Name = UseAbility3**. Set its atlas **Icon UV Min / Max** or use a separate texture. On the description entry set **Model Name = Ability3Tooltip**. Leave Tooltip Model Name empty on the book controls because the complete description is already visible; the action bar retains its contextual tooltips.
5. On the Use **IFCommandButton**, set **Model Name = Ability3**, **Action Name = UseAbility3**, **Use Model Label = false**, **Label = Use ability**. Its context, availability and reason come from that stable model. Do not reconstruct data every frame.
6. Set the screen's **Initial Focus Name** to a usable button such as **Use0**, and **Fallback Focus Name = Return**. Push it through **Get Root Layout → Push Screen**, **Layer = Menu**, **Data = your IFDataSource object**.

Expected result: the window participates in the same input presentation, cursor, focus, back navigation, theme and transition behavior as settings and pause. Changing a registered key updates the icon prompt through Enhanced Input. Opening a modal keeps the window's data alive while focus belongs to the modal.

Blueprint workflow (text source; interactive diagram in the web guide)

```
flowchart LR
    A[Combat arts command] --> B[Adapter: open IFScreen with its source]
    B --> C[Entries subscribe to stable skill and description models]
    U[Use ability / mapped prompt] --> V[Request UIAction with current context]
    V --> D[Game adapter validates active screen, actor state and skill availability]
    D --> E[Close book and request existing gameplay skill]
    E --> F[Actual resource / cooldown changes]
    F --> C
```

Supply your own gameplay

Implement **IFDataSource → Get UIModel** to return stable presentation objects, and **Request UIAction** to validate requests. For an **IFPresentationData**, edit **Title / Detail / Icon / Enabled / Unavailable Reason**, then call **Notify Data Changed**. Use **Set Fraction** for lightweight cooldown/resource updates. The convenience entry subscribes/unsubscribes during initialization, recycling and cleanup. Your own Blueprint can instead implement the IFEntry interface and draw any tree.

UI actions are requests. Validate actor lifetime, resource/range/cooldown state and authority in your game. Emberfall additionally rejects the use action when its book is inactive, transitioning, behind another screen, or bound to another source. A disabled UI button is useful feedback and does not replace validation.

For a networked game, show or close the window locally and send a validated gameplay request through your existing owned actor/RPC architecture. This local example is not an MMO server or replicated ability system. Inventory, equipment and talent progression are outside this plugin.

Change its look without changing the game

Duplicate **WBP_IFRPGAbilityCopy** to replace its typography or layout, or author your own **IFEntryWidget / IFEntry** Blueprint. Replace the card's entry class in Designer, keeping the model contract. Put project-wide brushes under a theme's **Surface Styles / Button Styles**; use **Theme Override** for one control. Call **Set Theme** to switch the whole live tree between Emberglass and Moonsteel. **Set Region Widget** replaces an entire HUD region with your own instantiated widget; **Refresh Screen Binding** binds a newly attached IFReadoutEntry to that screen's source.

The native **IFCenterTint** path preserves the center-origin tint reproduction. Resource bars use the separate **Forge Progress / Linear** art path. See button, resource and window styling, cooldown parameters, and game integration.

Current verification

The competitive pass records 13 runtime checks for actual CommonUI use, current stable model contexts, resource consumption, cooldowns, unavailable activation, malformed action rejection and Back. An early test found and fixed an action accepted during the closing transition. The separate customization receipt exercises region replacement, live theme changes, and a saved Blueprint-only source/entry/button interaction. Current native packaged mouse input opens the book and Use executes actual Aegis, with the applied Y prompt and real resource/recovery updates. Responsive review includes the full six-card window at1280×722. The active quality ledger records all test limits. Physical controller, touch and actual disconnect testing remain deferred.

GAME INTERFACES AND ADAPTERS

RPG revision: research, decisions and acceptance

Status: research complete; implementation in progress. The previous passing functional tests did not make the RPG presentation successful. This revision addresses the owner's rejected camera, HUD hierarchy and window treatment. The RTS stays separate.

References examined

Primary source	What it actually demonstrates	UIForge decision
WoW: updated HUD, including the published Modern Preset image	Player and target information grouped near the action bar, a clear central world view, small peripheral controls, and recognizable frame/ornament treatment. The article describes independent component configuration.	Compact combat cluster, restrained bronze/obsidian frame identity, no permanent boxed explanation under every icon. Independently authored geometry and artwork; no Blizzard asset copying.
FFXIV: Navigating the Game Screen, including the annotated PC HUD image	Party, objectives, target, status and casting have distinct roles. Many peripheral widgets use little opaque backing. Party targeting is interactive.	Small party rows, text-led objective tracker, transient cast/status information, secondary detail in tooltips/windows. No fake network chat to fill empty space.
FFXIV: HUD customization and HUD guide	Size/layout/display settings and separating target health, cast and status support different information priorities.	Continue using public layout profiles, entry classes and theme overrides; density should not require editing plugin C++. Do not claim an in-game drag editor unless implemented/tested.
GW2: action camera	Direct character/camera control, clear targeting states and automatic camera release for dialogs/panels.	RPG camera orbit/zoom and camera-relative movement, menu ownership wins over look input, gamepad look uses meaningful dead zones. Right mouse looks rather than issuing RTS movement orders.
Unreal: Spring Arms	Engine camera obstruction handling and configurable camera attachment.	An example-only Blueprint-editable spring-arm/camera rig; no camera system in the UI core.

These references support interaction/hierarchy choices, not a claim that one layout is universally best or that UIForge matches their art budgets. The historical published WoW/FFXIV images are references to specific layouts, not claims about every current patch. Reference pixels are viewed remotely and are not shipped.

Current implementation audit

- RPG uses a fixed camera approximately1,270cm from the character with650cm elevation. It cannot orbit. The diagonal view, ground selection rings and right-click world orders make it read as RTS.
- Player/target/party/objectives/cast all reuse boxed readouts. Their equal visual weight obscures what matters now. Minimap occupies192×204Slate units; actions have persistent captions below56px icons. Party rows retain large opaque backgrounds.
- Main/pause and journal share plain rectangular frames with little internal hierarchy. Journal is one vertical text stack, rather than a composed information window.
- Existing strengths: CommonUI stacks/focus/back/modal ownership, staged settings, remapped prompts, data/entry bindings, layout profiles, actual encounter actions and the independently reproduced Sakura center-origin cooldown. Preserve these.

Focused release matrix

Area	Retain	Change	Cost / proof
Third-person experience	Existing encounter and measured v022 actions	Orbit/zoom camera near character, camera-relative movement, look release on menu/popup; remove RPG right-click orders	Medium; real look/move/zoom, obstacle and panel ownership checks; verify RTS unchanged
Combat HUD	Models, input prompts, cooldown material, layout profiles	Compact action strip/vitals, quieter party/objectives, target/cast hierarchy, no player selection ring	Medium; 720p/1080p/ultrawide, enlarged text, tooltip and focus review
Styling	Theme/font/audio configuration	Original etched frame skins, inset surfaces, dividers and compact entries. Shared public surface-style keys instead of per-variation C++ subclasses	Low/medium; swap theme/style in Blueprint and inspect actual result
Windows	Functional settings, journal, confirmations and save/return	Consistent header/content/footer, section labels, close/back affordances, responsive reading width and spacing	Medium; complete playable menu/settings/journal/modal flow
Demonstration	Actual party/ability/quest data	Present the same working adapter more clearly	Low; repeated attack/cast/quest/party checks; no inventory, backend chat or new MMO simulation

Acceptance gates

1. The default RPG is immediately recognizable as third-person: visible character, usable forward view, orbit/zoom, camera-relative control. RTS retains its strategy camera/orders.
2. At1920×1080 normal scale, the central play region remains free of persistent opaque HUD; essential information fits a compact lower cluster and edge docks. Measure actual widget bounds; do not sacrifice text readability to hit an arbitrary percentage.
3. Every revised screen is a saved editable Widget Blueprint built from public UIForge controls. Theme surface styles, fonts, entry classes, positions and animation choices can be changed without plugin source edits.
4. Menus/settings/journal/dialogs have coherent detail and spacing, not oversized empty rectangles. Actions work and values change the actual example.
5. Preserve Sakura tint, v022 contact/grip/pose behavior, existing saves, core independence and the RTS. Build/cook/run on5.8.3, compile saved Blueprints, inspect real interactions/resolutions and record failures honestly. Physical controller/touch remain user-deferred; software events are labeled separately.

Implementation sequence: camera/input vertical slice → RPG-only theme/entries/HUD → full window pass → runtime/visual iteration → documentation and current packaged candidate. Use the existing bounded build hosts; preserve the before-rpg backup and prior signed receipts. Do not claim completion from this document.

Owner correction: combat hierarchy and binding regression

The owner rejected the lower target/player pairing. The new default places a dedicated target name, health and numbers at the top center. This is a product decision informed by the FFXIV target-bar documentation, not a claim that every MMO uses the same placement. The bottom center contains the player resource frame and six usable action slots. Avoid ornament repeating on every small readout.

The audit found two concrete regressions in the first RPG rework: its HUD CDO omitted the earlier mapping-aware Shortcuts array, and its health-bar authoring helper did not override Forge Progress's SquarePerimeter default to Linear. Right-mouse capture also needs an explicit look policy. Current verification must cover the actual InputKeySelector UI,

applied gameplay behavior, prompt changes, cancel/conflict handling and reloading saved bindings. Historical RTS rebinding evidence cannot substitute for these RPG checks.

GAME INTERFACES AND ADAPTERS

Army and RPG HUD authoring

This revision uses public UIForge features. **ForgeExamples owns the game rules.** An existing game supplies its own selection, units, combat, quests and authority through `IFDataSource`. Neither army simulation nor an inventory backend is a core plugin dependency. Follow game integration for project setup and Blueprint settings for the complete settings workflow.

The current revision has scoped runtime/package verification; see current evidence and limits and the rework record. Earlier Todo receipts describe the prior presentation.

Mixed HUD input

In your **IFScreen** Blueprint defaults:

1. Set **Input Policy = Game And UI**. **Game And UI Mouse Capture = Capture During Mouse Down** keeps the cursor visible and forwards ground presses to the game viewport. In UE5.8, `NoCapture` prevents the viewport from forwarding primary mouse-down events; it is unsuitable for world selection/right-click commands.
2. Add your movement/order **player-mappable row names** to **Gameplay Key Mappings**. These use applied Enhanced Input bindings, so a remapped camera arrow still pans instead of navigating a focused button. The plugin forwards key-down/up to the local controller once and releases held inputs when another screen takes ownership. Text editors and key selectors retain input; disable **Shortcuts Enabled** for a custom editor or open it on a separate screen.
3. Reserve only the gameplay axes needed in **Gameplay Analog Keys**. The examples use `Gamepad_LeftX/Gamepad_LeftY`; CommonUI keeps directional navigation for its other controls.
4. The example HUDs set **Show Keyboard Navigation Focus = false**. Keyboard hotkeys update prompts without opening a tooltip on a parked focus target. Gamepad input still restores visible navigation focus immediately. Menus/dialogs keep keyboard navigation focus enabled. Enable this property for a HUD intended for direct keyboard widget navigation.
5. Set **Initial Focus Name** to a nested control path, for example `HUDLayout/Order3/EntryCommand`; set **Fallback Focus Name = HUDLayout/Pause**. Set **Back Action Name = Pause**. Your adapter opens the pause screen and validates gameplay commands.

Do not set input modes repeatedly in Tick. CommonUI owns the active screen policy. Pause, settings and modal screens use **Menu**; world commands must stop while those screens own input. Physical controller and touch-device testing remains user-deferred/unverified; synthetic event coverage is recorded separately.

CommonUI owns focus restoration across stack activation. **Initial Focus Name** is the initial/fallback target, not a request to reset focus every time a submenu closes. Device-presentation refreshes reconcile cursor/style without replacing CommonUI's departing-control cache. Input actions are gated while a stack transition is in progress; wait until the new screen is active before issuing another navigation action.

A Blueprint roster with recycled entries

Create these objects in your game adapter using **Construct Object from Class**:

- One **IFPresentationData** (or your own compatible UObject) per stable unit/item/quest identity.
- One **IFCollectionData** for the visible selection, roster or quest collection.

Implement **Get UI Model** on `IFDataSource`: return the collection for the name `Roster`. When game selection changes, build an object array and call **Set Items** on that collection. Reuse each object's identity; do not recreate all objects when one health value changes. `Set Items` ignores nulls/duplicates and broadcasts only when membership/order changes. For health/title/count changes, edit the corresponding presentation object and call **Notify Data Changed** once.

Create a Widget Blueprint with parent **IFReadoutEntry**:

1. Add an engine **Tile View** or **List View** named **ItemList**, with **Is Variable** checked.
2. Set its **Entry Widget Class** to your entry Blueprint. The engine virtualizes the collection; scrolling recycles entries through the documented lifecycle.
3. On the collection widget instance, set **Model Name = Roster**. No game-specific C++ widget class is required.

Create the entry Blueprint with parent **IFReadoutEntry**. Add any of `TitleText`, `DetailText`, `ValueText`, `IconImage`, `FractionBar` and `EntryCommand`; all are optional. Set **Entry Action Name = SelectUnit**. The optional command button receives the current entry object as action context. Your adapter's **Request UI Action** validates that identity and performs selection. On a manually authored entry, **Set Command Context** provides the same behavior. Reset subscriptions/context in **Release Entry** before reuse.

The saved RTS example uses **WBP_IFArmyTile** and **WBP_IFArmyRoster**. Its portrait buttons isolate a unit; Shift-click removes that unit from the current selection. List membership is game selection, so its engine list selection mode is None. A different game can use engine single/multiple list selection and forward **On Item Selection Changed** to its adapter instead.

For an arbitrary User Widget, implement **IFEntry** and **User Object List Entry**, then implement **Initialize Entry**, **Refresh Entry**, **Set Entry Selected** and **Release Entry**. The optional IFEntryWidget base already connects the engine list lifecycle. Your Blueprint can override **On Entry Initialized**, **On Entry Refreshed**, **On Entry Selection Changed** and **On Entry Released**. See entry contracts for the full lifecycle.

Selection feedback without UI polling

Add **InterfaceForge Selection Overlay** to the full-viewport canvas of your IFScreen, outside scaled HUD regions. Set **Model Name = SelectionOverlay**. Return an **IFSelectionData** from your adapter for that name.

For each selected/targeted entity, use **Project World Location to Screen** on the owning local controller. Divide the projected center and projected radius axes by that viewport's dimensions; append an **IFSelectionMarker**. Its center and two axes describe a projected ground ellipse, so perspective and aspect ratio are preserved. Set **Color** for your semantic state and **Order Destination** for a destination cross. Call **Notify Data Changed** once after updating the batch.

For a drag rectangle, set **Marquee Visible**, **Marquee Start** and **Marquee End** using normalized viewport coordinates. The overlay is passive and hit-test invisible. Your game performs picking/selection and owns entity lifetime; the widget never scans actors or Mass entities. The example adapter bounds its population and prunes dead selection references. Its viewport math demonstrates one local player; split-screen-specific projection has not been certified.

Map interaction and camera footprint

An **InterfaceForge Map** binds to **IFMapData**. Supply normalized **Markers**, an optional baked **Background**, and optional **View Corners** for a camera footprint; batch updates with **Notify Data Changed**. No runtime SceneCapture is required.

Set **Location Action** for primary click/touch and **Alternate Location Action** for secondary click. The selected normalized location is available on the map data. The Blueprint nodes **Press Location**, **On Location Pressed** and **On Alternate Location Pressed** also allow your own controls. Ironfront uses primary click to recenter the camera and secondary click to issue a move/attack-move order. The adapter converts map coordinates to world coordinates. Touch currently uses the primary action; author a separate action for secondary touch gestures.

For the map's default navigation control, focus it, then press Enter/controller south button to edit the map cursor. Arrows/D-pad move it; Enter/south confirms the primary action, Space/controller west triggers the alternate action, and Escape/east cancels. Each confirmation leaves cursor editing, so directional focus navigation can leave the map. These are the default UI controls; gameplay hotkeys use Enhanced Input remapping. Device hardware has not been tested.

Themes, compact controls and semantic states

Use your project **IFTheme** for typography, colors, sounds and default button padding. Named **Button Styles** and **Surface Styles** now skin controls and shared Window/Header/Combat/Row surfaces; see style authoring. A dense command tile can set **Override Content Padding**, **Content Padding** and **Label Font Size** locally. A size of zero follows the theme; accessibility text scale still applies. This avoids copying an entire theme for one small control.

Use an **IFHUDLayoutProfile** to name, anchor and size regions. **Set Region Widget** replaces a region with your Blueprint widget; **Set Layout Profile** replaces composition. **Auto Height** measures live content at the authored width, allowing enlarged/wrapped text to grow a panel; `Size.Y` remains its Designer preview estimate. Put related rows in a Vertical Box, as the example party stack does, so they move together instead of overlapping at fixed offsets. Keep virtualized scrolling collections in a bounded height. The current WBP_IFArmyRoster keeps its five group controls and selection grid in one Auto Height region. Its Blueprint **Event On Entry Refreshed → Entry Data → Cast To IFCollectionData → Items → Length → (Count + 9) / 10 → Clamp (0,3) → ×50 → Set Height Override** gives the grid zero, one, two or three rows. The height stays bounded for virtualization; selection changes drive the graph, without Tick.

Minimum Canvas Size optionally caps the composition's scale so essential regions stay on screen. This fit cap may limit the requested overall HUD scale in a narrow viewport; text enlargement remains available. Menus/settings use their own responsive composition.

For a compact strip whose captions must grow with text accessibility, enable **Auto Height** and **Auto Width** on its region. Use ordinary Vertical Boxes and Size Boxes with a minimum width inside each entry; allow captions to determine their natural width. Emberfall's action strip uses this pattern instead of squeezing enlarged words into a fixed icon width. Fixed-width cards wrap their headings; scrolling army tiles keep bounded geometry. The RTS menu, production queue and commands share a vertical **OrdersDock**, so content growth cannot move them through one another.

IFActionPrompt → Label Font Size controls the base key-label size; the player's text scale still applies. Use **Refresh Prompt** after changing it at runtime. Buttons similarly expose **Label Font Size** and optional **Content Padding** for compact controls. These are local overrides, not duplicated theme assets.

An IFReadoutEntry can opt into **Use Model Accent** for a semantic fraction-bar color. The RPG target frame uses **Accent Color** from its model to distinguish allies from hostiles, while the surrounding theme stays consistent. Ordinary widgets can bind their own colors in **On Entry Refreshed**.

The command atlas is original SVG geometry in `ContentSource/UI/CommandVectors`, with a deterministic raster import. Change an icon texture/data source or **Icon UV Min/Max** in Blueprint. No icon generation service is required.

RPG presentation and actual actions

Emberfall's action controls send requests through `IFDataSource`. Its example adapter handles continuing attack, movement interruption, contact timing, costs/recovery and target lifetime. The plugin owns no ability rules. Party entries request inspection of the selected ally; the target frame reads the same observable data contract.

WBP_IFQuestJournal reads the live **Objective**, **QuestStory** and **TrackQuest** models from the encounter adapter. **Track** changes HUD tracking; **Return to Game** uses **Request Screen Back**. It does not increment an unrelated observation counter. Replace those models with your quest system without changing the screen layout.

The action icon uses **InterfaceForge Center Tint** over the icon. It retains Sakura's inspected center-origin interior angular mask and its `Percent`, `RemainingTimeOpacity`, `PassedTimeOpacity`, `StartAngle`, `Reverse` and `Color` parameters. It is not a square perimeter or tube effect. See center tint. Replace the texture, tint opacities and orientation through ordinary Blueprint properties.

For smooth live progress, call **Set Fraction (IFPresentationData)** with your authoritative normalized value. **On Fraction Changed** updates an IFReadoutEntry's bars/tint without rebinding labels, loading icons or rebuilding its collection. Use **Notify Data Changed** for grouped title/detail/availability edits. The example sends only active cooldown fractions each frame and other presentation changes at a lower rate; pausing its adapter freezes both. The original material mask and its timing semantics are unchanged.

Scope

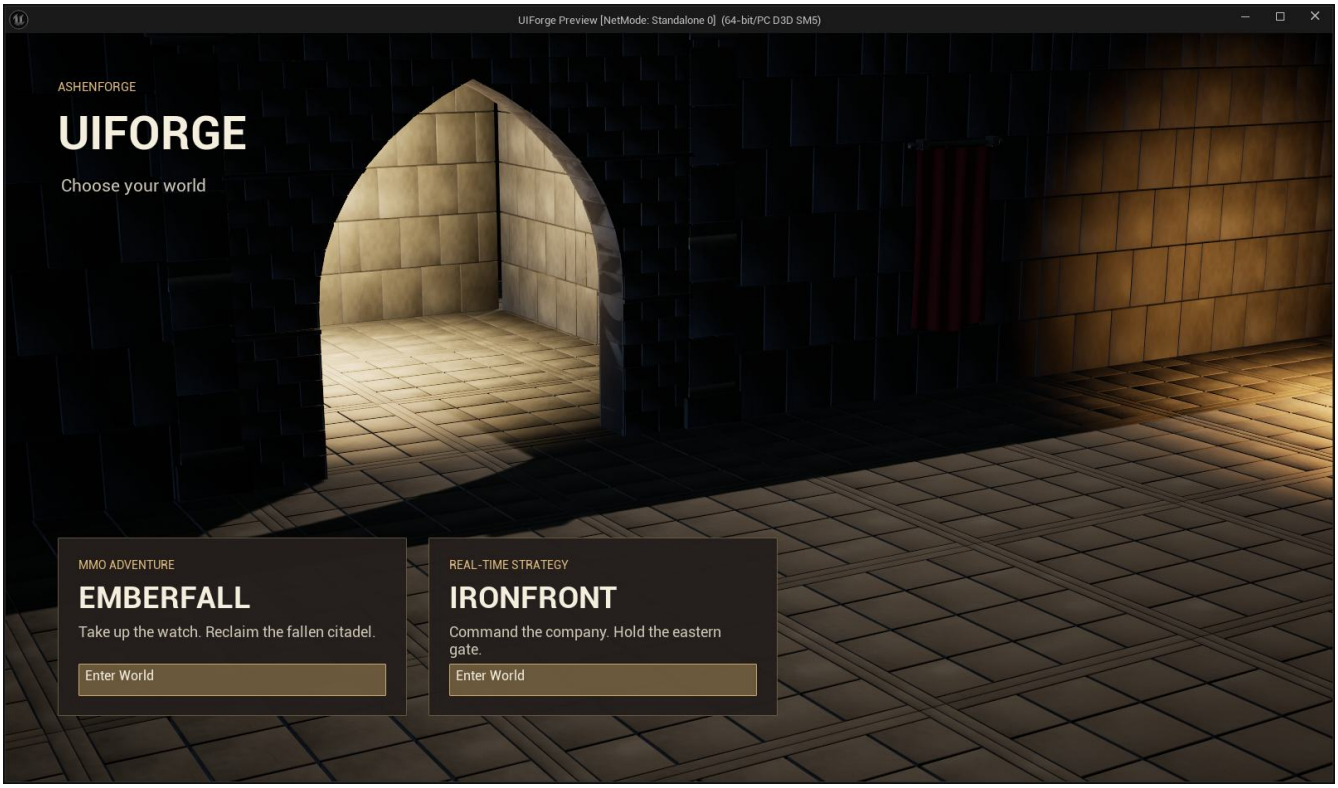
These are local MMO-style and RTS interfaces with a small responding game adapter. They do not implement online MMO services, an RTS economy, inventory or a multiplayer army simulation. CommonUI, settings, presentation, data contracts and Blueprint authoring are the UI product; the army/combat code is inspectable example integration.

GAME INTERFACES AND ADAPTERS

Example world presentation

The **2026-10-26 environment pass** updates the hub, Emberfall and Ironfront. These are saved, editable example levels. UIForge's runtime/plugin C++ and the encounter rules are unchanged.

Sideways walls, tipped/overscaled rocks, vertically stretched rubble and the disconnected gate ramp were removed. Upright wall runs now meet at measured edges. The bonded gate opens onto a paved, lit passage. Columns and braziers are grounded outside the army lanes. Rear closures, a low terrace edge, a stone foundation and a shared terrain backdrop complete the outer camera views. Soft fill preserves character readability without the initial washed-out highlights. Minimap images are rebaked from the actual geometry.



Revised example hub



Revised army courtyard



Completed outer walls and terrain at maximum RTS pan and zoom

Edit the environment

Open `/Game/ForgeExamples/Maps/ForgeGateway`, `Emberfall` or `Ironfront`. The **Environment** actor folder contains ordinary saved mesh/light actors. Replace those in the level; no plugin subclass is required. Keep the encounter actor's camera/UI references when changing scenery. The small demo's playable bounds remain unchanged; its decorative passage and outer wings are outside those bounds. Expanding gameplay belongs to the example adapter or your game.

Castle, knight and props are bounded AshenForge-owned MassForge art. The original `SM_IFCourtyardTerrain`, soil albedo/normal and their source/provenance are separate example assets. The terrain has12,800 triangles and no collision/navigation role. The two512px textures replace runtime procedural noise. One mesh/material is shared across all three worlds. The playable floor retains its navigation; the outer paving is decoration. No new example dependency is added to the UI plugin.

In the development checkout, `ContentSource/World` contains the deterministic generators and editable OBJ/PNG sources. The curated example archive includes OBJ/PNG/provenance/README files. The geometry and textures are code-authored; no external imagery or image-generation model is used. This is provenance, not a marketplace classification decision.

Verification scope

Saved map reload checks measure upright/grounded geometry and confirm old clutter is absent. All four legal RTS pan corners are visually reviewed at maximum65-degree zoom. Army selection/orders/recruitment and RPG attack/cast/interruption scenarios run again after navigation rebuild. Test waits account for the example's existing0.1-second simulation-step cap; gameplay time is not changed. The gallery includes the unchanged UI at720p,1080p and ultrawide, plus enlarged scaling.

The environment is verified on UE5.8.3 / Win64. The package and source-identity records for this pass are in the existing `Workspace/Rebuild583/WorldPolish` lane; previous ArmyHUD receipts remain historical. Hardware controller/touch, other platforms and isolated GPU/FPS certification are outside this environment-only pass. Current product verification distinguishes earlier UI testing from these new world checks.

GAME INTERFACES AND ADAPTERS

Example character animation

The MMO and RTS knights use MassForge's **HumanMotion_v022** infantry performances. The animation and encounter code belong to **ForgeExamples**, the separate demonstration game. InterfaceForge itself has no character, combat or MassForge dependency.

Editable assets

Open `Content/ForgeExamples/Animation/HumanMotion22`:

- **ABP_IFDemoCharacter**: ordinary editable Animation Blueprint, parent **ForgeDemoAnimInstance**.
- **BS_IFLocomotion**: idle/walk/run Blend Space. **Ground Speed** is planar displacement in cm/s, including crowd separation. These demonstration actors do not have CharacterMovement velocity.
- **A_IF_idle**, **A_IF_walk**, **A_IF_run**, **A_IF_melee_attack**, **A_IF_cast**, **A_IF_hit_reaction**, **A_IF_death**: seven locally saved sequences. No source FBX, VAT bank or MassForge runtime is required.

Blueprint workflow (text source; interactive diagram in the web guide)

```
flowchart LR
    Speed[Ground Speed] --> Blend[BS_IFLocomotion]
    Blend --> Slot[Slot: ForgeAction]
    Slot --> Pose[Output Pose]
    Action[Play Action: selected sequence] --> Slot
```

The Blend Space samples use the source bank's reference speeds: idle0, walk106.74186, run610.7501cm/s. Input smoothing is0.12s. Editing these sample points changes the gait mix; they do not change encounter movement speed. Review foot placement after changing either the mesh or movement speeds.

Replace a performance in Blueprint

1. Open **ForgeGateway**, **Emberfall** or **Ironfront**, then select its **ForgeDemoWorld** actor.
2. Under **Example / Art**, change **Character Animation Class**, **Idle Animation**, **Move Animation**, **Attack Animation**, **Cast Animation**, **Hit Animation** or **Death Animation**. The saved Animation Blueprint owns the three locomotion samples: edit **BS_IFLocomotion** to replace those clips.
3. Set **Melee Contact Time** and **Spell Release Time** to the new performance's actual contact/release seconds. Current values are **0.6333333** and **0.5**. Damage is validated at contact, rather than when the button is pressed.
4. Set **Mesh Forward Yaw** for the replacement mesh. The imported knight faces component+Y, so the default is90degrees. **Blade Socket Offset** is the measured v022 transform on `hand_r`; this is fixed presentation, with no equipment rules.
5. Play the map. Check forward travel, turns, stopping, attack recovery, casting, interruption and death. A new animation with a different skeleton needs a compatible retargeted sequence and Animation Blueprint.

For direct Blueprint presentation, obtain the mesh's **Get Anim Instance**, cast to **Forge Demo Anim Instance**, then call **Play Action** with a sequence. **Hold Final Pose** keeps a death pose after its final frame. **Return To Locomotion** blends the action out. The encounter adapter calls these nodes' C++ equivalents; it owns damage, orders and life state separately.

Actions use Unreal's **ForgeAction** slot in **ForgeActions**, with0.09s blend-in and0.12s blend-out. **Action Blend In** and **Action Blend Out** are editable in the Animation Blueprint Class Defaults. Montages are cached per character/used clip and reused for subsequent actions. Moving or attacking characters retain their committed action when hit. The short hit reaction is shown only when the receiver is idle, avoiding repeated full-body recoil overwriting the strike or sliding the feet. Pause freezes the mesh animation as well as encounter damage. Animation advancement follows the encounter's capped simulation step when an unfocused editor throttles.

These are standard Animation Slots and Animation Montages. Developers may replace the example Animation Blueprint with their own graph; character animation is not a requirement for displaying gameplay through UIForge.

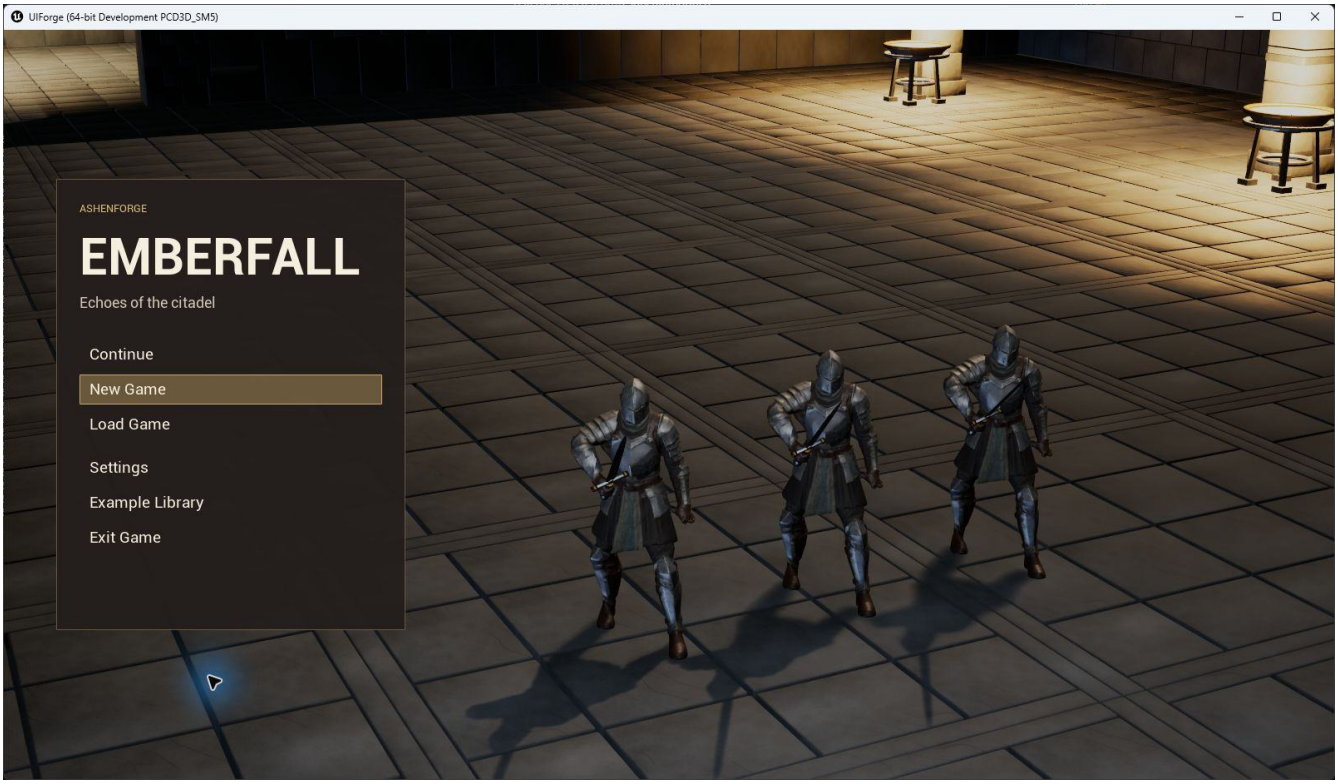
Provenance and limits

The new motions are AshenForge's edited/retargeted **Quaternius CC0 Standard Universal Animation Library** performances. They are not exclusive original motion capture. The separate example ships the acknowledgement and source hashes in `ArtSource/Animation` (development source: `ContentSource/Animation`). Character geometry and ownership are unchanged. Only seven skeletal clips were selected; jumping, falling, stun and MassForge VAT assets were not added to the UI demonstration.

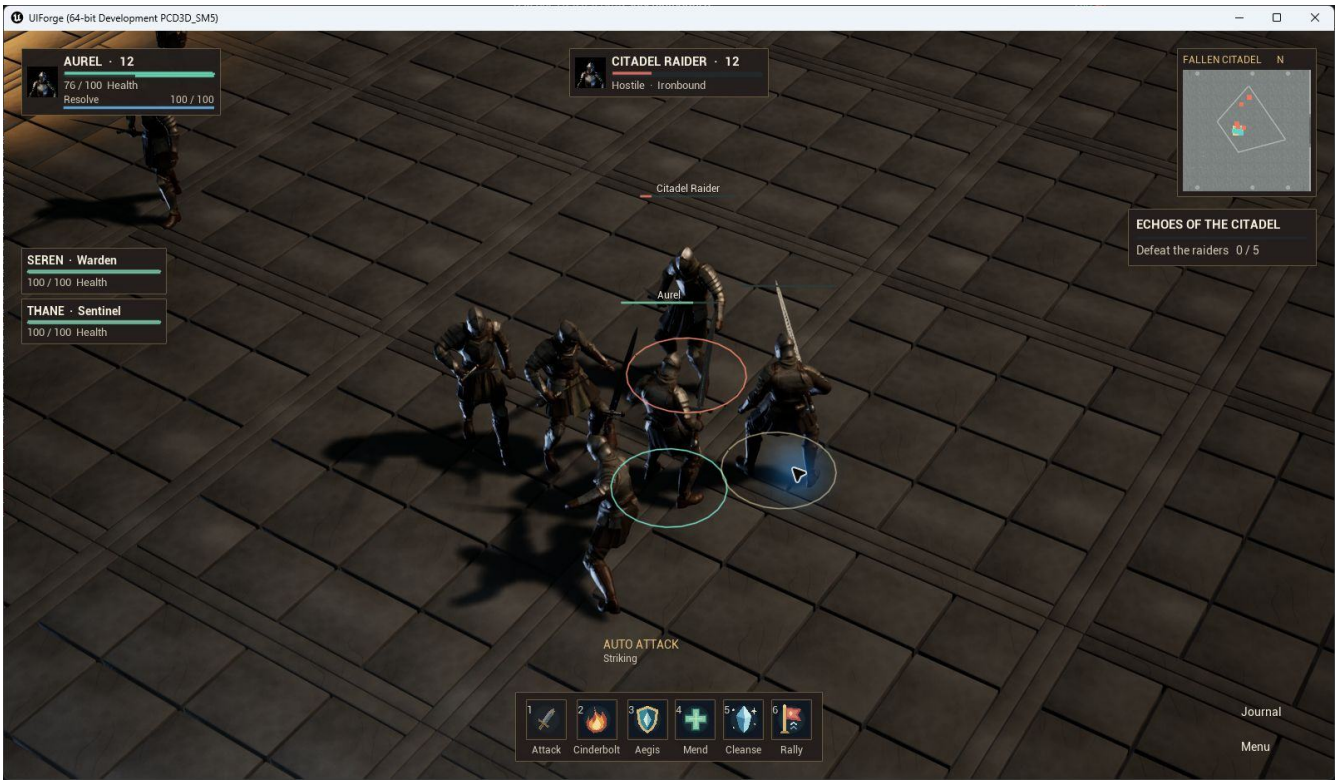
Runtime verification records for this revision are under `Workspace/Rebuild583/Animation22` outside the source distribution. Verification includes sampled source-pose parity, actual movement/facing, actions, pause and the existing RPG/RTS gameplay scenarios. Hardware controller/touch tests remain user-deferred; no animation change establishes network or cross-platform certification.

Packaged runtime views

These are native Win64 Development game windows captured after real input, not pose mockups.



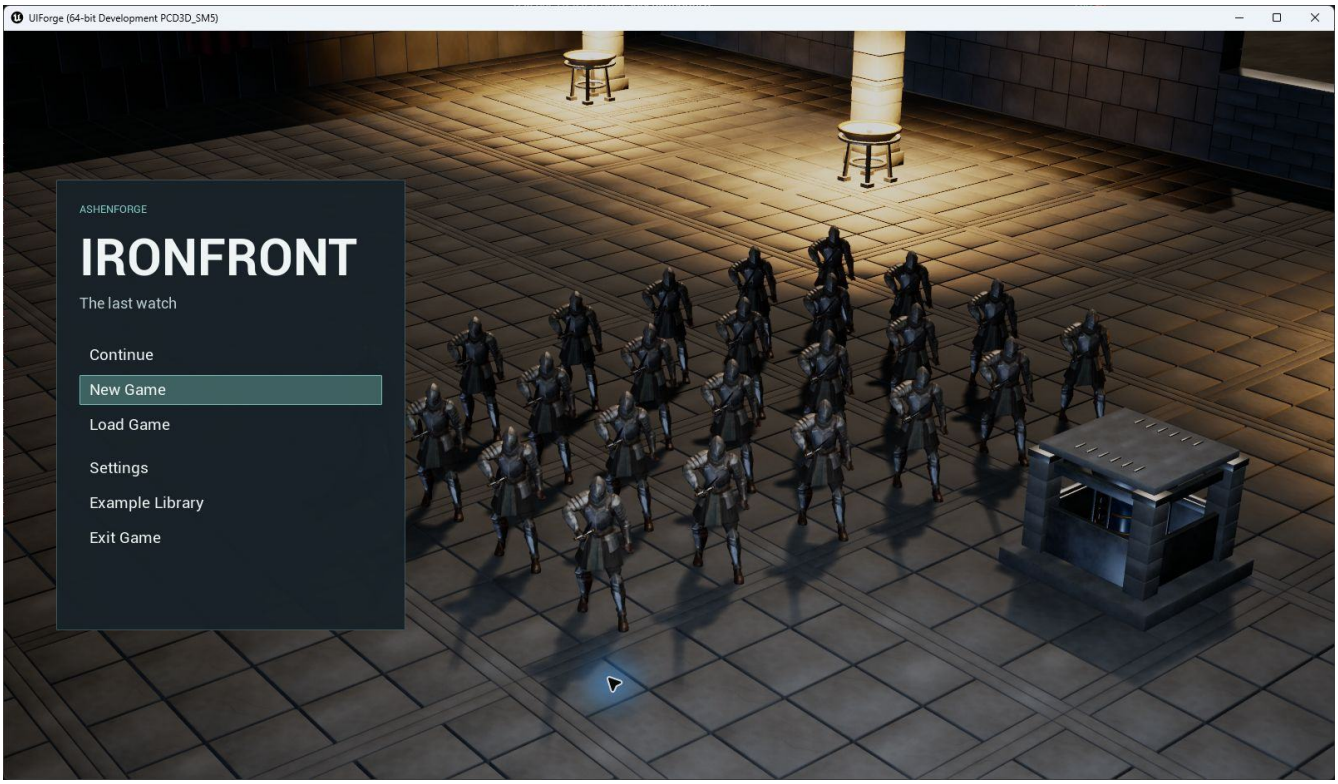
RPG menu with v022 idle staging



Repeated real RPG attacks and health changes



Cinderbolt input activates cast presentation and cooldown



RTS menu with the updated army presentation



Selecting24 and issuing a formation move



The same packaged army encounter reaches victory with deaths and surviving soldiers

GAME INTERFACES AND ADAPTERS

Blueprint RTS commands and selection

Historical example recipe. The current UE5.8.3 Ironfront/Emberfall interfaces and their public Blueprint workflows are in army and RPG HUD authoring and the current gallery. Earlier paths and engine references below describe the retained earlier example.

ForgeOutpost uses an original InterfaceForge layout and presentation. Game Master's design, widgets, artwork, styling and animations stay reference-only. The example's small local Outpost provider owns selection and command rules; InterfaceForge owns presentation, navigation, input prompts and model/entry lifecycle.

Run `/Game/Maps/ForgeOutpost` and choose **Begin expedition**. **Next scout** selects one scout. **Select all scouts** selects the current group and becomes **Select one scout**. Clicking a scout replaces selection; Shift-click toggles it. Selected scout bodies use mint, with gold for unselected bodies. The UI buttons provide selection without a pointer. The remapping-aware prompt still resolves `IFExampleInteract` for Next scout.

Patrol moves the selected group to the beacon, updates the group summary, completes the objective and earns 40 supplies once. **Rally selected scouts** moves only that selection. **Train scout · 60** starts a four-second production queue and adds a scout. Busy commands stay focusable, display their reason, and reject activation. Pause freezes the example's movement/production clocks. Return to main releases the HUD's subscriptions. This local demonstration is not a multiplayer RTS simulation or a MassForge implementation.

Create a command entry in Blueprint

The saved **WBP_OutpostCommand** derives from **IFEntryWidget**. It contains **CommandChoice**, a **WBP_OutpostCommandButton** deriving from **IFButton**, and an Object Reference variable **ActionSource**. The button's ordinary Designer tree has a Border named `FocusBorder`, and IFText components named `LabelWidget` and `AvailabilityWidget`. IFButton supplies theme, input/focus feedback and availability presentation to these optional components.

The button's **On Appearance Updated** graph casts the inherited Text Block references to **IFText**, sets their **Theme Override** from the event's Theme, and calls **Refresh Appearance**. Label uses Base Size 12, with Body when Enabled and Caption when unavailable; the reason uses Base Size 10 and Caption. This public hook preserves player text scaling, focus feedback and local/global theme changes while giving dense commands a local typography choice. The cast is necessary because IFButton's inherited component references are typed as Text Block.

Use these graphs:

Event	Nodes and data	Expected result
On Entry Refreshed	Cast Entry Data to IFPresentationData → Set Label on CommandChoice, Text = Title → Set Action Availability, Available = bEnabled, Reason = UnavailableReason	Initial state and subsequent model updates change the same button
CommandChoice: On Button Base Clicked	Cast Entry Data to IFPresentationData → Request UI Action (IFDataSource interface message), Target = ActionSource, Action Name = StableKey	External provider validates and responds to the command
On Entry Released	Set ActionSource = None → Set Label to empty → Set Action Availability(false, empty) → Cancel Latent Actions	Reuse cannot invoke a previous provider, retain its label/reason or complete an old deferred reveal

The base entry binds `OnDataChanged` and unbinds the old model before release/reassignment. It does not know the meaning of Patrol, Rally or Recruit. A developer can use this same entry for another game by supplying a compatible model and IFDataSource object.

After updating availability, **Delay Until Next Tick** → **HasAnyUserFocusedDescendants** → **Branch** checks whether this entry owns focus after input activation finishes. **Get Parent** → **Is Valid** → **Get Parent** → **Cast to ScrollBox** → **Force Layout Prepass** → **Scroll Widget Into View** requests a reveal with Widget to Find = Self, Animate Scroll = false, Scroll Destination = IntoView and Padding = 4. This sample's entry sits in a Vertical Box inside a Scroll Box. Missing parents or a different container skip the reveal. The deferred focus check and fresh measurement let a newly visible reason fit inside the scroll viewport. This is a bounded response to a model update, without per-widget polling. Entries without focus never pull the user's scroll position. Release cancels pending latent actions.

Compose the HUD

In **WBP_AshenRTSHUD** → **Initialize Screen**, call **Parent: Initialize Screen**. Initialize the existing readouts from Resources, Selected, Objective, Queue and Notice. For each command entry, call **Get UI Model** → **Initialize Entry**, then **Set ActionSource = Screen Data**. Set the source **after** Initialize Entry: initialization first releases an old assignment and the release graph clears ActionSource.

Designer entry	Get UI Model name / StableKey
NextChoice	SelectNext
GroupChoice	ToggleGroup
PatrolChoice	Patrol
RallyChoice	Rally
TrainChoice	Recruit

`Selected.Quantity` is the selection count. Title describes a single scout, group or empty selection; Detail summarizes moving and waiting scouts. Each command model supplies its own current label, availability and reason. Edit the existing stable model, then call **Notify Data Changed** once after a group of edits. No widget scans the world population.

Blueprint workflow (text source; interactive diagram in the web guide)

```
flowchart LR
    A[HUD Initialize Screen] --> B[Get UI Model]
    B --> C[Command: Initialize Entry]
    C --> D[Set ActionSource = Screen Data]
    E[Provider edits command model] --> F[Notify Data Changed]
    F --> G[On Entry Refreshed]
    G --> H[Set Label and Set Action Availability]
    I[Button activation] --> J[Request UI Action interface message]
    J --> E
```

Event On Initialized → **Set Initial Focus** targets `PatrolChoice.CommandChoice`. **Event Construct** → **Set Navigation Rule Explicit** connects the inner buttons in the order Next → Group → Patrol → Rally → Train → Pause, with inverse Up links. Construct runs after the Slate widgets exist, so cross-tree navigation resolves the actual buttons. Standard Scroll Boxes reveal focused commands. `RTSSelectionDock` keeps SelectionReadout above the command Scroll Box; that Scroll Box uses a Fill vertical slot. Scrolling to a command retains the selection summary. At small viewports with enlarged UI/text, the command panel deliberately scrolls rather than overlapping the world or shrinking the chosen text size.

The HUD sets its existing readouts' `ReadoutTitle.BaseSize = 12` and `ReadoutDetail.BaseSize = 11`, followed by **Refresh Appearance**, before assigning data. This local Blueprint choice keeps passive information readable at 800×600 with both UI and text at 150%, while retaining the common readout classes, theme and player scaling.

Customize and verify

Duplicate either command Widget Blueprint, change its Designer tree and refresh graph, and choose the new class for a HUD entry. Retain the inner IFButton for CommonUI navigation or explicitly link your chosen focus targets in Construct. Change ActionSource or model keys to consume another IFDataSource; selection rules remain in that provider. Replace themes through **Set Theme**, or set a widget's **Theme Override** for a local change.

The opt-in Development fixture `-IFVerifyRTS` uses the actual saved Blueprint HUD and synthetic Slate Enter/arrow events, while its setup, pause/return and actor-lifetime controls are programmatic. It tests group selection, observable movement, busy rejection, pause/resume, production, selection-only rally, stale actor cleanup, nested navigation and release subscriptions. `-IFVerifyRTSEnlarged` previews 150% UI/text without saving. `-IFVerifyRTSCapture` captures the rendered training state. Exact revisions and results are in `Evidence/RTS_SELECTION_RUNTIME.md`.

Physical mouse/controller/touch tests of this revision, drag selection, production cancellation, large entity populations, multiplayer and the optional Mass adapter remain separate gates. Do not infer them from the local synthetic-input fixture.

Compact overview and visible production

The original passive card uses **RTSPassiveFit (Overlay)**, Fill/Bottom inside the existing .55-weight lower dock. Its existing Size Box/Surface follows content, with padding14 and retained bounded scrolling. **RTSOverviewColumn** pins QueueReadout and the existing remap-aware IFActionPrompt below the resource/notice Scroll Box. Queue title/detail use per-instance **Base Size12 → Refresh Appearance** in Construct; its model and the shared readout class are unchanged. Command entries keep48-unit minimum targets,6-unit gaps and no trailing gap. All five labels fit at default1280×720; at enlarged800×600, navigation scrolls commands and passive prose while production/prompt remain pinned. These are normal Designer/Blueprint properties.

Both HUDs share the public navigation hint host. The body shows the command description in a reserved .28→.46 lane and yields to result notifications. It uses the same source model and public entry refresh/release contract. Pointer popups and the action provider remain independent. The examples retain original styling; Game Master's design is not redistributed.

GAME INTERFACES AND ADAPTERS

Emberwatch: an RPG interface through public features

Historical example recipe. The current UE5.8.3 Ironfront/Emberfall interfaces and their public Blueprint workflows are in army and RPG HUD authoring and the current gallery. Earlier paths and engine references below describe the retained earlier example.

Open `/Game/Maps/ForgePilgrimage` in the standalone UE5.6.1 project and Play. **Begin journey** → **Interact** → **Carry the embers** → **Approach** → **Kindle** completes the small scene's objective. **Ward** restores the two demonstration vitality displays and shows a timed status. Pause offers the same settings, journal and return confirmation used by the RTS example. The example has its own original layout, frames, vector icons and primitive scene. Game Master's bespoke visual design remains exclusive to Game Master.

The backend is `AForgePilgrimage` in the example project. It implements **IFDataSource** and owns these local demonstration actions. It is not a combat, ability, inventory or multiplayer implementation. InterfaceForge contains no reference to this actor or its controller. A project can replace it with a Blueprint object implementing **Get UI Model** and **Request UI Action**.

Compose the HUD

WBP_EmberRPGHUD, **WBP_EmberAction** and **WBP_EmberActionButton** are saved, editable Widget Blueprints under `/Game/UI/Blueprint`. Their graphs use public plugin and standard Unreal nodes. The authoring editor module is not required at runtime or to edit these assets.

Model name	Presentation in this example	Updates
FrontEnd	Shared main menu Title/Detail	Branding supplied by this data source
Hero / Companion	Reused WBP_ProductionReadout	Title, vitality Detail and Fraction
Quest	Reused WBP_HUDReadout	Accepted, current task and completed text
Context / Notice	Context panel	Interaction target and action feedback
Status	Timed party status	Title and remaining seconds
Kindle / Ward / Step	WBP_EmberAction	Stable Key, Title, Fraction, Quantity, Enabled, Unavailable Reason

`Event Initialize Screen → Parent: Initialize Screen → Get UI Model → Initialize Entry` binds each readout. Stable model objects broadcast **Notify Data Changed**; entries refresh without scanning actors. For each action entry, set **Sigil Style** before **Initialize Entry**, then assign **Action Source** from the screen's Data afterward. Initialize Entry first releases the previous binding, and the custom release graph clears Action Source. Assigning the source before initialization would erase it.

On Entry Refreshed → **Cast to IFPresentationData** drives these public nodes:

- **Set Label**, using Title, on AbilityChoice.
- **Set Remaining Fraction**, using Fraction, on CooldownTint.
- **To Text (Integer)** → **Set Text**, using Quantity, on CountdownText.
- **Set Action Availability**, using Enabled and Unavailable Reason, on AbilityChoice. The entry remains navigable during cooldown; the provider also rejects unavailable commands.
- **Get Dynamic Material** → **Set Scalar Parameter Value**, with Parameter Name `Sigil`, using Sigil Style on SigilImage. Get Dynamic Material has execution pins; connect them.

- **Quantity > 0** → **Branch** → **Set Visibility** shows the tint/countdown plate during cooldown and collapses them when ready.

On Button Base Clicked → **Cast Entry Data** → **Request UI Action (Message)** sends Stable Key to Action Name and Action Source to Target. The provider validates the request. This is the same interface a developer-created Blueprint entry can call. The example rejects Approach before accepting the keeper's request and after arrival, preventing an unreachable quest and repeated teleporting. This rule belongs to the example adapter.

On Entry Released clears Action Source, label and countdown text, resets the tint fraction, collapses its layers and disables interaction. The base entry unbinds its model. Every subsequent initialization supplies new data before interaction.

The inspected cooldown behavior

The icon has a 80×80 outer frame, 9-unit interior padding, and an Overlay containing the original SigilImage, **InterfaceForge Center Tint**, and a countdown plate. The tint covers the interior above the icon; the frame stays clear. Its boundary meets at the center. As remaining fraction decreases, the clearer region opens clockwise from the top. This independently authored material reproduces the inspected Sakura behavior; no source icon, frame, material graph or package is redistributed.

Public control	Meaning
Set Remaining Fraction	1 starts covered, 0 uses passed-region opacity throughout
Remaining Opacity / Passed Opacity	Defaults .75 / .15, matching the inspected action-bar instance
Tint	Overlay color
Start Angle	Turns clockwise from the top; .25 starts at the right
Reverse	Reverse the opening direction
Refresh Tint	Apply changed Designer/property values to this image's own MID

A plain Image may use `/InterfaceForge/UIForge/Materials/M_IFCenterTint` directly. Material parameters are `Percent`, `PassedTimeOpacity`, `RemainingTimeOpacity`, `StartAngle`, `Reverse`, and `Color`. Supply normalized remaining time through Percent; the UI material starts no timer and knows no ability system. The example provider advances one pause-aware clock, broadcasts changed fractions and ceil-rounded seconds, and freezes while a menu or modal covers the HUD.

The example-only `/Game/Art/M_EmberSigil` has `Sigil` (0 ember, 1 ward, 2 approach) and `Color`. **On Appearance Updated** → **Get Dynamic Material** → **Set Vector Parameter Value**, Parameter Name `Color`, applies **Select Color** (Theme.Accent when Enabled, Theme.Muted Text otherwise). Only the icon changes color; the tint's .75/.15 opacities remain unchanged. Replace this Image's material or brush in Blueprint to use your own artwork.

Explain unavailable actions

Set Action Availability (Available, Reason) blocks activation through CommonUI while keeping the button in keyboard/controller navigation. It does not validate gameplay rules. The example provider remains responsible for accepting or rejecting **Request UI Action**.

Add a Text Block named **AvailabilityWidget** to a custom IFButton tree to display the reason automatically. The name is an optional binding, like LabelWidget and FocusBorder. The reason is visible for every input family, needs no hover or failed click, follows the theme/text scale, and collapses when available. Custom trees may instead read **Unavailable Reason** or handle **On Appearance Updated**. That event's Enabled value now represents actual interaction availability, including CommonUI's interaction-enabled flag.

The original RPG entry reserves 168 units of minimum height for stable action targets and allows longer translated reasons to grow. Its examples show **Speak with the keeper first**, **Recovering**, and **Already at the hearth**. After acceptance/cooldown completion the reason clears and the icon returns to the accent color. **On Entry Released** → **Set Action Availability(false, empty Reason)** clears the previous reason before recycling. For tint-bearing buttons, keep CommonUI's **Apply Alpha On Disable** disabled so unavailable styling cannot silently halve your configured material

opacity.

The original action-button appearance graph ends with **Get Focus Border** → **Set Padding** (Left/Right 8, Top/Bottom 4). This overrides the theme's roomy menu padding for a compact HUD control without duplicating a theme or editing the plugin. **Focus Border**, **Label Widget**, and **Availability Widget** are optional, read-only Blueprint component references in **InterfaceForge | Designer**. Check that they exist before accessing them in a different tree. The appearance event runs after the native defaults, so a custom Blueprint can apply its own component presentation there.

The action dock's Canvas slot spans anchors **(0, .54)** → **(.65, 1)** with margins **(24, 0, 12, 24)**. This gives enlarged action cards enough vertical space for the reason. Keep both the desired card height and the available pane height in mind: focus scrolling cannot display an entire card when the card itself is taller than the pane. Use standard Designer anchors/padding, wrapping and Scroll Box settings to adapt your own HUD.

The fixed160-unit entry uses **Wrap Text At144** on AvailabilityWidget alongside automatic wrapping. Source1280×720 lock/unlock/recovery/arrival interactions passed. The explicit-width repair now passes actual desired/allocated bounds at measured800×600,1280×720 and1920×1080 with150%UI/text. Source and cooked800×600 offscreen DX11 captures show the full reason. Cooked native interaction retest remains open. See `Evidence/ACTION_AVAILABILITY_RUNTIME.md` for exact revisions; this limited layout proof is not release readiness.

Compact passive readouts without new classes

The RPG uses the same **WBP_ProductionReadout** for both party members and **WBP_HUDReadout** for the objective. In **Initialize Screen**, after **Parent: Initialize Screen**, it gets each instance's **Readout Title** and **Readout Detail**, sets the InterfaceForge Text **Base Size** to12, and calls **Refresh Appearance**. The ordinary model initialization chain follows. These local instance changes leave the RTS and journal's typography intact; no new theme or readout subclass is needed. Later theme/text-size events use the configured Base Size.

The party and quest surfaces use bottom anchor Y=.36 with the same margins. The example adapter publishes **LYRA ·72%** and **ORIN ·86%** in Title, leaves redundant Detail empty, and supplies Fraction for the existing progress bars. The percentage and bar both change when Ward restores vitality. Your adapter can supply different names and values through the same model contract. This is presentation data, not a plugin-owned health system.

The saved header hierarchy now content-fits these surfaces inside **PartyHeaderFit** and **QuestHeaderFit**, ordinary Overlay widgets. Their Canvas slots retain the .36 bottom anchor as available space; the surface's Overlay slot has Horizontal Alignment **Fill** and Vertical Alignment **Top**. Surface padding is14. The existing clipped Scroll Box and readout column remain inside. This gives short content only the height it needs and retains bounded scrolling for longer content. It does not scale down semantic text or replace the model graphs.

You can recreate this in Designer: wrap the surface in Overlay, move its previous Canvas anchors/offsets to that Overlay, and set the surface slot to Fill/Top. Change padding or anchor bounds for your own layout. Keep **Initialize Entry**, **On Entry Refreshed** and **Notify Data Changed** as documented above; layout does not change the binding contract. Responsive recipe and runtime evidence contain actual measurements. Initial English party/objective content fits at both tested scales; arbitrary later-state/localized text and the lower HUD's broader polish remain open.

Maximum-scale structural checks fit both complete party rows and the initial objective at measured800×600/1280×720/1920×1080. Source and cooked800×600 DX11 captures were visually reviewed with both percentages, bars and the complete objective visible. Later quest states, longer localization and cooked interaction remain separate checks.

Customization and navigation

Change the screen's Designer anchors, entry classes, images, or On Entry Refreshed graph. Use **Set Theme** or a widget Theme Override for semantic colors, typography, spacing and audio. The shared main menu consumes optional FrontEnd Title/Detail; the example supplies an intentional two-line `EMBER / WATCH` title. Long unbroken titles also allow per-character wrapping through ordinary Text Block properties.

Bounded Scroll Boxes use **Instant Scroll**. The action region uses **Navigation Destination = Bottom or Right**, keeping its label and unavailable reason visible when the whole tall card cannot fit at enlarged text size. Context panels retain **Center**.

Event Construct → **Set Navigation Rule Explicit** links connect Kindle ↔ Ward ↔ Approach ↔ Interact, and Interact ↔ Pause vertically. Apply links in Construct after the nested widgets have their Slate targets; On Initialized is too early for these cross-tree links. This allows a directional move to reveal a button outside the current scroll view. At 800×600 with 150% UI and text, the action row wraps; some passive text and the upper portion of a tall card are scrollable rather than simultaneously visible. InterfaceForge Text defaults to standard character wrapping for long words; its normal Designer wrapping settings remain overridable. Native IFActionPrompt's text occupies a Fill slot and wraps long labels. Developers can provide their own prompt class using **On Prompt Updated** and the resolved key/glyph/text.

The saved WBP_EmberRoot chooses the same result body as WBP_ForgeRoot with a different public Notification Dock. A reserved lane separates party/objective panels from the lower action/context area, which begins at Y=.54. Blueprint notifications documents actual Ward/hearth results, owner cleanup, class replacement and current test limits.

`WBP_EmberActionButton` also reveals feedback that grows **after** an action, without requiring another focus move. **On Appearance Updated** runs a **Sequence**: the first path applies its existing appearance immediately; the second checks **Navigation Focused** → **Delay Until Next Tick** → **Has Focused Descendants** → **Find Parent Widget Of Type** (Starting Widget=self, Type=ScrollBox) → **Cast To ScrollBox** → **Force Layout Prepass** → **Scroll Widget Into View** (Widget To Find=self, Animate Scroll=false, Scroll Destination=Bottom or Right, Padding=4). Connect the execution pins on Find Parent Widget Of Type; it is callable rather than pure. A missing parent exits through the cast failure. This standard Blueprint graph follows CommonUI's parent lookup across nested widget trees and remeasures the changed caption before revealing it. It preserves the entry class, model and command routing. Native pointer/touch interactions and interrupted/paused deferred updates still need broader testing.

Keep context actions reachable

The latest saved context surface contains **ContextReadoutDock**, a Vertical Box. Its clipped **ContextReading** Scroll Box takes the Fill slot and holds Context/Notice/interaction prompt. **PinnedContextActions** below it holds Interact and Pause, separated by8units. These controls remain reachable while longer prose scrolls. The surface uses12units padding, with8units below the reading region. This uses standard Designer layout features and the existing public buttons.

Event Construct → **Sequence** retains explicit navigation on its first path. Its second path sets ContextReadout and NoticeReadout's **Readout Title / Readout Detail** → **Base Size=12** → **Refresh Appearance**. The adapter supplies compact **KEEPER/HEARTH** context names, keeping descriptive story text in dialogue/objectives. A smaller font alone did not make the original wrapped title fit at800×600150%; the current full-title and pinned-control bounds assertions pass in source and the exact package. See loading/arrival evidence.

Begin now calls the public async loader through the separate example adapter. Ready initializes the same Blueprint HUD with its provider; cancel/failure preserves the main menu, and the saved failure dialog offers return/retry. No gameplay starts before Ready. Blueprint loading explains replacing the body and using the actual node present in the shared Settings click graph.

The same common input presentation, screen input policy, focus and settings services run in both genres. Mouse/keyboard automation is runtime evidence; physical controllers, touch and non-Windows platforms remain unverified. See the verification records for the tested revision and remaining gates. This is a development example, not a finished release.

Bottom action dock and status layout

The saved **ActionDockFit** Overlay retains the responsive action-region Canvas slot and content-fits **ActionDockColumn** with Fill/Bottom alignment. **StatusReadout** is an Auto first child outside the scrolling action region; the original Scroll Box is the Fill second child. Its existing ActionColumn/Wrap Box/entry hierarchy and Construct navigation links remain. Cards use Width Override160/Min Desired Height168/no fixed height. Long reasons grow and scroll.

StatusReadout enables the shared entry's **Presentation** → **Inline Detail** option. Its Title/Detail use Base Size12/Refresh Appearance; normal quest/RTS readouts keep stacked mode. This ordinary editable Blueprint choice is documented with actual nodes in Readout layouts. Configure it before construction; a live Boolean toggle alone does not run a layout-refresh event.

Current5.6 source and exact fresh cooked checks at1280×720/defaults and800×600/150% UI/text verify the whole focused Ward icon/tint/countdown, recovery reason and pinned party status. Synthetic Down reaches Pause, Up returns to Interact, Left reveals Approach and its complete unavailable reason. A tall Approach card can still scroll its upper icon away at enlarged settings. Action runtime evidence also records14complete/scoped current results and the failures that drove these bounds checks. Physical devices, broader tooltip placement, long translations and final HUD/scene polish remain open.

Descriptions without covering cooldowns

The saved HUD registers **NavigationHintHost** through the public screen node and uses **WBP_ForgeContextHint** for keyboard/controller descriptions. Its centered, content-fitted Surface stays inside the .36→.54 lane. Result messages take reading priority; while Ward's result is active, the hint is hidden and the full cooldown/status stay unobscured. After the result, navigation descriptions return. Pointer hover uses the original customizable popup; blocked actions can now show it through native outer-widget mouse events without enabling activation. Host recipe gives the actual nodes and input/lifetime contract. No change to the inspected center tint, material parameters, icon masking, countdown or provider clock.

Current glyph revision: the example opts into original neutral CommonInput face-position art; remap Apply refreshes IFActionPrompt in place, and empty Action Label shows only a glyph/readable key. RPG header keeps the cue visible beside scrolling prose with a.62shared divider. Current29Blueprint source/archive runtime proof is5.6; the newer28-class sample proof above predates it. Glyph authoring recipe and exact runtime/limitations. Physical controller tests remain explicitly deferred.

The context/action docks also use normal editable Canvas anchors to retain the enlarged interaction heading inside a reduced Safe Zone. Narrow action areas can wrap into a scrollable column; focus reveals Ward and preserves its recovery/tint feedback. Safe-area authoring documents the current layout and verification scope.

GAME INTERFACES AND ADAPTERS

Optional MassForge entity presentation

Historical adapter boundary: the receipts below cover MassForge0.2 and their recorded core revision. The current TODO delivery/source candidates exclude this optional adapter and do not certify it against the latest MassForge release. The current core is separately verified without MassForge; these recipes remain a reference for a future adapter verification pass.

InterfaceForgeMass is a separate integration plugin under `Integrations/InterfaceForgeMass`. It requires InterfaceForge and MassForge; neither the core UI plugin nor default Outpost/Emberwatch project requires it. The inspected MassForge0.2 API and UE5.8.3 Win64 are the tested integration boundary. Eight current source/exact Development-package reports pass with both providers at normal/enlarged scale, plus native lifetime tests and two clean saved Blueprint compilations. Runtime evidence records exact commands and limits. Physical controller tests remain deferred; no multiplayer or older-engine integration claim is made.

Show a represented actor or actorless entity

1. **Construct Object from Class** → **IFMassEntitySource**, using your owning screen/controller as Outer. Retain it in a variable while the view is open.
2. Make a **MFAttributeId** with Domain `Vital`, Name `Health` (or your project's actual attribute). Call **Observe Actor** with the player/world context, represented Actor, attribute, display Title and Fixed Maximum. Its result is MassForge's actual representation result: Success, Different World, Not Represented, Stale Representation, and other explicit failures.
3. For actorless entities, call **Observe Entity** with a **MFEntityHandle** obtained in the same world. The handle contains Index and SerialNumber, but no world identity. Always supply the originating world; a copied bare index or handle from another world is unsafe.
4. Call **Get Entity Model** → **Initialize Entry** on a normal custom IFEntry/IFEntryWidget. Alternatively **Get UI Model**, name `Entity`, returns the same stable presentation object through IFDataSource. **On Entry Refreshed** reads Title, Detail, Fraction, Enabled and Unavailable Reason with your usual public widgets.
5. If your schema has a maximum attribute, set **Maximum Attribute** before observing; Name `None` uses Fixed Maximum. A configured but missing maximum produces an unavailable state, rather than silently inventing a valid reading.
6. On screen cleanup/selection removal, call **Stop Observing**, then release the entry and your retained source reference. The registry also uses weak references, so it does not keep discarded sources alive.

Blueprint workflow (text source; interactive diagram in the web guide)

```
flowchart LR
    A[Game selection or actor] --> B[Observe Actor / Observe Entity]
    B --> M[Stable IFPresentationData]
    F[MassForge attribute events] --> C[World registry: coalesced dirty handles]
    C --> M
    M --> W[Your Blueprint IFEntry]
    W -->|Release| R[Stop Observing]
```

The registry has a hard512-source bound. Native attribute changes mark only watched entity identities dirty, coalescing repeated changes. A shared half-second lifetime check reads only registered handles; no widget ticks or whole-population query are introduced. Reads that report **Mass Is Processing** retain the previous fraction and expose a pending state, then retry in a later safe view tick. Destroyed/stale generations disable the retained model. Index reuse never retargets an old source automatically; explicit rebinding is required.

Forward actions without implementing gameplay

The source is read-only by default. Set **Forward Actions** true and bind **On Action Requested** only if your project wants entries to send selection/commands. **Request UI Action** rejects empty actions, missing consumers, released/stale entities and invalid worlds. An accepted return means the request was forwarded, not that gameplay completed.

Your callback owns action validation and network authority. Route mutations through the game's existing server/ownership rules. The adapter never grants resources, applies damage, spawns entities or implements inventory. In multiplayer, displaying local replicated entity data does not authorize a client mutation or certify MassForge replication behavior.

Equivalent standalone implementation

The optional `Examples/ForgeMassExamples` project supplies a small original Watchpost interface. **Lyra/Orin** select a scout; **Hold/Ward** change health, **Withdraw** destroys its entity, and **Reinforce** explicitly rebinds the retained entry to a new generation. `-IFMassStandalone` supplies ordinary mutable IFPresentationData to the same saved screen and entry. It compares providers inside a host that still requires MassForge; it is not proof of plugin absence.

The default Outpost and Emberwatch project independently runs without MassForge. Its readout/action Blueprint entries use the same **Initialize Entry / On Entry Refreshed / Release Entry / Request UI Action** contracts. A project can replace its source with a Blueprint IFDataSource and **Notify Data Changed**, without changing the entry's parent class. Watchpost authoring documents the actual saved graphs, responsive layout and command routing.

Dependencies and current limitations

Copy the optional integration into a project's Plugins folder only when MassForge and InterfaceForge are installed; enable it explicitly. Do not add it to the default standalone project. Private SDK validation copies under Evidence are excluded from distribution and contain no reference content assets. Only original bridge code is intended to ship in this integration.

Attribute view state includes Uninitialized, Ready, Waiting for Mass, Invalid Entity, Unavailable Attribute, Invalid World, Capacity Reached and Released. This bridge observes configured scalar attributes; abilities/effects/tags/inspection sections require explicit provider extensions, not a claim that every MassForge system is already mapped. Current-core runtime evidence verifies the actual saved example and lifecycle on UE5.8.3 Win64/MassForge0.2. Physical controls, non-Windows platforms and multiplayer behavior remain unverified.

GAME INTERFACES AND ADAPTERS

Incremental migration from the reference games

Game Master and Sakura Serenade are read-only references for this development goal. The following steps are guidance for a future project-owned migration branch. They do not authorize modifying either reference checkout. Preserve existing Git changes and gameplay behavior; introduce one UI flow at a time with an immediate rollback path.

Keep game design and gameplay in the game

Game Master's layouts, styling, frames, artwork, and animations remain exclusive to Game Master. A game-owned widget may keep that design while using InterfaceForge; it must not be moved into the distributable plugin or standalone examples. Audit every hard/soft asset dependency before any redistribution. Mixed AshenForge/SoftMage headers and purchased packs in the references are not ownership-transfer evidence.

The new Outpost/Emberwatch examples are original designs with separate small providers. Inventory storage, equipment authority, combat rules, quests, production and selection remain owned by the game. The UI consumes data and requests actions; replacing a widget must not silently replace those systems.

Game Master: move ownership, retain CommonUI

The inspected **UGMUI_UserInterfaceSubsystem** is GameInstance-scoped and holds **CreatedPrimaryLayout** after **RegisterCreatedPrimaryLayoutWidget**. **UGMUI_PrimaryLayout** uses CommonUI composition. InterfaceForge instead provides **IFLocalPlayerSubsystem** → **Create Root** → **Register Layer / Push Screen** for each owning local player.

1. In a separate migration branch, create a project-owned IFRootLayout Blueprint and register HUD/Menu/Modal stacks. Keep the existing game layout as a separate fallback.
2. From the local PlayerController, get **IFLocalPlayerSubsystem** and **Create Root**. Route only the chosen pilot flow through it. Do not keep two active CommonUI roots competing for the same player's navigation/modal focus.
3. Create a project-owned IFScreen Blueprint for one menu/HUD panel. Set its **Initial Focus**, call parent **Initialize Screen**, and retain game-specific art/layout in that project asset.
4. Add a provider implementing **IFDataSource**. **Get UI Model** returns stable observable objects; **Request UI Action** forwards requests to the current gameplay controller/subsystem after its existing validation. Publish changed fields with **Notify Data Changed** rather than recreating models or polling the whole entity population.
5. Replace repeated widget-local handling of **FGMUI_InputPresentationState** with **IFInputPresentationSubsystem** → **On Input Presentation Changed**. Keep engine CommonInput controller data and project glyph art. Preserve meaningful pointer/keyboard distinction, focus restoration and cursor policy; test the pilot while a modal is open and after returning from a screen.
6. Keep current game save/audio/graphics policies. Route staged UI scalar preferences through IFSettingsSubsystem only where intentionally migrated. EnhancedInputUserSettings still owns remaps; CommonUI Accept/Back rows are separate.

The standalone examples opt into CommonGameViewportClient, IFNavigationInputData and Trigger Click. Evaluate the reference game's existing equivalents before changing project settings. A source-level compatibility cast is not a reason to duplicate the frontend controller architecture inside the plugin.

Sakura: adapt controller outputs, preserve the tint

The inspected **SSWidgetControllerBase** binds to ASSPlayerController, ASSPlayerState, USSAbilitySystemComponent and USSAttributeSet, with initial broadcasts followed by updates. Keep those gameplay bindings inside Sakura's provider. Translate their output to stable IFPresentationData or your own UObject models, then drive IFEntry/IFScreen widgets through public lifecycle events.

For an action entry: **Initialize Entry → assign the game Action Source**; **On Entry Refreshed** updates label, availability, remaining fraction and countdown. **On Entry Released** clears the previous source/subscriptions. **Request UI Action** remains a request to Sakura's existing ability system. The UI must not grant an ability or decide server authority.

The requested cooldown is a center-origin angular tint over the icon, not a perimeter tube. The independently authored **InterfaceForge Center Tint** consumes normalized remaining fraction and configurable opacity/tint/direction. See center-tint controls, original behavior/provenance, and the saved RPG action recipe. Preserve the game's existing timing and hide-on-ready behavior when adapting it. Keep third-party icon/frame/art dependencies in Sakura; use original or project-owned art in any distributable example.

Sakura has both SSUI and main-module UI/settings code, including SSUISubsystem. Choose one migrated owner for each settings/remap transaction and retire duplicate bindings only after the pilot passes. Do not reorganize unrelated systems to make the new UI fit.

Optional MassForge path

MassForge0.2 uses FMFEntityHandle with **Index + SerialNumber** and explicit world context. Use **Get Mass Forge Entity from Actor** for represented actors; actorless entities use their originating world-scoped handle. A bare index is unsafe. GetAttribute can report **Mass Is Processing**, missing schema/fragment or stale entity; preserve a pending reading rather than treating busy as destroyed.

The separate InterfaceForgeMass integration is being implemented against those actual APIs. It does not become a dependency of InterfaceForge or the default examples. Its presentation source must observe only selected/visible handles, coalesce native attribute events, release stale entities and forward game actions to project-owned handlers. See RELEASE_ACCEPTANCE.md for its current unverified gates; installation/runtime claims await actual adapter evidence.

Prove the pilot before expanding

Exercise main/settings/HUD/pause/return, Apply/Cancel/defaults/failure, current save compatibility, genuine gameplay responses, screen cleanup, list recycling, and controller/pointer/keyboard transitions. Compare behavior before/after in the game branch. Physical controller testing is deferred in this environment and must not be inferred from software events. Keep an explicit switch back to the original flow until the game-owned pilot is accepted.

VERSION AND VERIFICATION

Visual catalog - every current interface and widget

Historical baseline catalog. Current UE5.8.3 Emberfall/Ironfront captures are in the product gallery. This older gallery remains for review and does not certify the current rebuild.

127 real Unreal captures: running menus and HUDs, complete widgets, Blueprint Designer views and graph wiring. Open any image at full size to inspect it.

Current implementation · 18 October 2026 · UE5.6.1, with the optional widget previews in UE5.8.3.



RPG interface running in Unreal

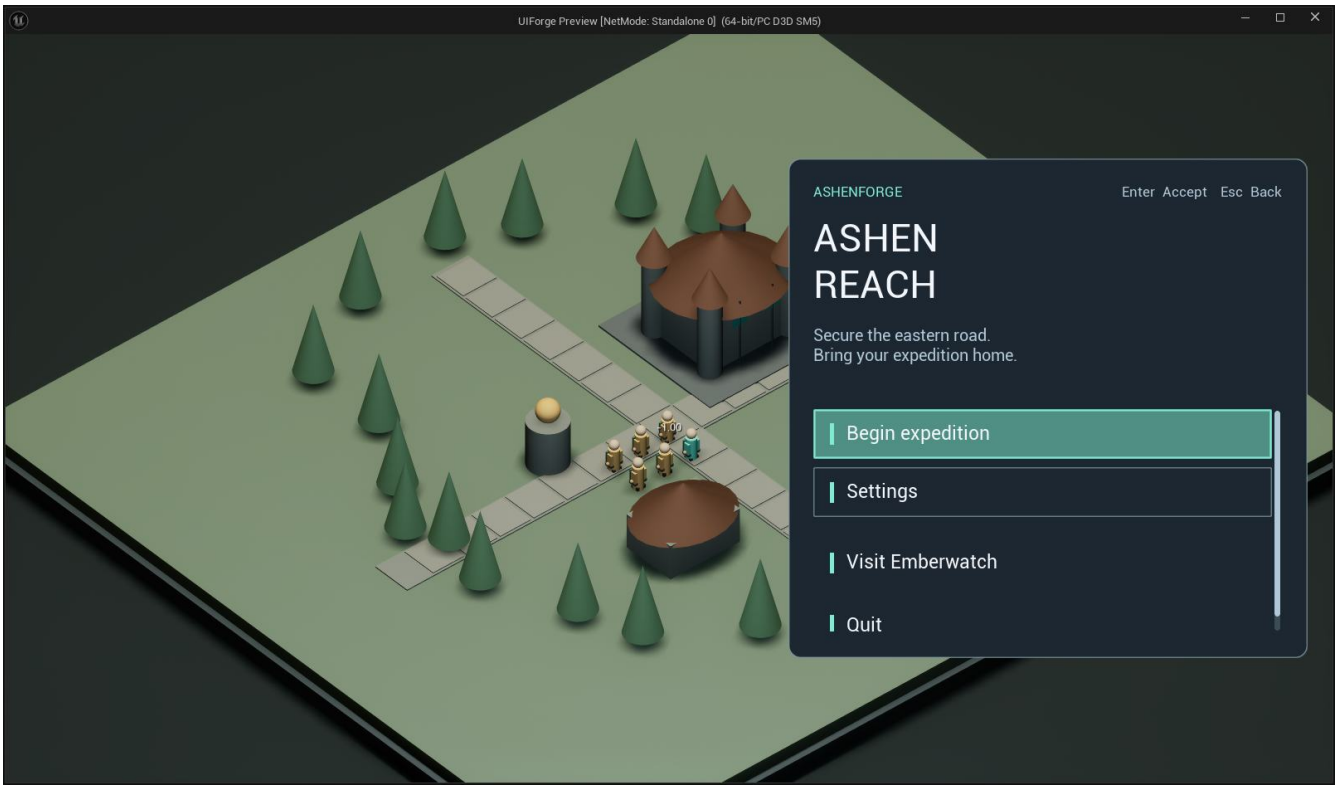
Open RPG interface at full size

Read the image labels: live PIE windows show running examples; isolated UMG previews render the saved/native widget with explicit sample data; Designer and graph images show Unreal's actual asset editor. Roots and routing containers are shown through a composed interface because they own no independent visual body.

Code and Blueprint scheme · Blueprint authoring guide

Running RTS example

MainMenu



Actual main menu with expedition and world-travel controls.

Open full-size image - Actual main menu with expedition and world-travel controls.

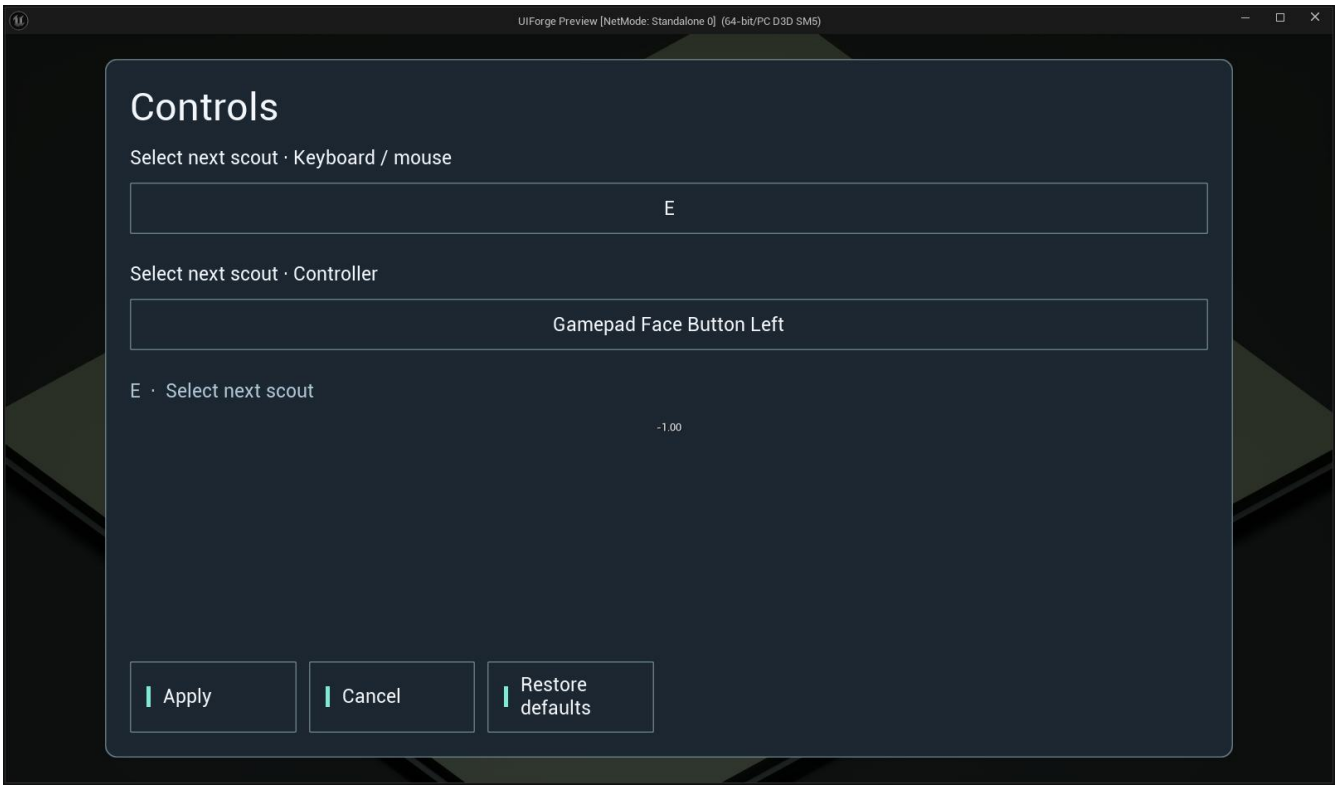
Settings



Actual UI/audio/text settings, locale and motion options.

Open full-size image - Actual UI/audio/text settings, locale and motion options.

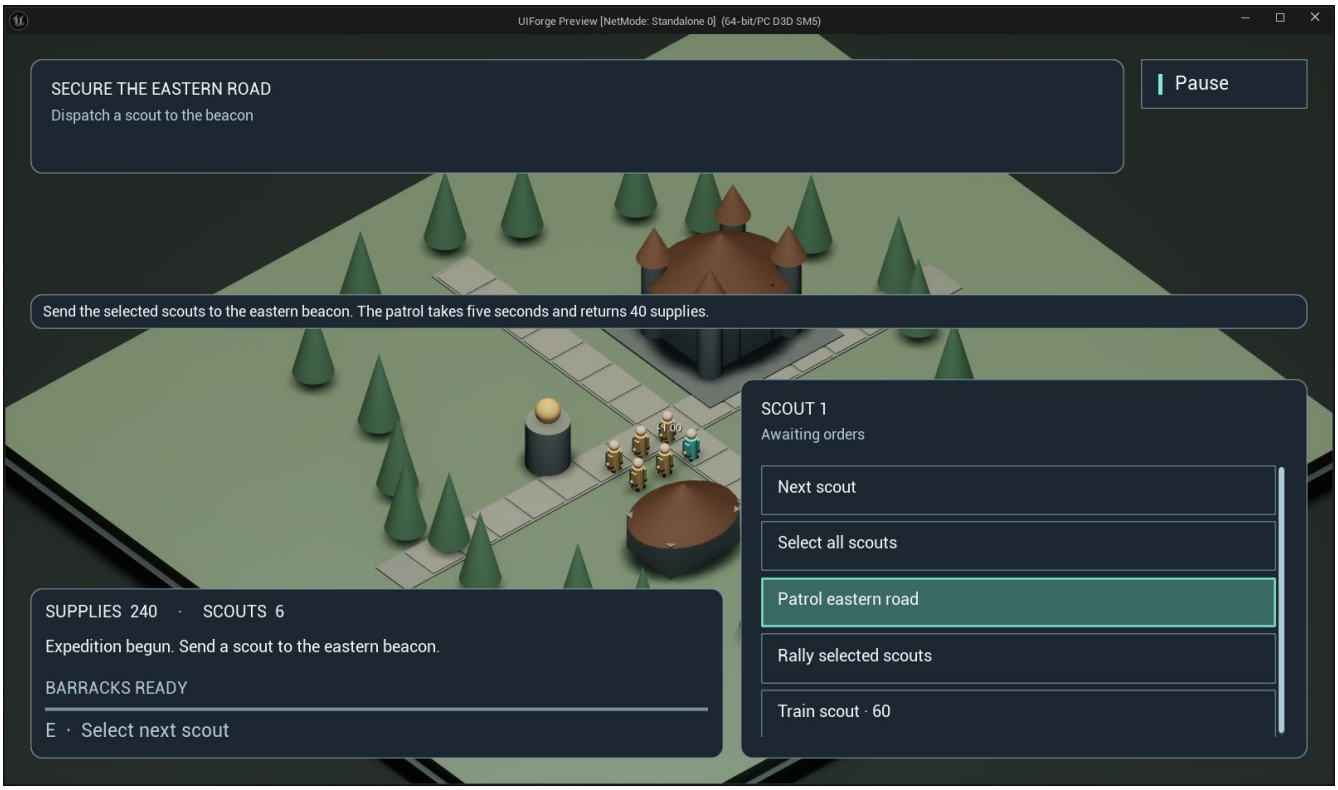
Controls



Keyboard/controller remapping, Apply/Cancel/Defaults.

Open full-size image - Keyboard/controller remapping, Apply/Cancel/Defaults.

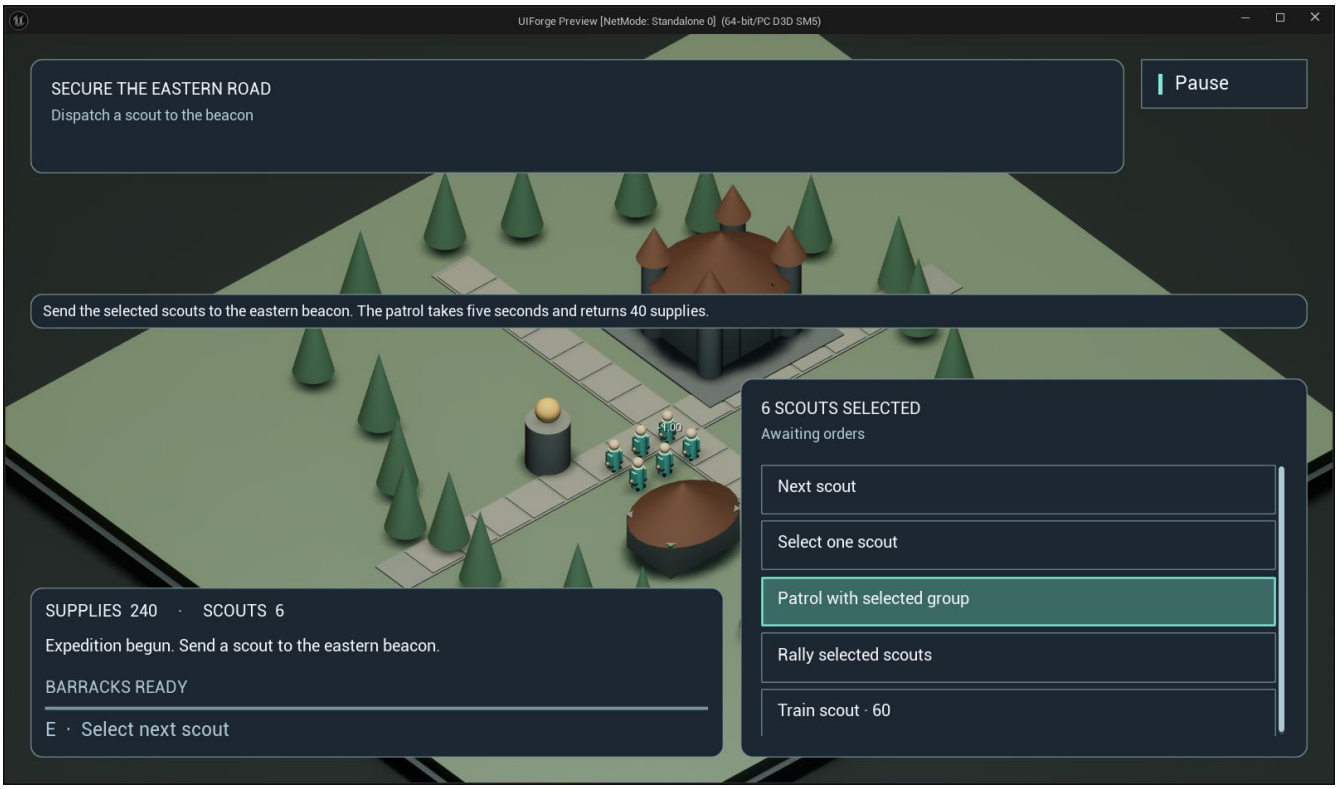
HUD



Actual selection, command, resource and production HUD.

Open full-size image - Actual selection, command, resource and production HUD.

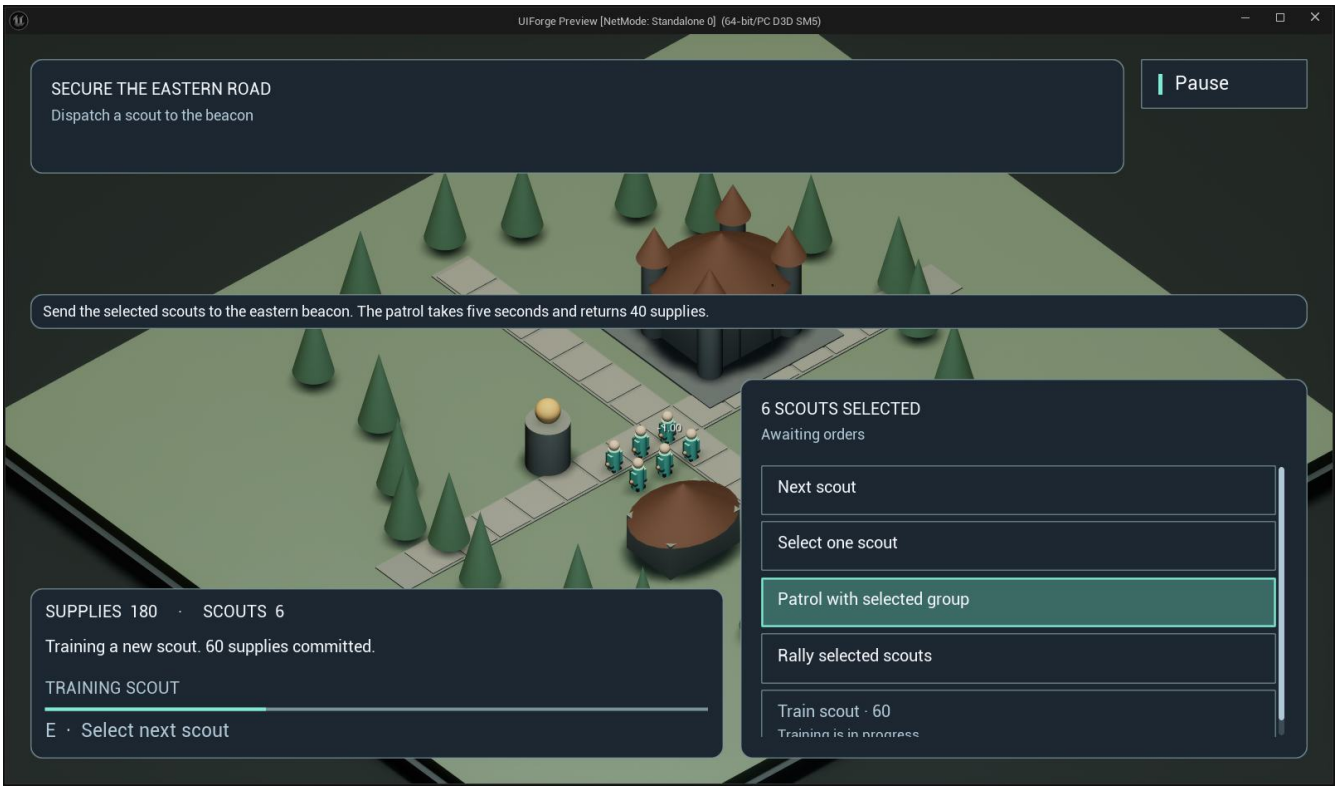
MultiSelection



Multi-selection summary and contextual commands.

Open full-size image - Multi-selection summary and contextual commands.

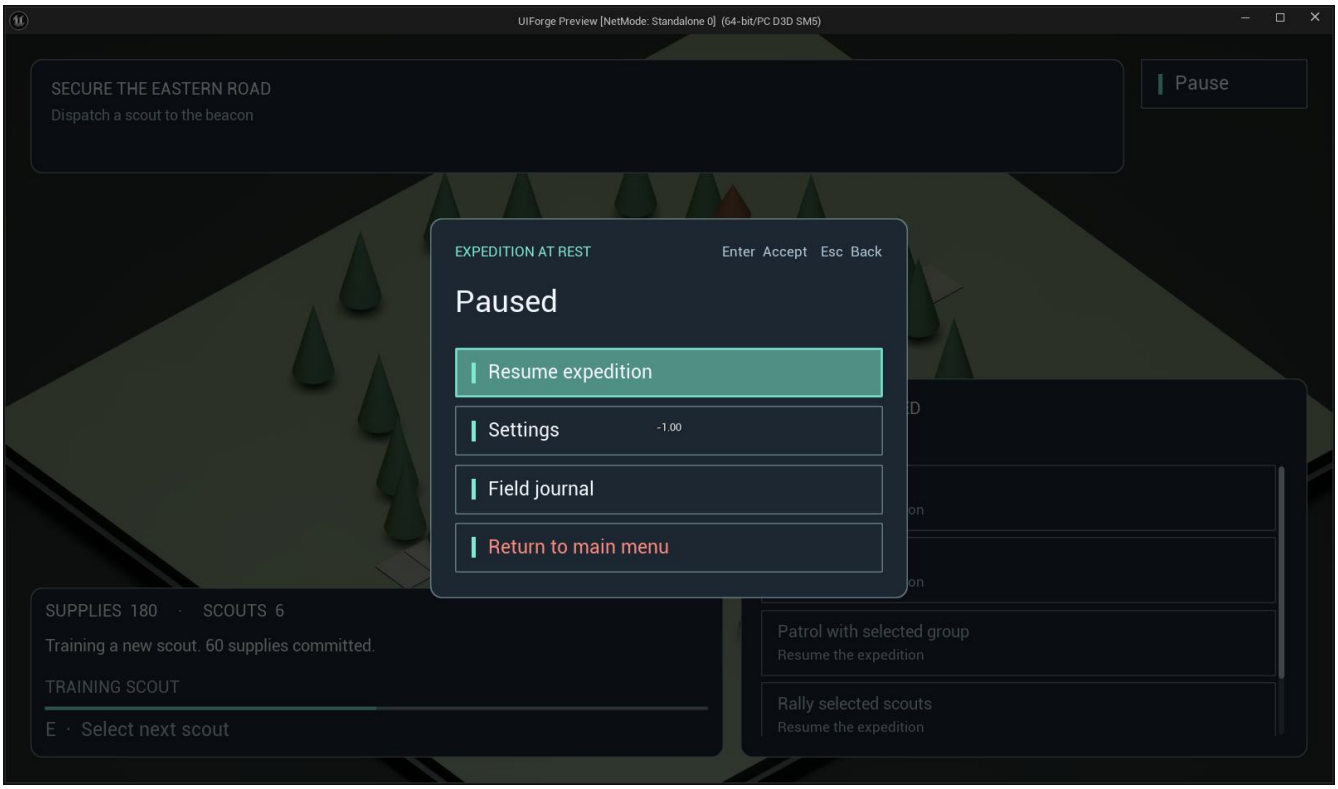
Production



Training queue and resource feedback after a real order.

Open full-size image - Training queue and resource feedback after a real order.

Pause



Pause menu over the game scene.

Open full-size image - Pause menu over the game scene.

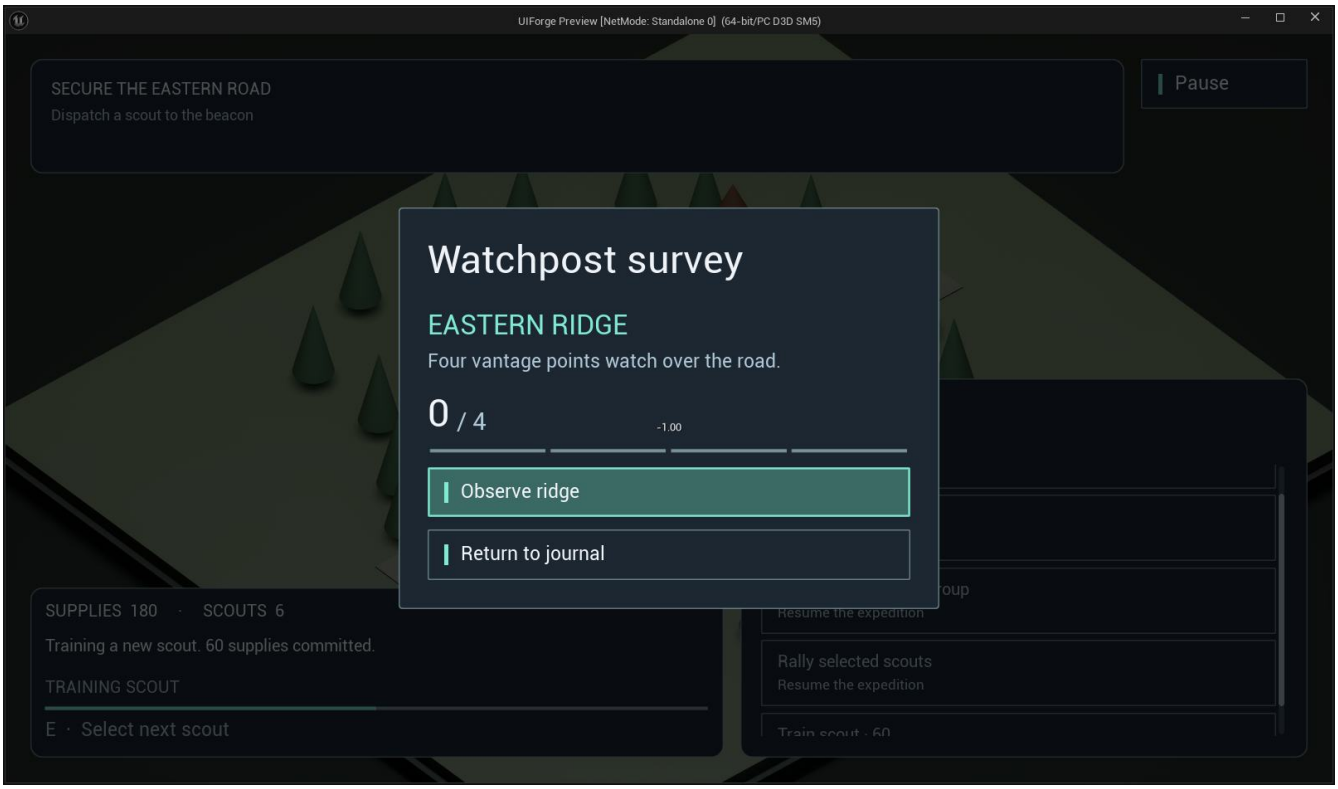
Journal



Journal collection, theme choices, survey and card navigation.

Open full-size image - Journal collection, theme choices, survey and card navigation.

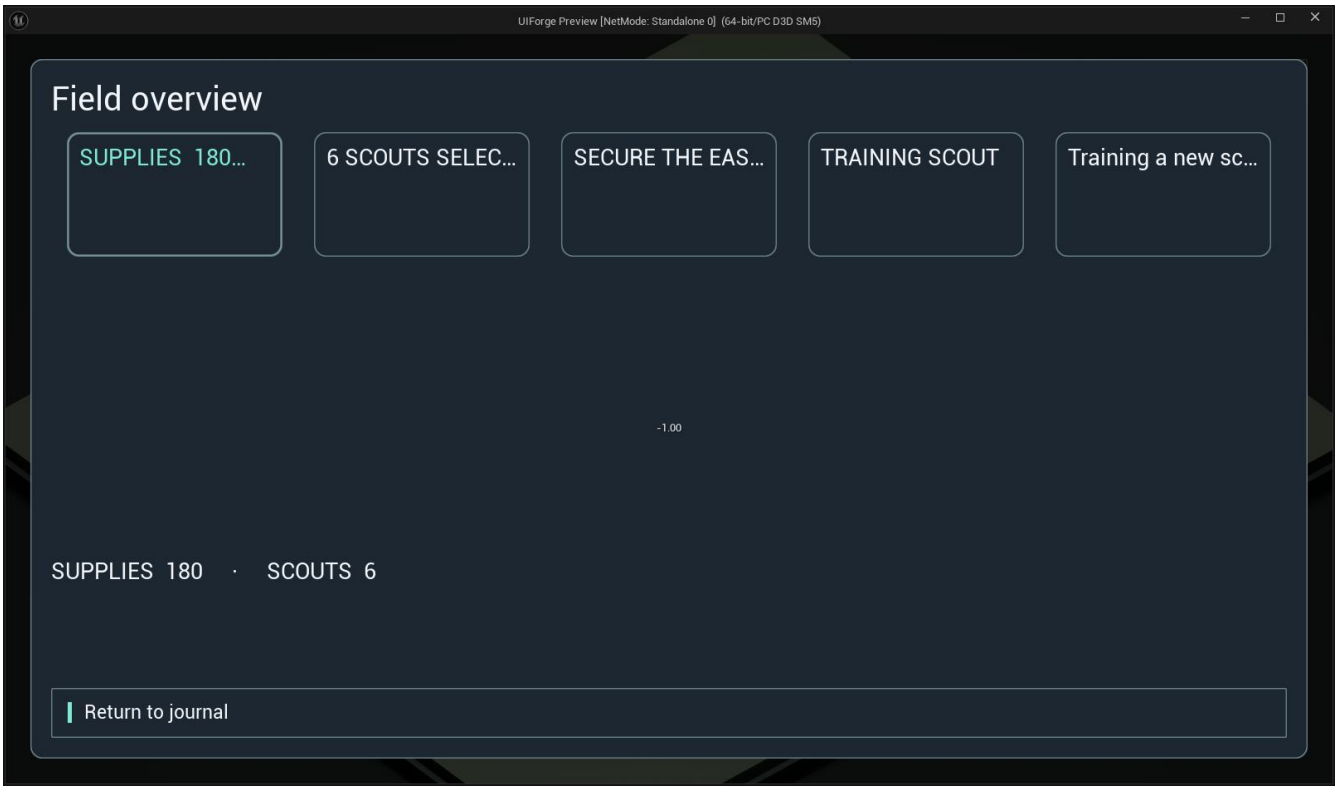
Survey



Developer-created Blueprint screen and entry using a Blueprint data source.

Open full-size image - Developer-created Blueprint screen and entry using a Blueprint data source.

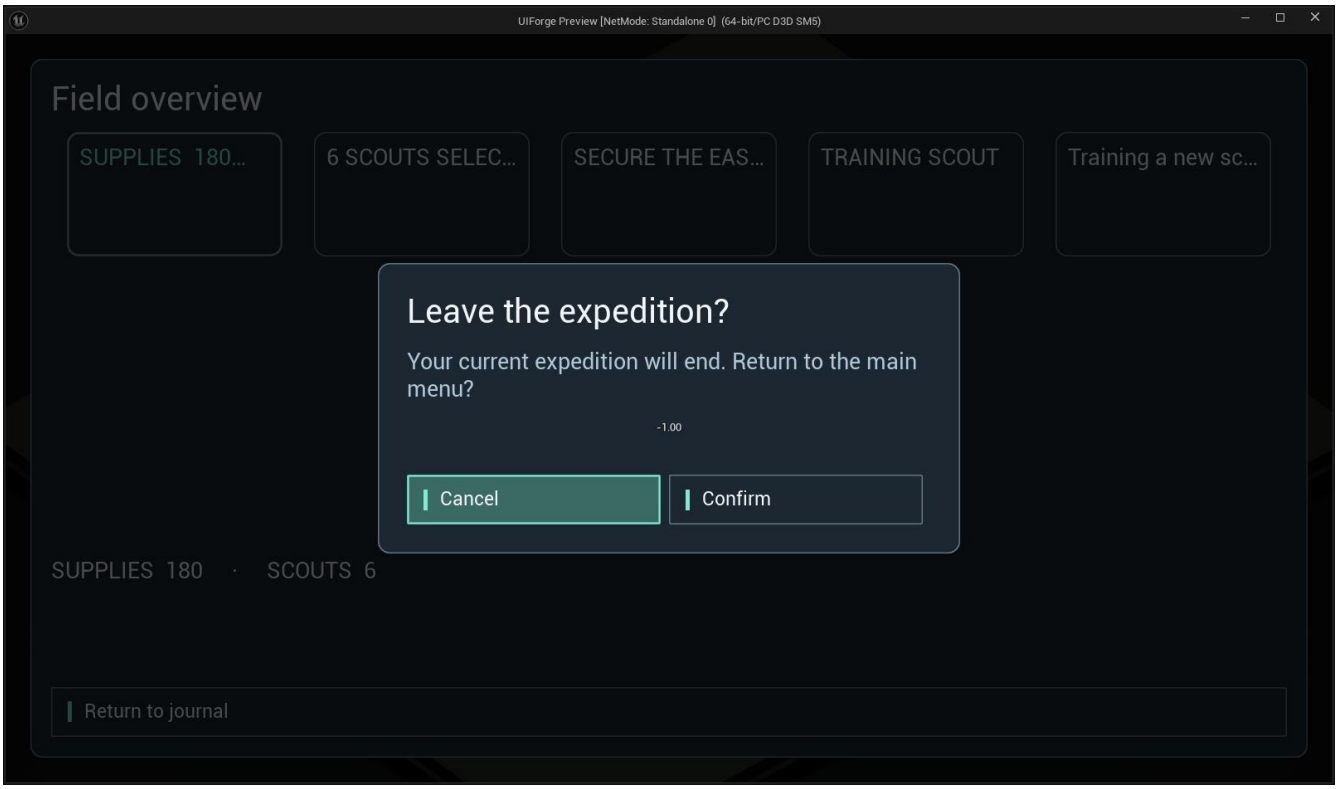
Cards



Virtualized field-card grid with selection detail.

Open full-size image - Virtualized field-card grid with selection detail.

Confirmation



Modal confirmation layered above a custom Blueprint screen.

Open full-size image - Modal confirmation layered above a custom Blueprint screen.

Journal Ember

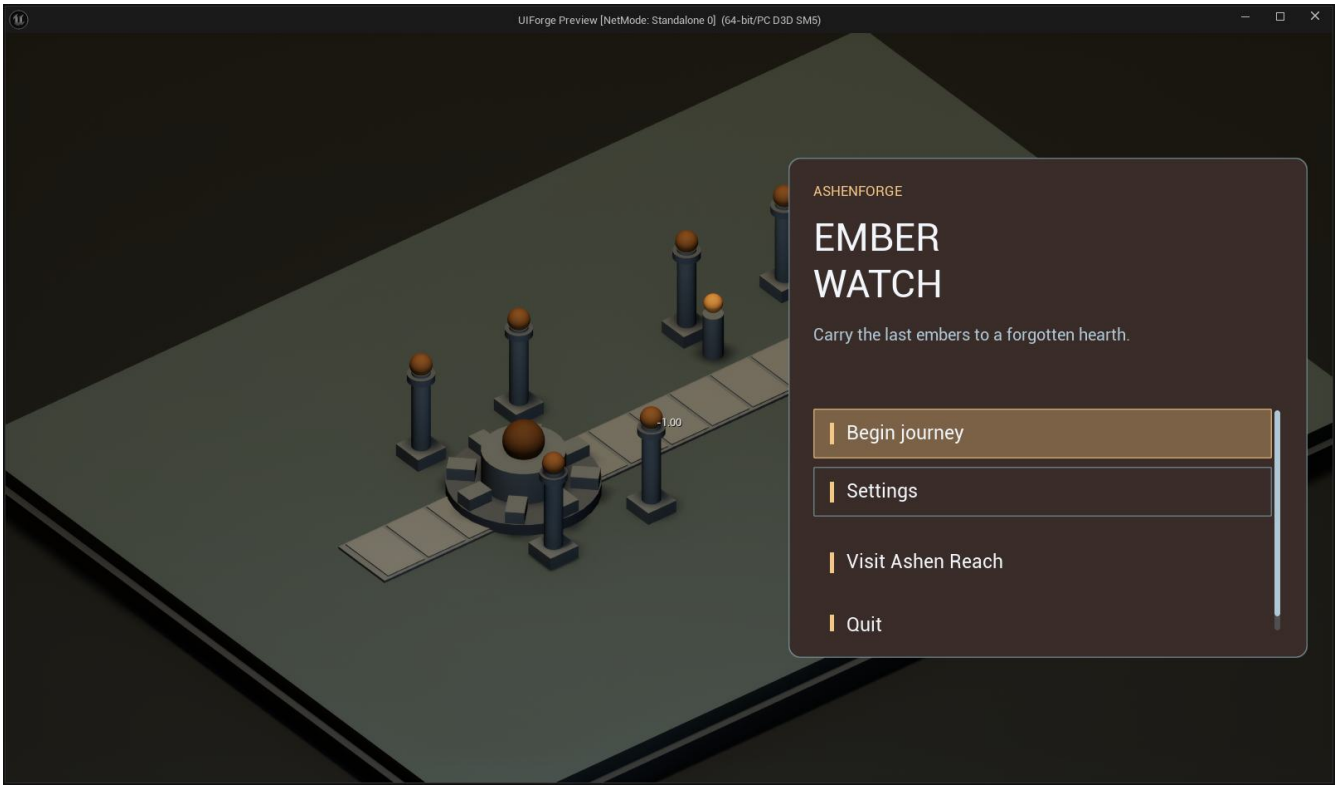


Same journal with the alternate theme; no duplicated screen tree.

Open full-size image - Same journal with the alternate theme; no duplicated screen tree.

Running RPG example

MainMenu



Actual RPG menu over Emberwatch; shares menu widgets with a distinct theme and data source.

Open full-size image - Actual RPG menu over Emberwatch; shares menu widgets with a distinct theme and data source.

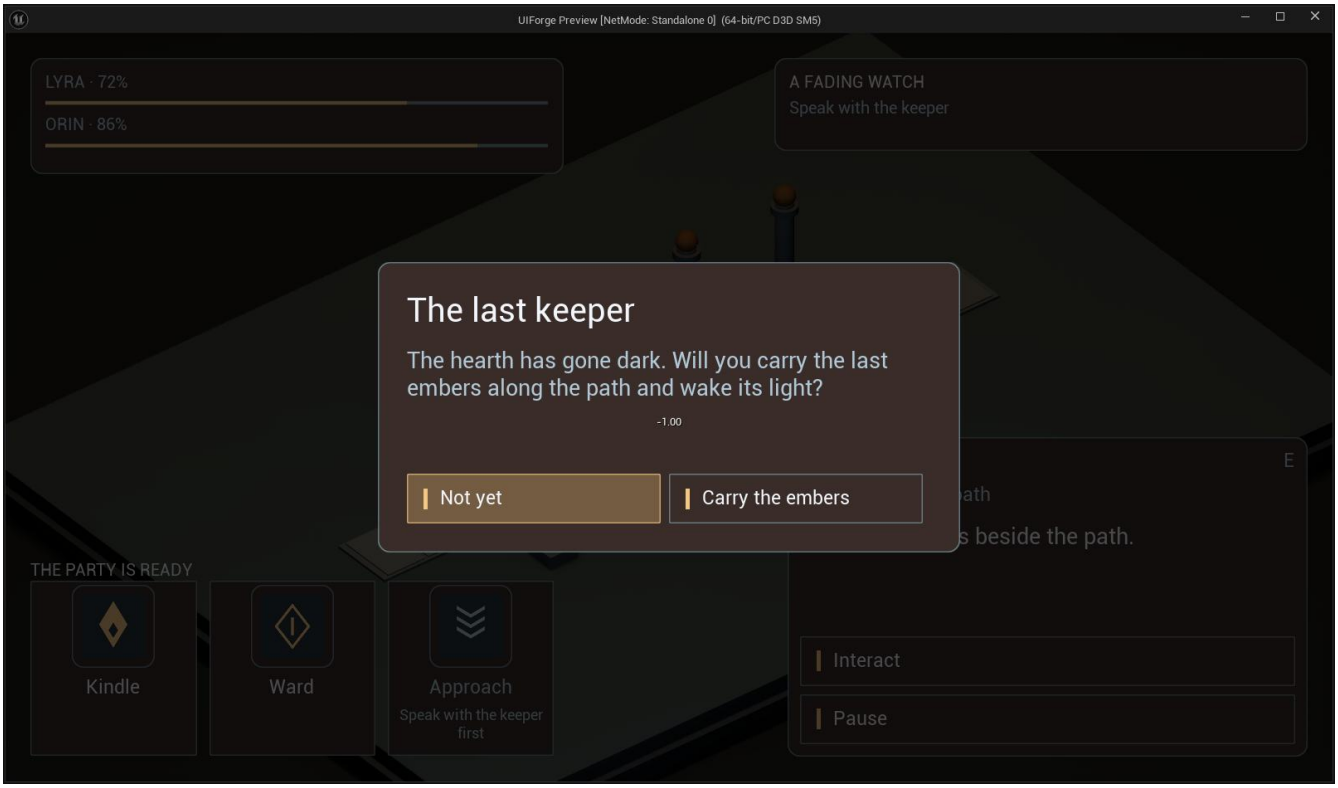
HUD



Party status, quest, contextual interaction and three action entries.

Open full-size image - Party status, quest, contextual interaction and three action entries.

KeeperDialog



Actual keeper interaction confirmation, before accepting the quest.

Open full-size image - Actual keeper interaction confirmation, before accepting the quest.

QuestAccepted



Quest accepted; Approach becomes available.

Open full-size image - Quest accepted; Approach becomes available.

Hearth



Party at the hearth, with contextual status and actions.

Open full-size image - Party at the hearth, with contextual status and actions.

Kindle Cooldown



Actual icon and center-origin tint during the Kindle cooldown.

Open full-size image - Actual icon and center-origin tint during the Kindle cooldown.

Ward Cooldown



Party restoration, notification and running Ward cooldown.

Open full-size image - Party restoration, notification and running Ward cooldown.

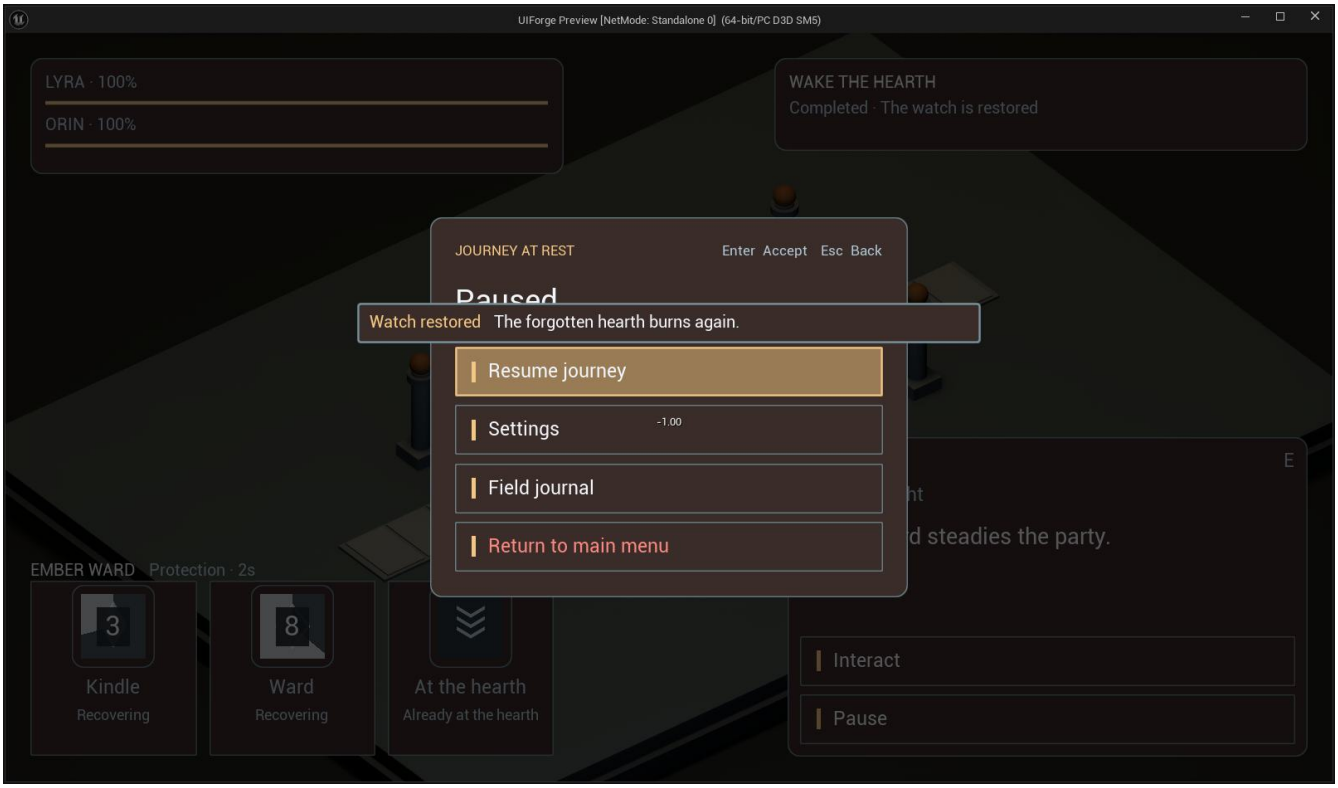
Ward MidCooldown



Same cooldown at a later fraction; no replacement progress effect.

Open full-size image - Same cooldown at a later fraction; no replacement progress effect.

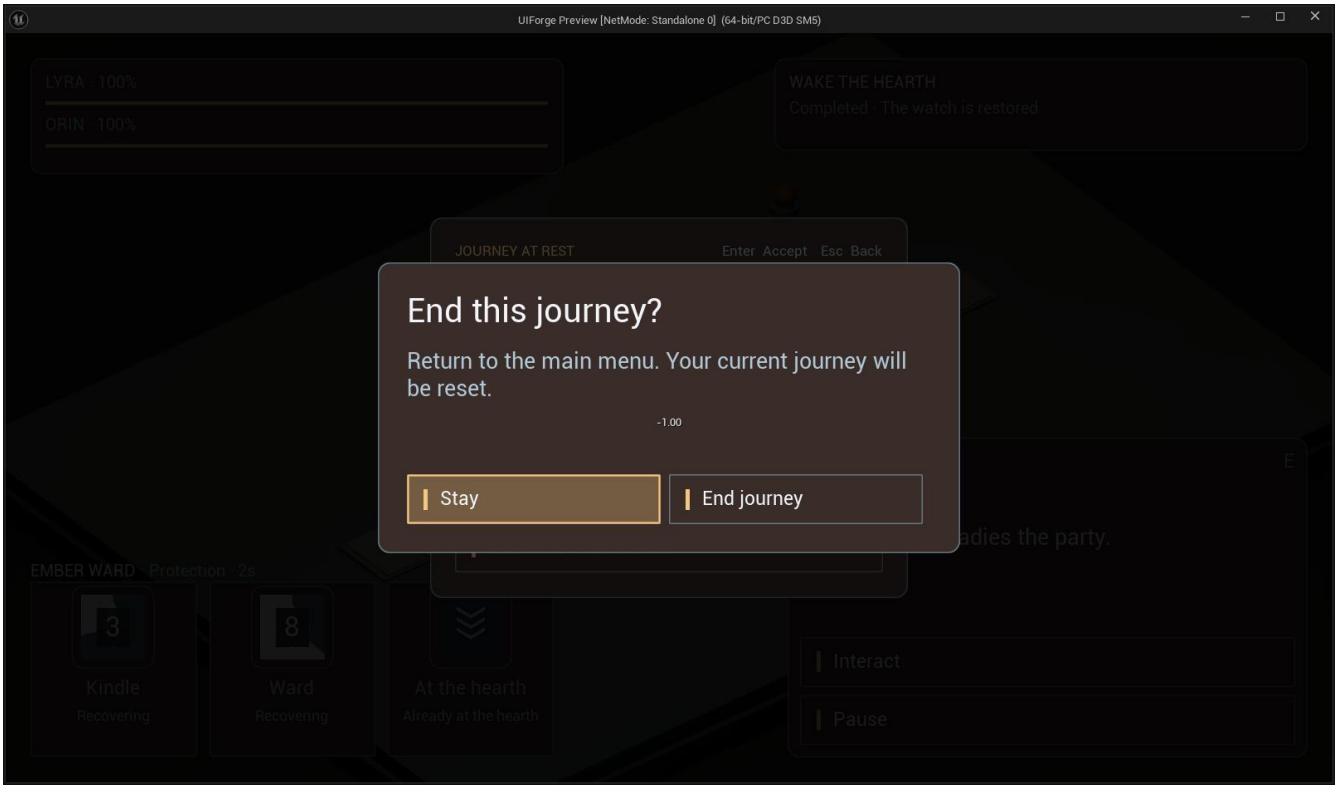
Pause



Shared pause flow retaining RPG context.

Open full-size image - Shared pause flow retaining RPG context.

ReturnDialog

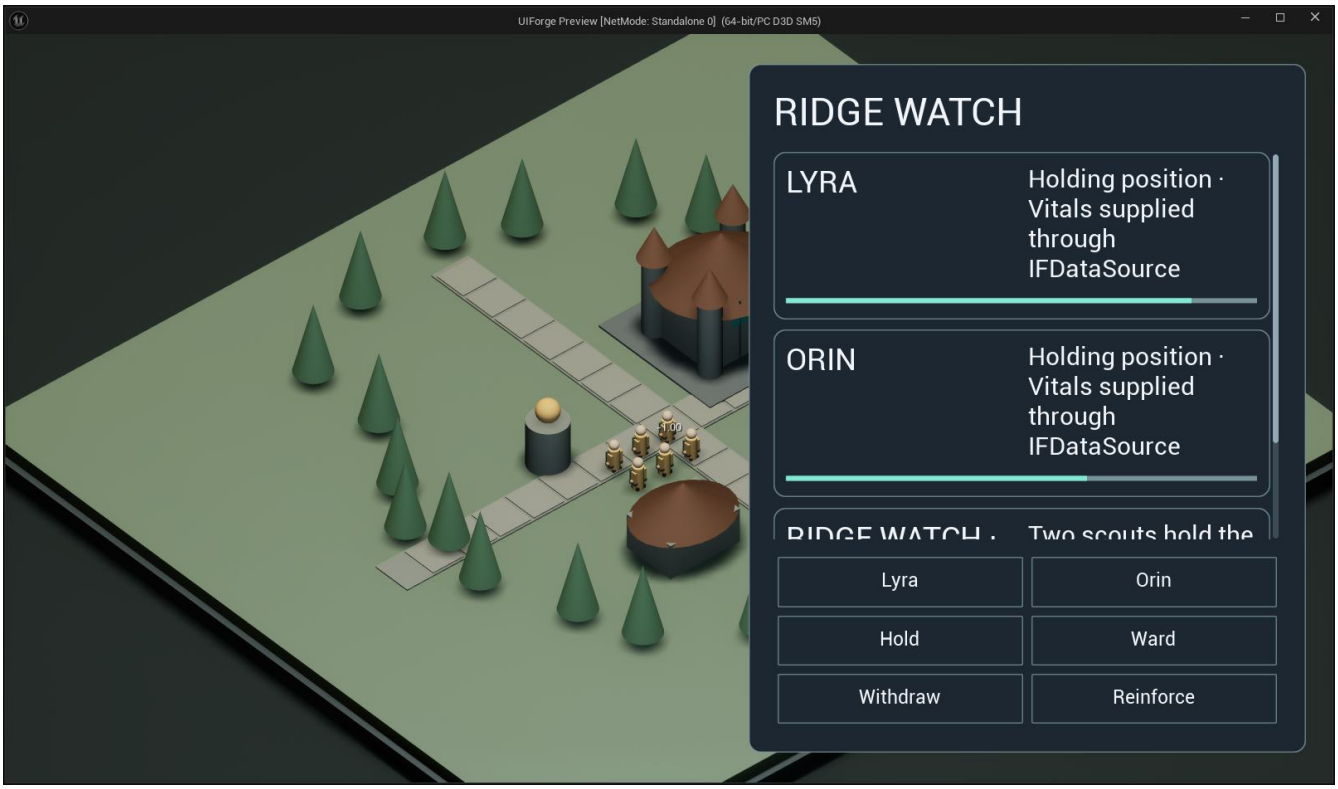


Actual return-to-menu confirmation.

Open full-size image - Actual return-to-menu confirmation.

Optional example interface

These are the original optional Watchpost widgets. A documentation-only IFDataSource supplies the visible values; its commands do not implement entity/gameplay actions. The background is the owned UIForge outpost scene, not a claim that the original MassForge example world was executed.



Optional Watchpost interface in UE5.8 with explicit preview data

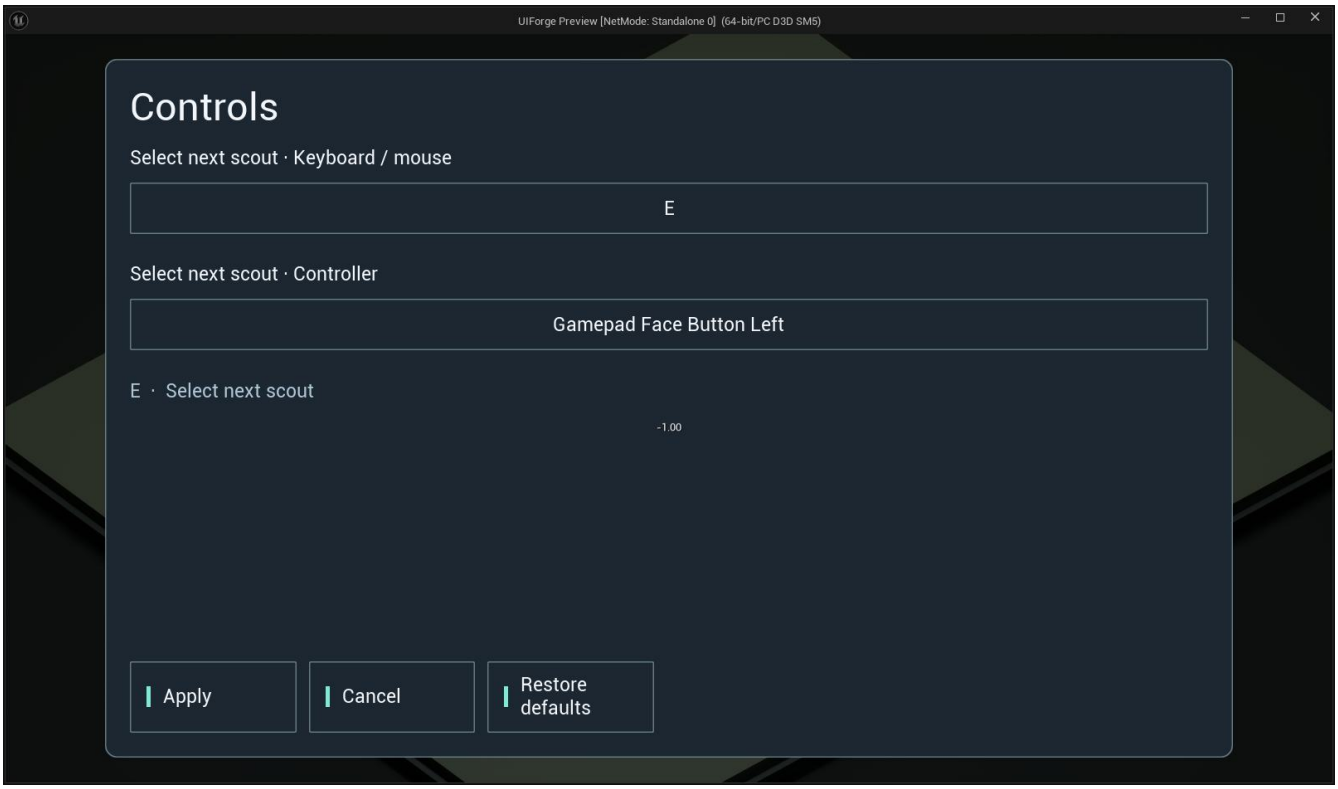
Open full-size image

Every Widget Blueprint

Each entry has a rendered interface/complete-widget image and its actual Designer editor. Isolated entry data is deliberately visible in the captions. The folder/code document lists the complete Designer hierarchy and every saved graph node.

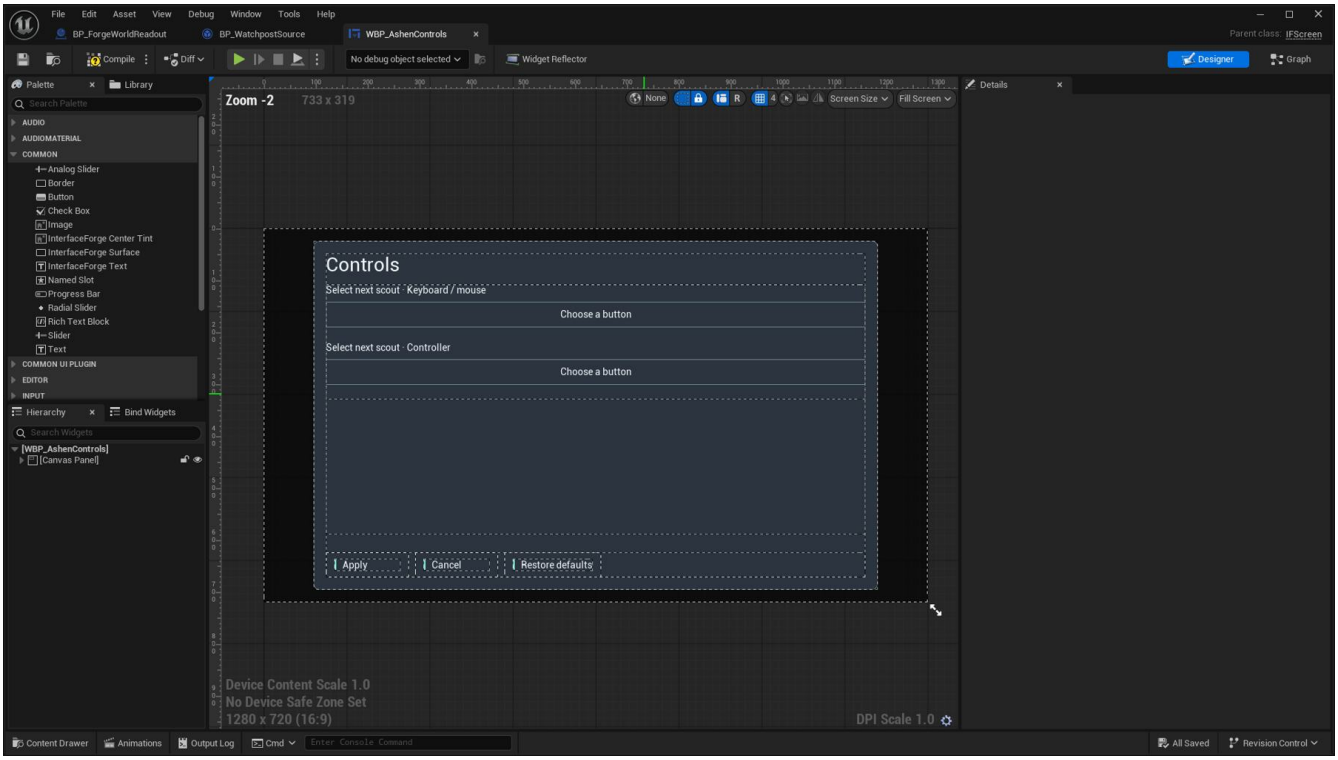
WBP_AshenControls

`/Game/UI/Blueprint/WBP_AshenControls.WBP_AshenControls` → `IFScreen` · screen.



WBP_AshenControls - screen

Open full-size image - Live composed interface.

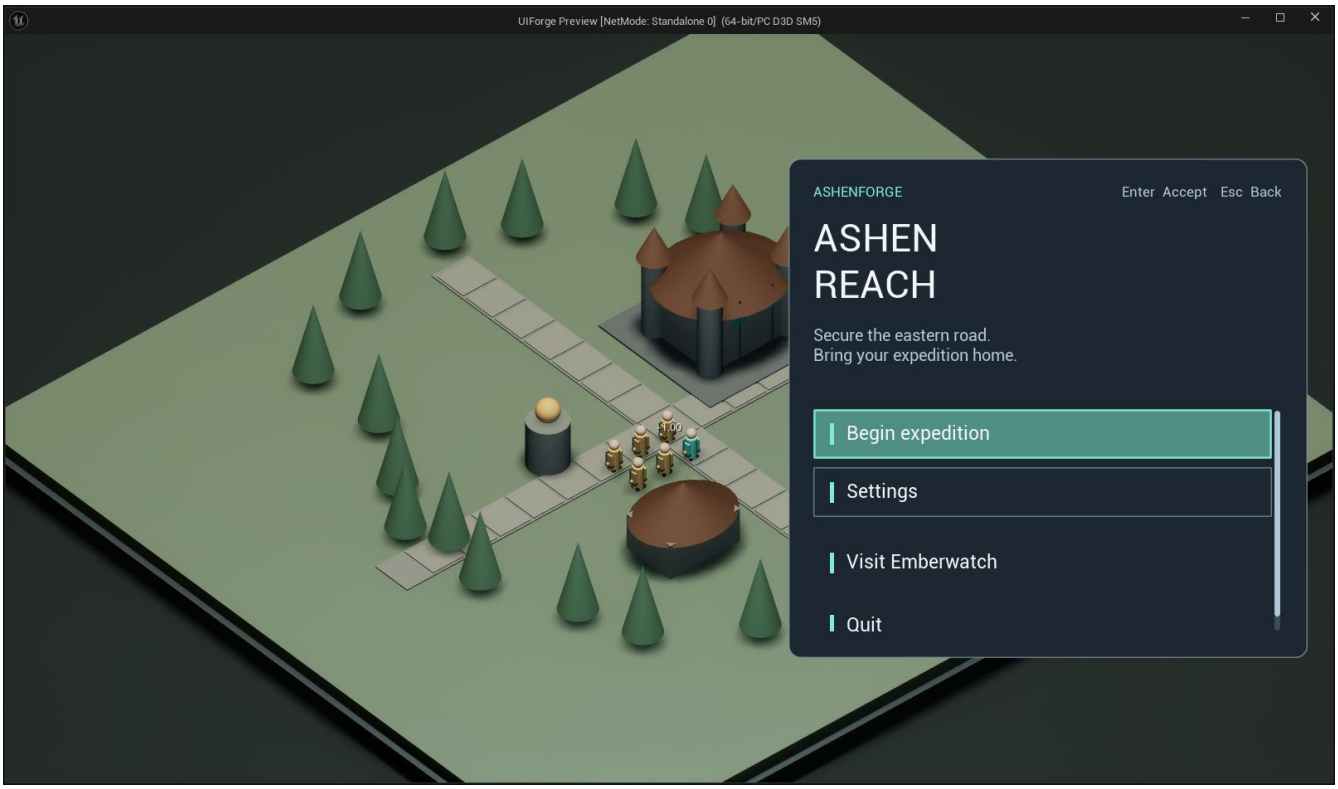


WBP_AshenControls in Unreal Designer

Open full-size image - Actual asset editor; design-time values can differ from runtime bindings.

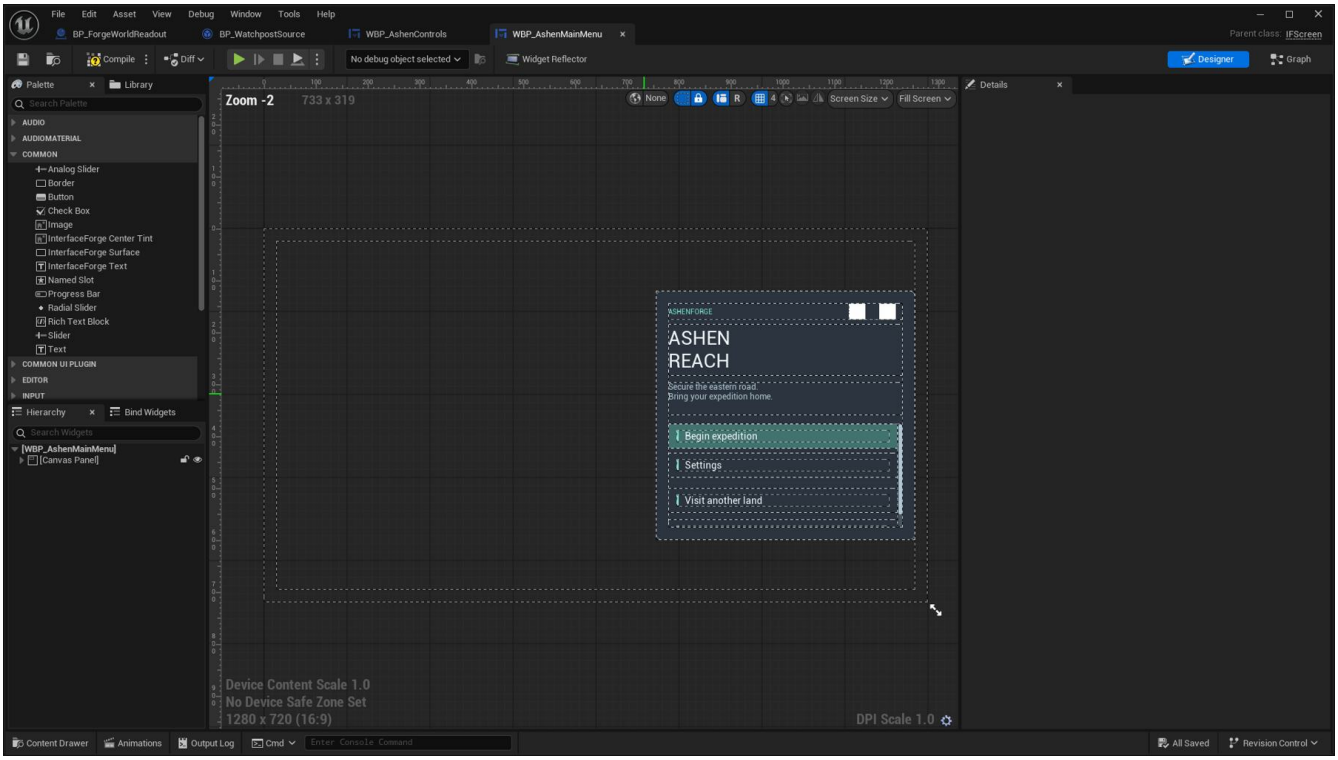
WBP_AshenMainMenu

`/Game/UI/Blueprint/WBP_AshenMainMenu.WBP_AshenMainMenu` → `IFScreen` · screen.



WBP_AshenMainMenu - screen

Open full-size image - Live composed interface.

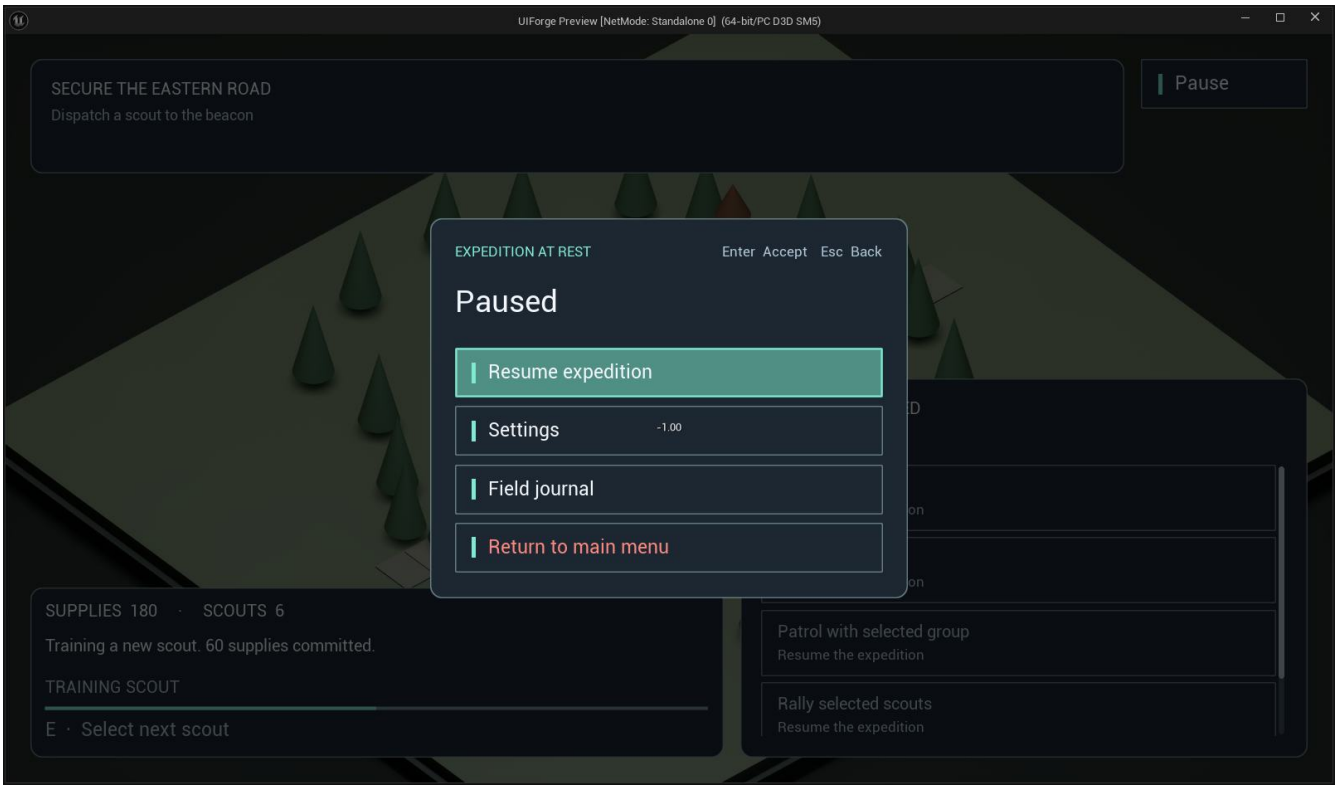


WBP_AshenMainMenu in Unreal Designer

Open full-size image - Actual asset editor; design-time values can differ from runtime bindings.

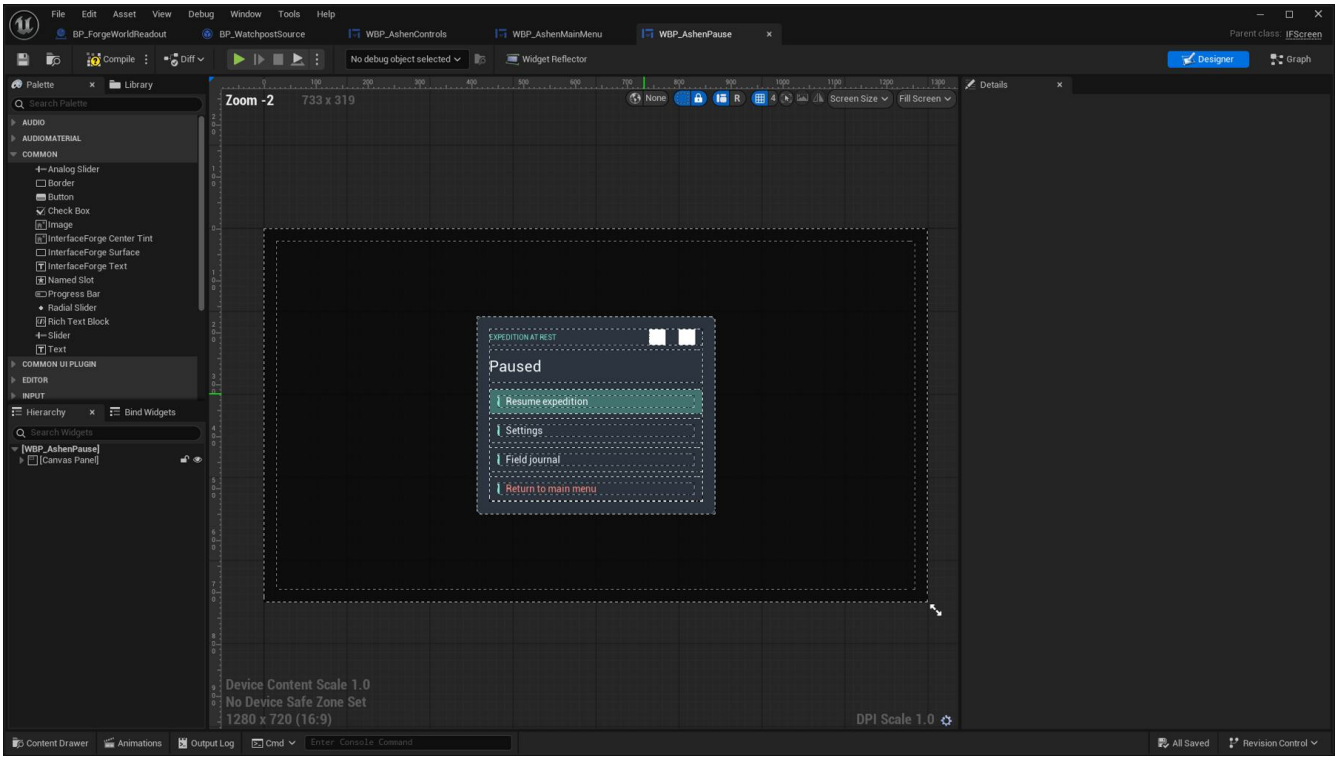
WBP_AshenPause

`/Game/UI/Blueprint/WBP_AshenPause.WBP_AshenPause` → `IFScreen` · screen.



WBP_AshenPause - screen

Open full-size image - Live composed interface.

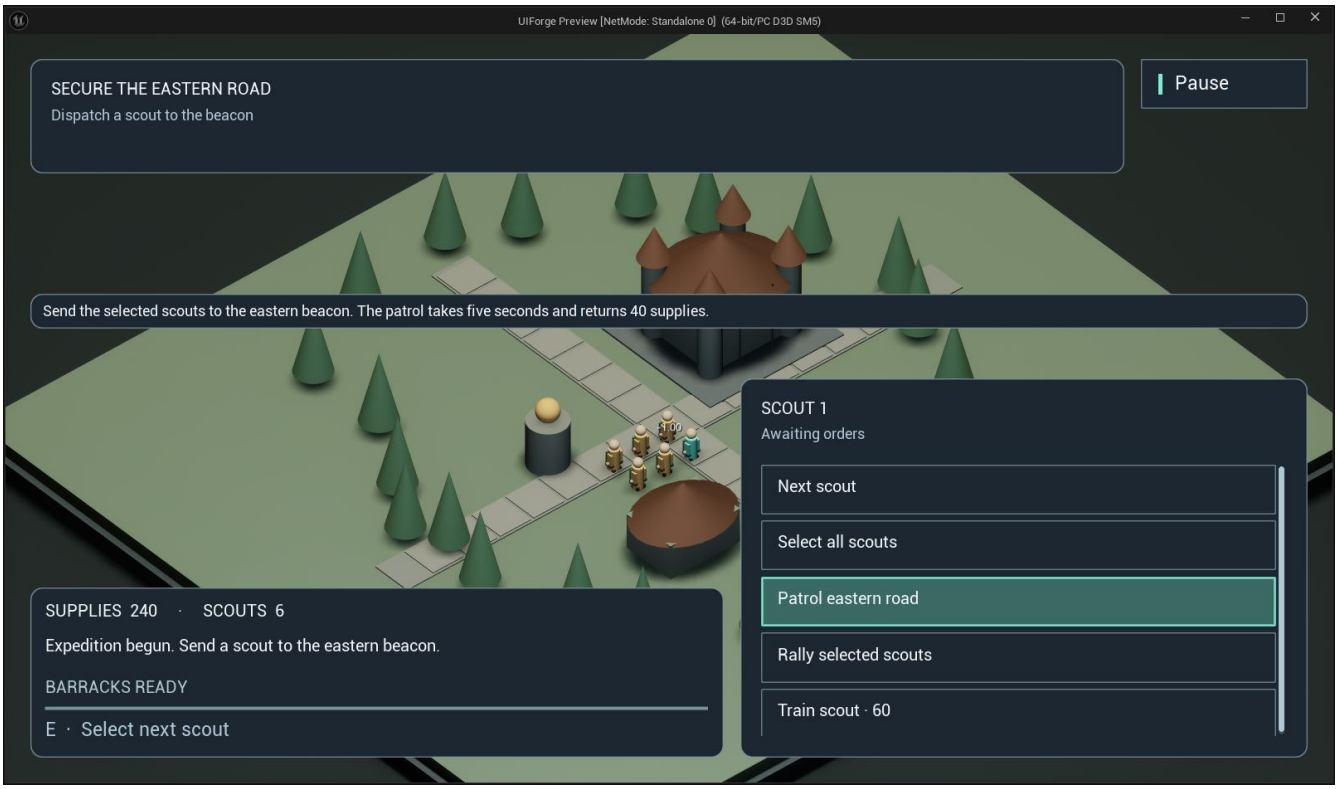


WBP_AshenPause in Unreal Designer

Open full-size image - Actual asset editor; design-time values can differ from runtime bindings.

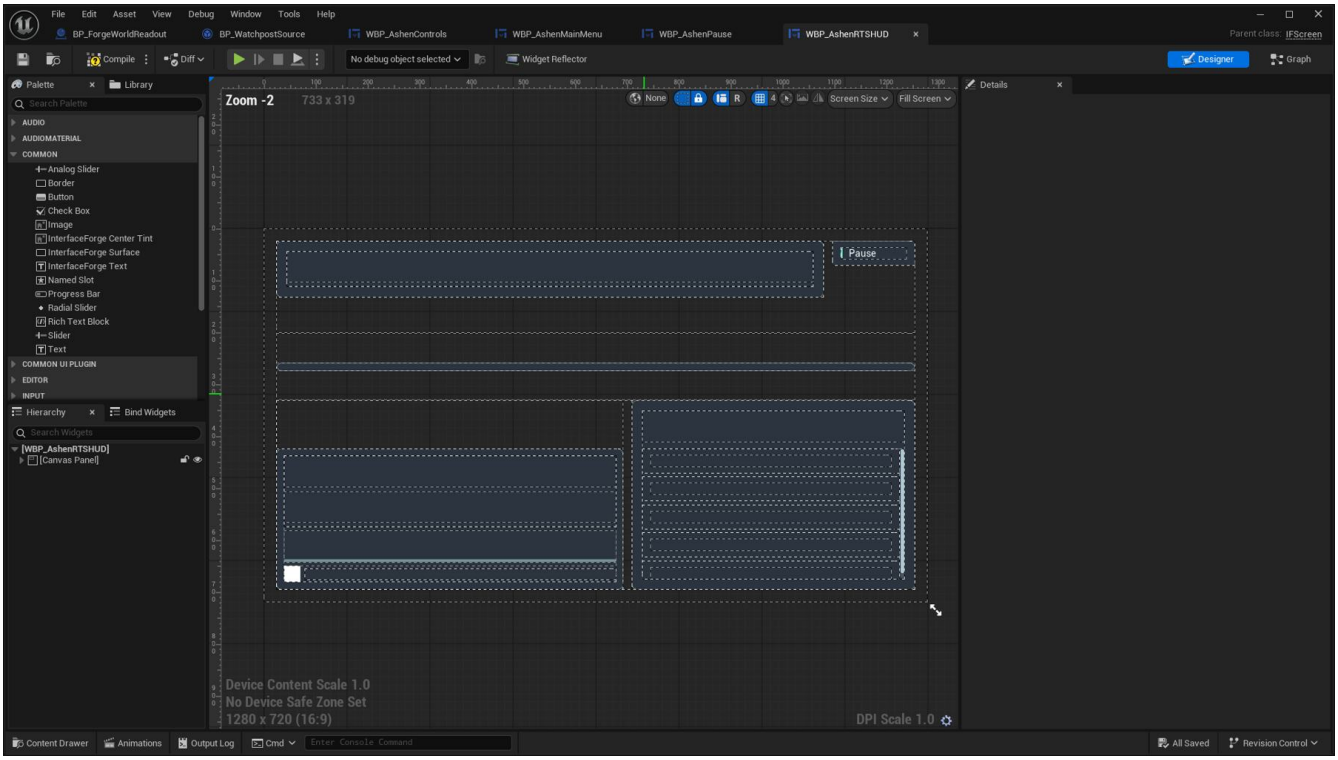
WBP_AshenRTSHUD

`/Game/UI/Blueprint/WBP_AshenRTSHUD.WBP_AshenRTSHUD` → `IFScreen` · screen.



WBP_AshenRTSHUD - screen

Open full-size image - Live composed interface.

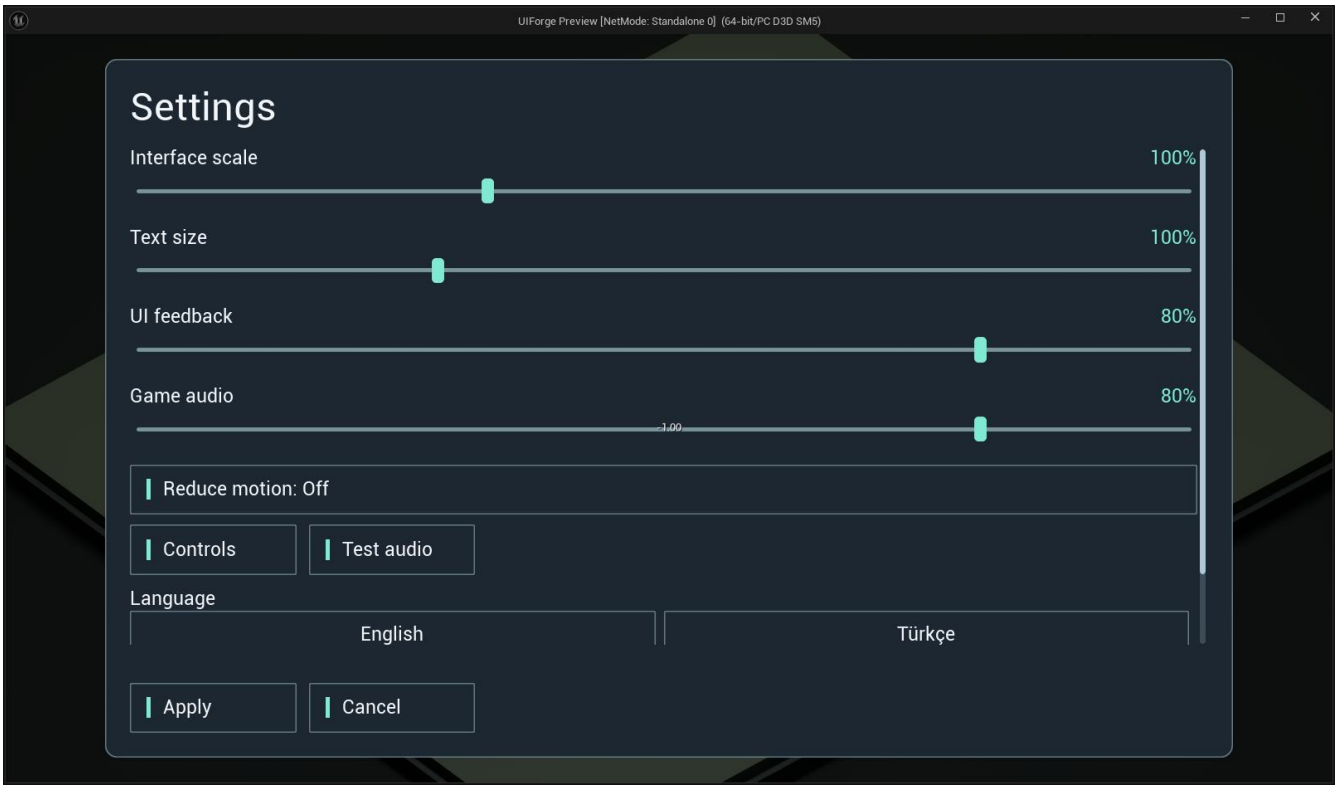


WBP_AshenRTSHUD in Unreal Designer

Open full-size image - Actual asset editor; design-time values can differ from runtime bindings.

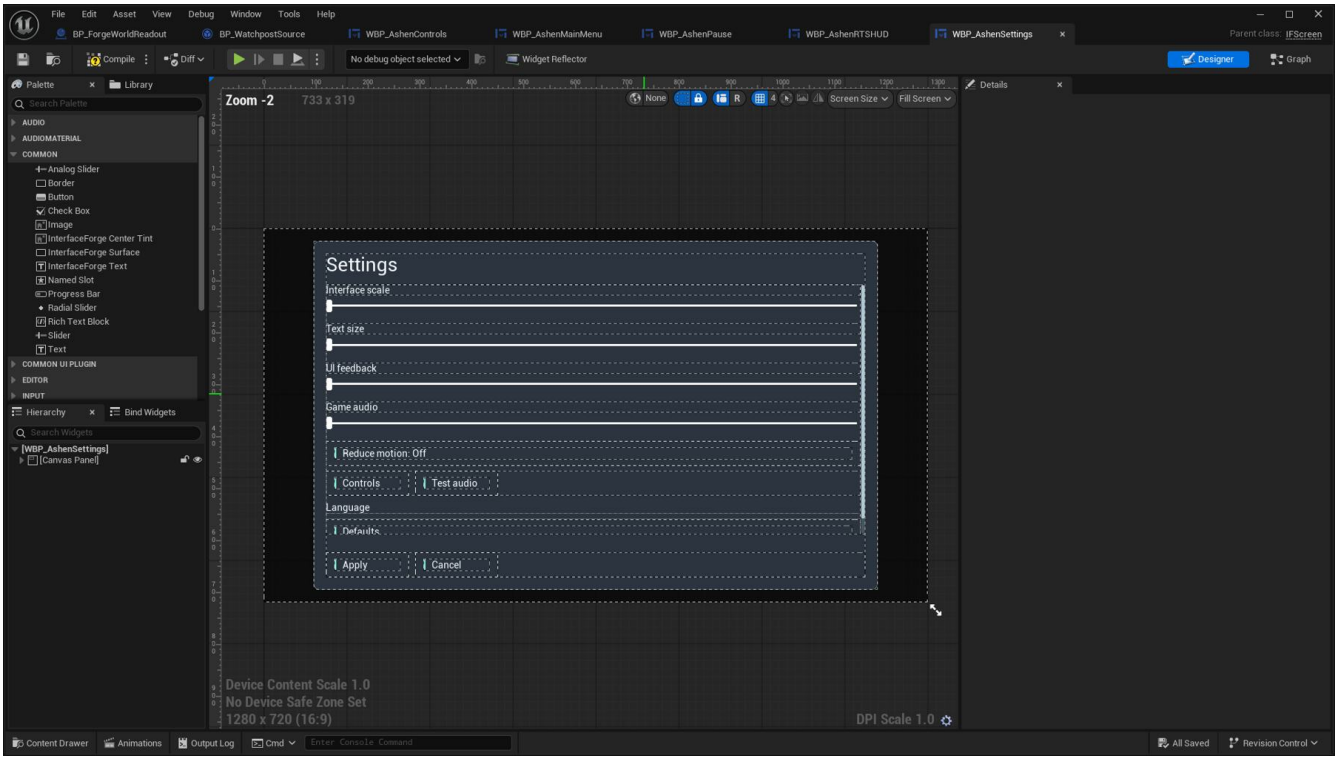
WBP_AshenSettings

`/Game/UI/Blueprint/WBP_AshenSettings.WBP_AshenSettings` → `IFScreen` · screen.



WBP_AshenSettings - screen

Open full-size image - Live composed interface.



WBP_AshenSettings in Unreal Designer

Open full-size image - Actual asset editor; design-time values can differ from runtime bindings.

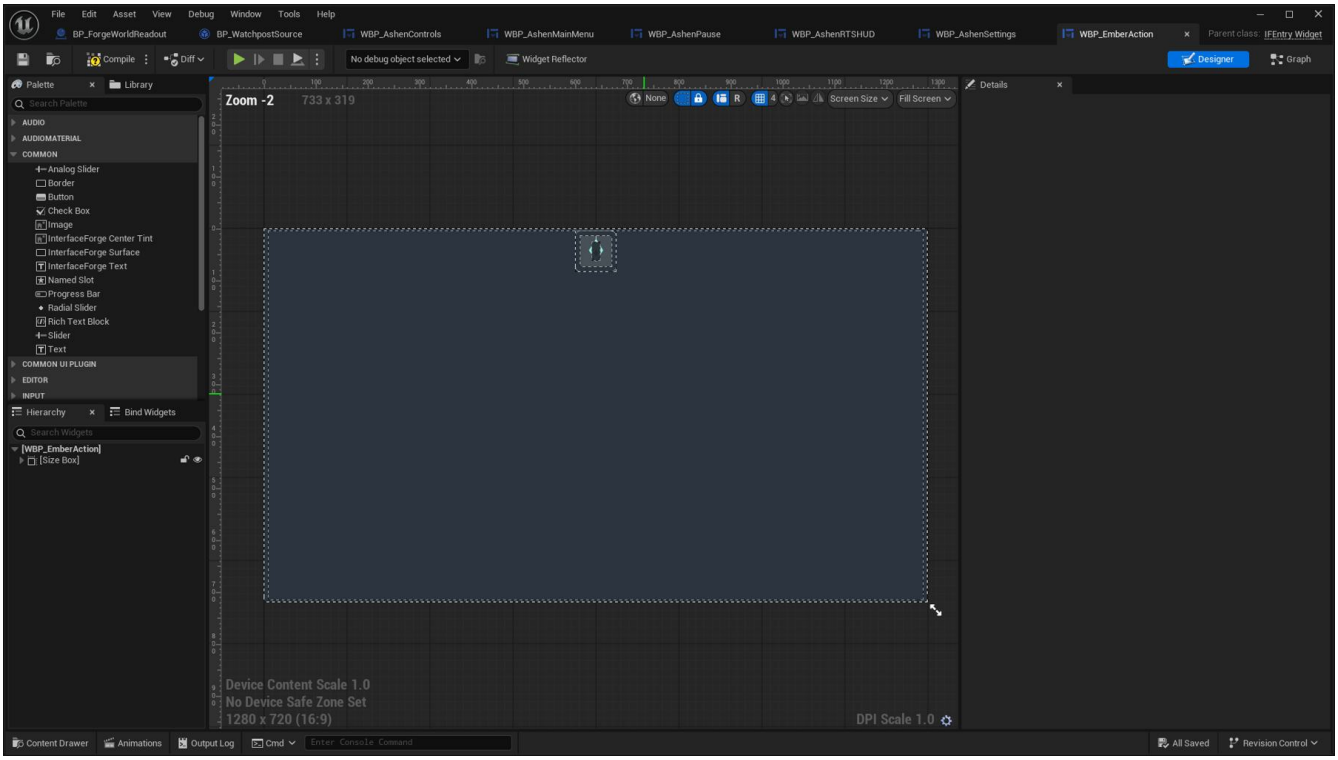
WBP_EmberAction

`/Game/UI/Blueprint/WBP_EmberAction.WBP_EmberAction` → `IFEntryWidget` · entry.



WBP_EmberAction - entry

Open full-size image - Isolated saved widget; entries use explicit sample presentation data. Layout previews do not replace the live-window images above.



WBP_EmberAction in Unreal Designer

Open full-size image - Actual asset editor; design-time values can differ from runtime bindings.

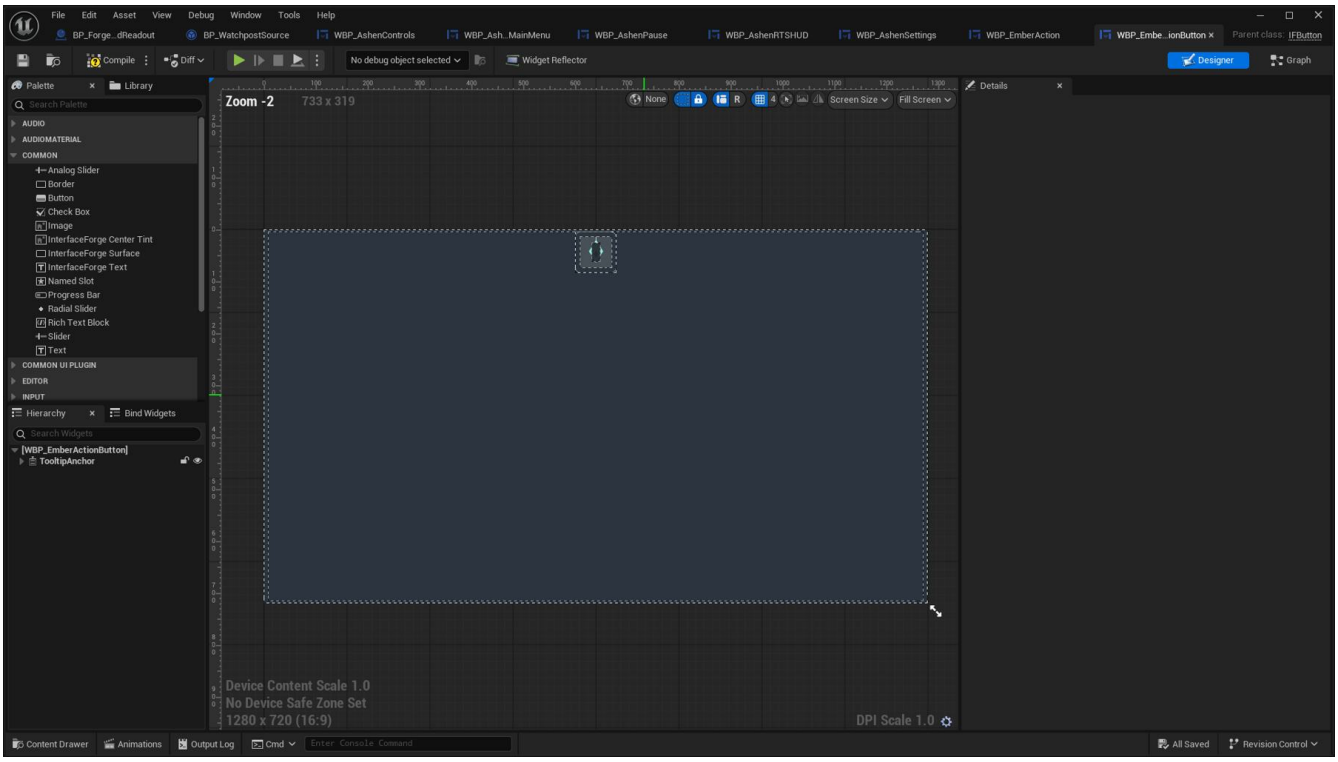
WBP_EmberActionButton

`/Game/UI/Blueprint/WBP_EmberActionButton.WBP_EmberActionButton` → `IFButton` · button.



WBP_EmberActionButton - button

Open full-size image - Isolated saved widget; entries use explicit sample presentation data. Layout previews do not replace the live-window images above.



WBP_EmberActionButton in Unreal Designer

Open full-size image - Actual asset editor; design-time values can differ from runtime bindings.

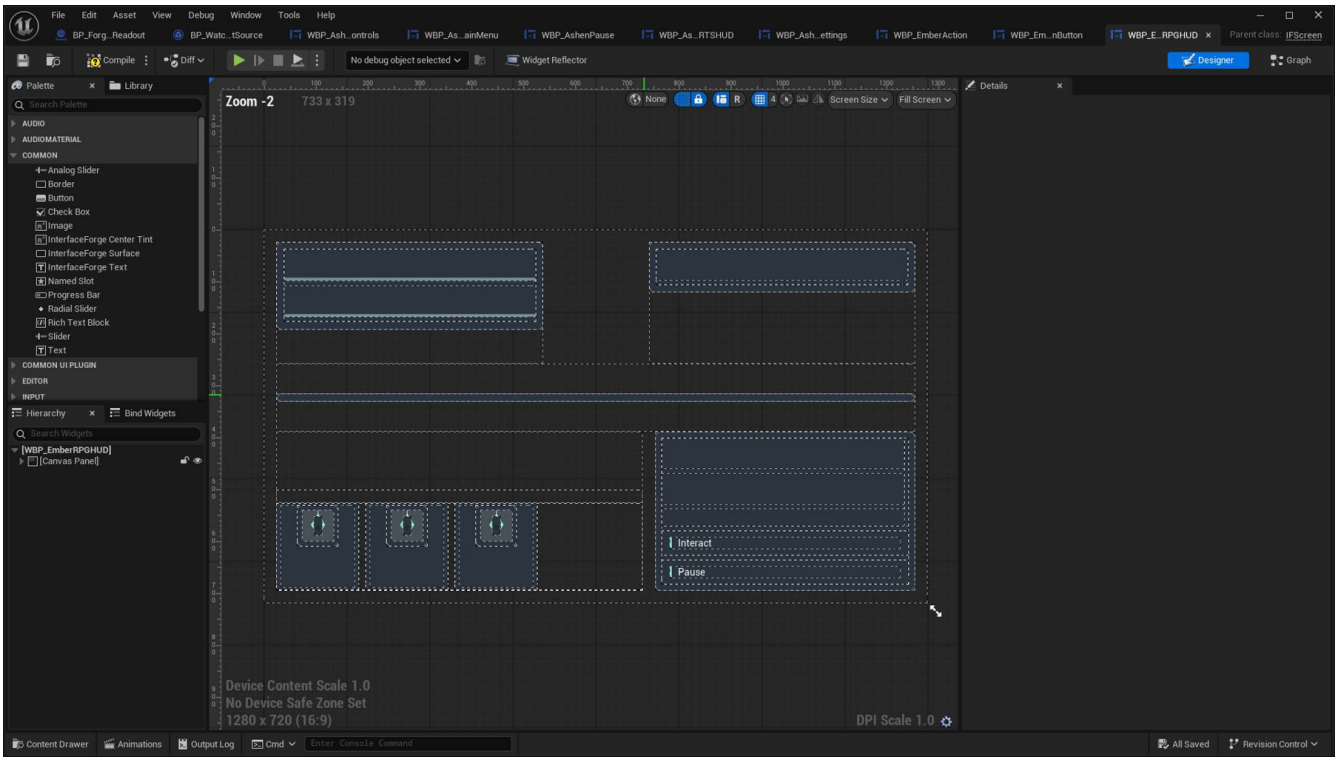
WBP_EmberRPGHUD

`/Game/UI/Blueprint/WBP_EmberRPGHUD.WBP_EmberRPGHUD` → `IFScreen` · screen.



WBP_EmberRPGHUD - screen

Open full-size image - Live composed interface.



WBP_EmberRPGHUD in Unreal Designer

Open full-size image - Actual asset editor; design-time values can differ from runtime bindings.

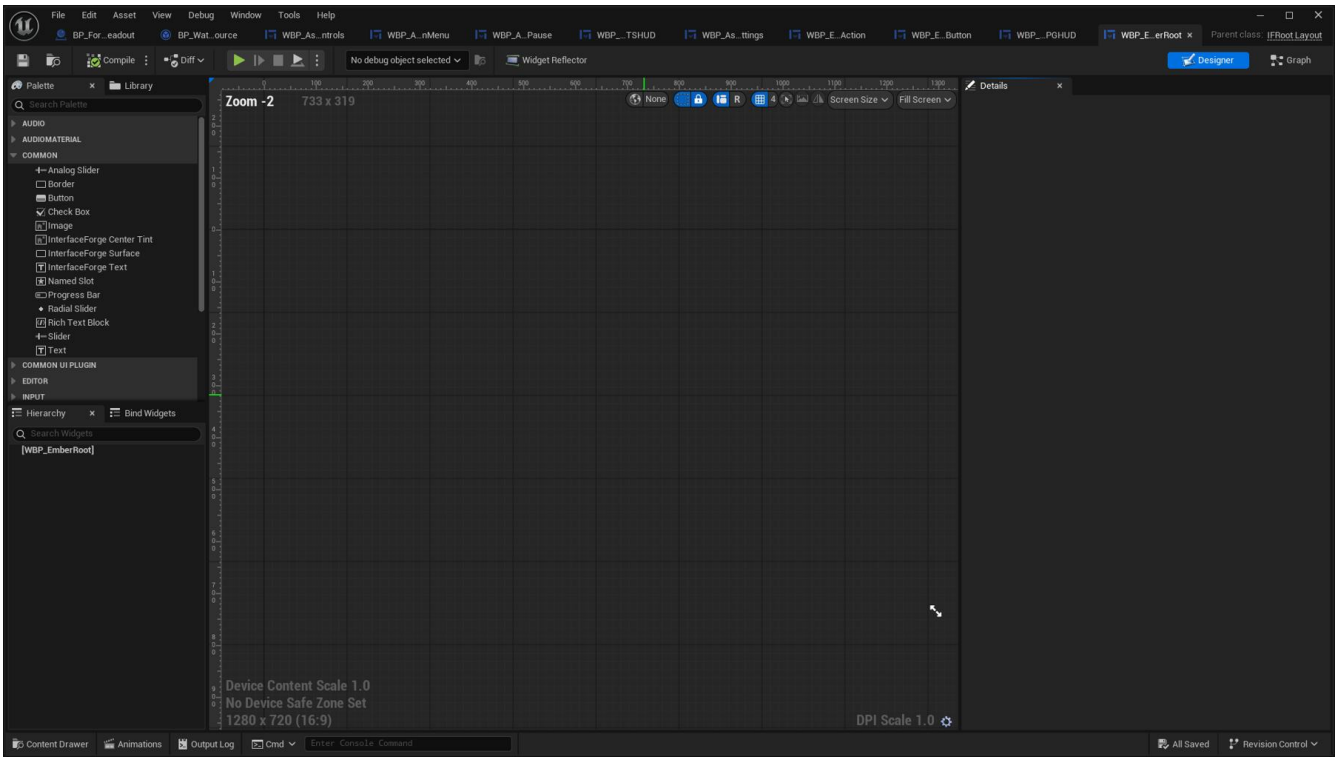
WBP_EmberRoot

`/Game/UI/Blueprint/WBP_EmberRoot.WBP_EmberRoot` → `IFRootLayout` · root. RPG root composition. A root is a layer container, not a separate menu.



WBP_EmberRoot - root

Open full-size image - Live composed interface.



WBP_EmberRoot in Unreal Designer

Open full-size image - Actual asset editor; design-time values can differ from runtime bindings.

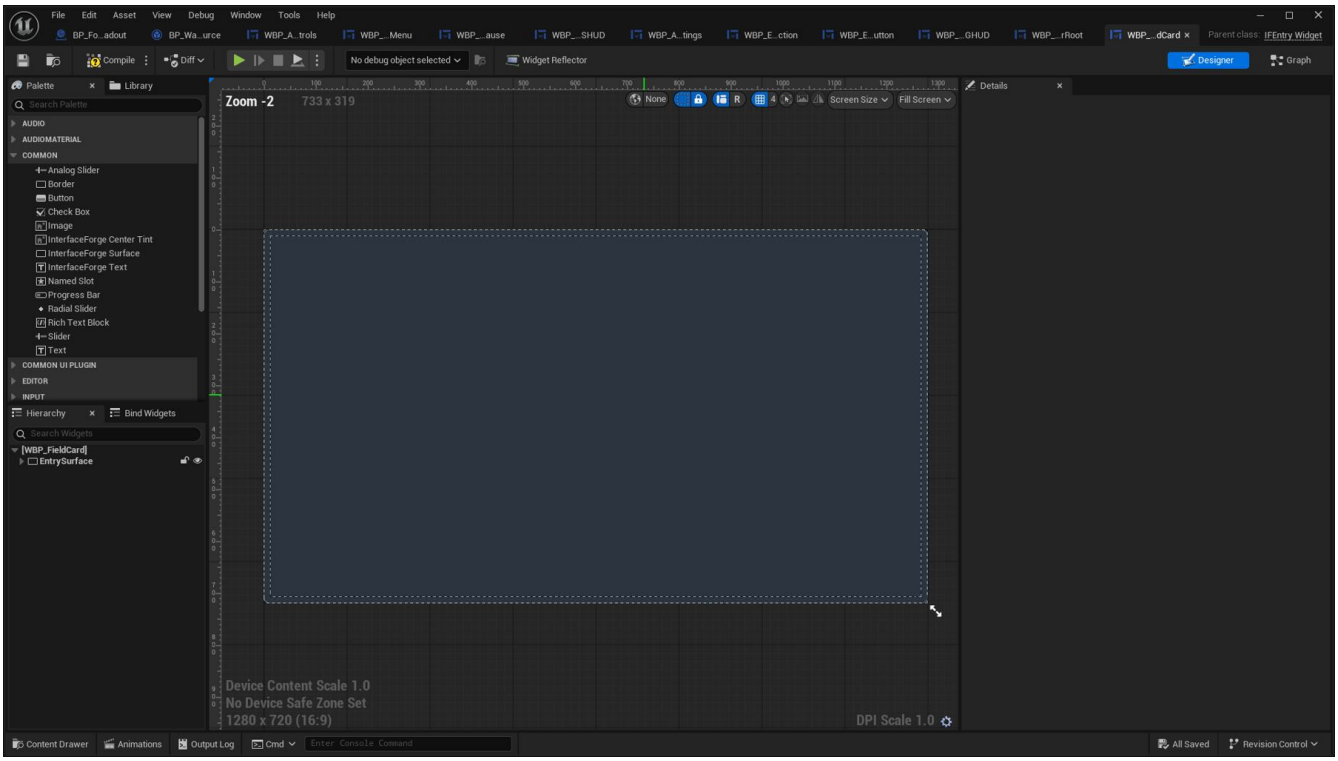
WBP_FieldCard

`/Game/UI/Blueprint/WBP_FieldCard.WBP_FieldCard` → `IFEntryWidget` · entry.



WBP_FieldCard - entry

Open full-size image - Isolated saved widget; entries use explicit sample presentation data. Layout previews do not replace the live-window images above.

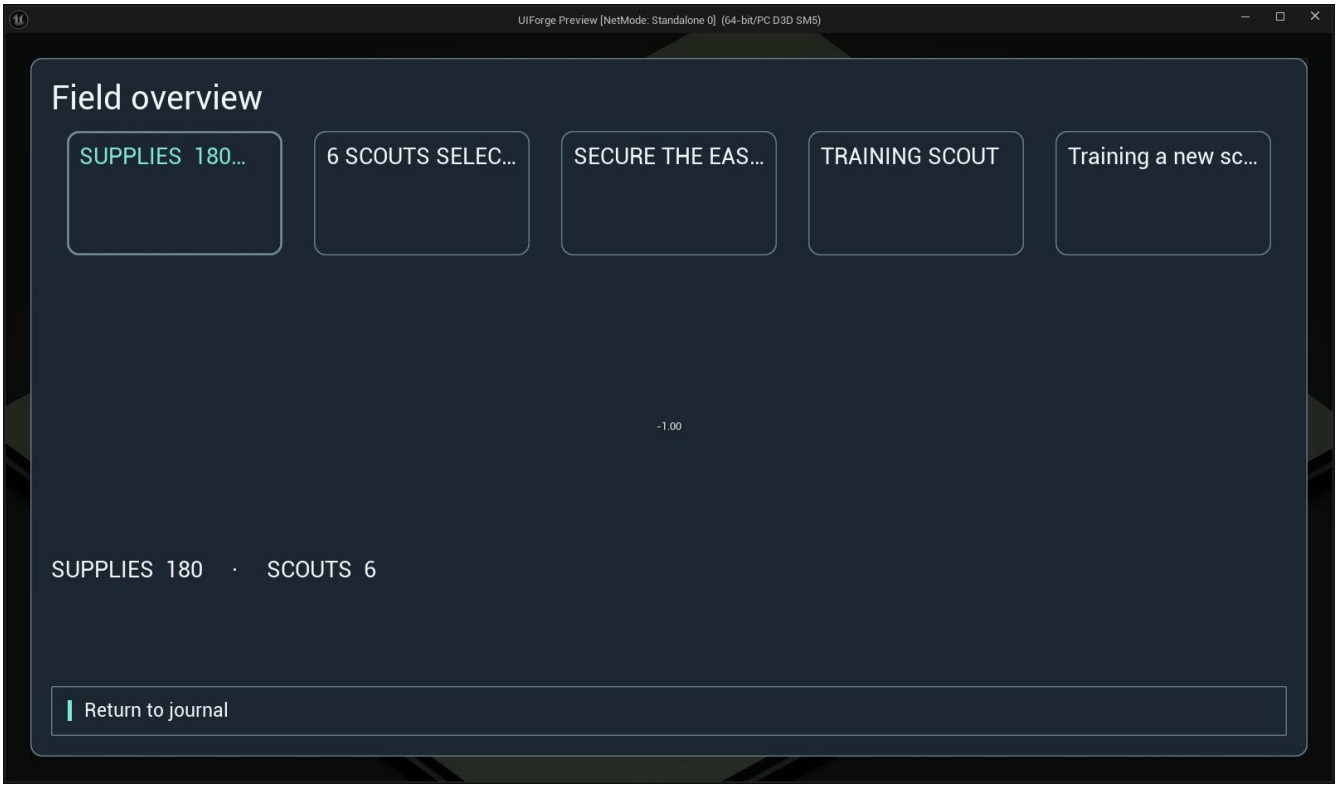


WBP_FieldCard in Unreal Designer

Open full-size image - Actual asset editor; design-time values can differ from runtime bindings.

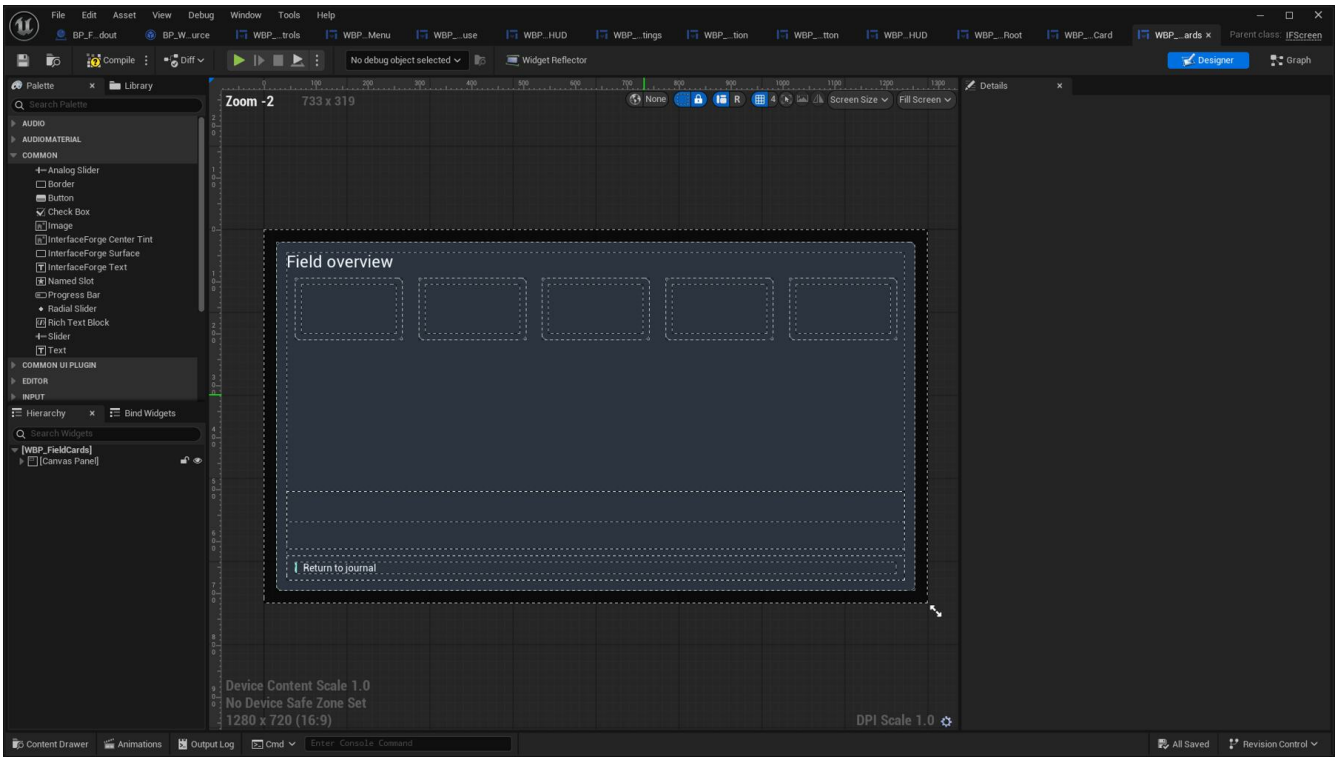
WBP_FieldCards

`/Game/UI/Blueprint/WBP_FieldCards.WBP_FieldCards` → `IFScreen` · screen.



WBP_FieldCards - screen

Open full-size image - Live composed interface.



WBP_FieldCards in Unreal Designer

Open full-size image - Actual asset editor; design-time values can differ from runtime bindings.

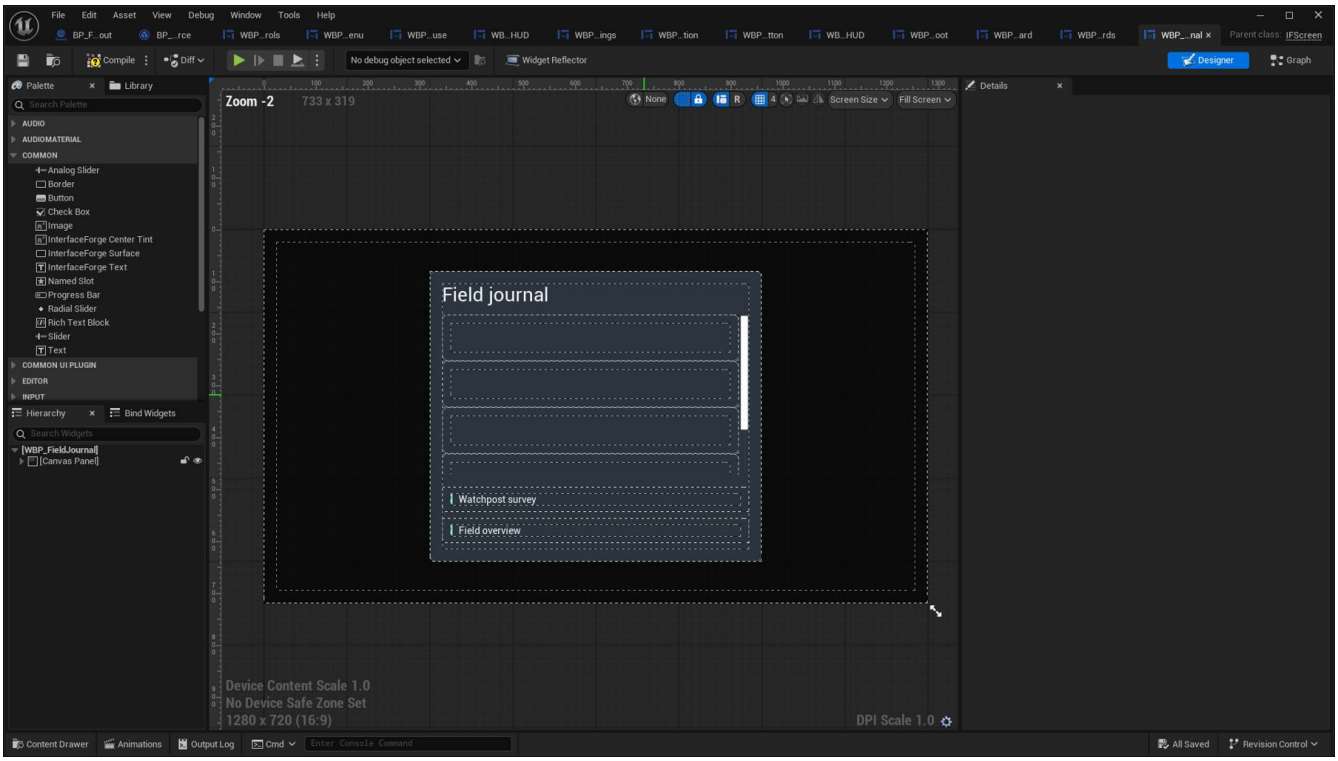
WBP_FieldJournal

`/Game/UI/Blueprint/WBP_FieldJournal.WBP_FieldJournal` → `IFScreen` · screen.



WBP_FieldJournal - screen

Open full-size image - Live composed interface.



WBP_FieldJournal in Unreal Designer

Open full-size image - Actual asset editor; design-time values can differ from runtime bindings.

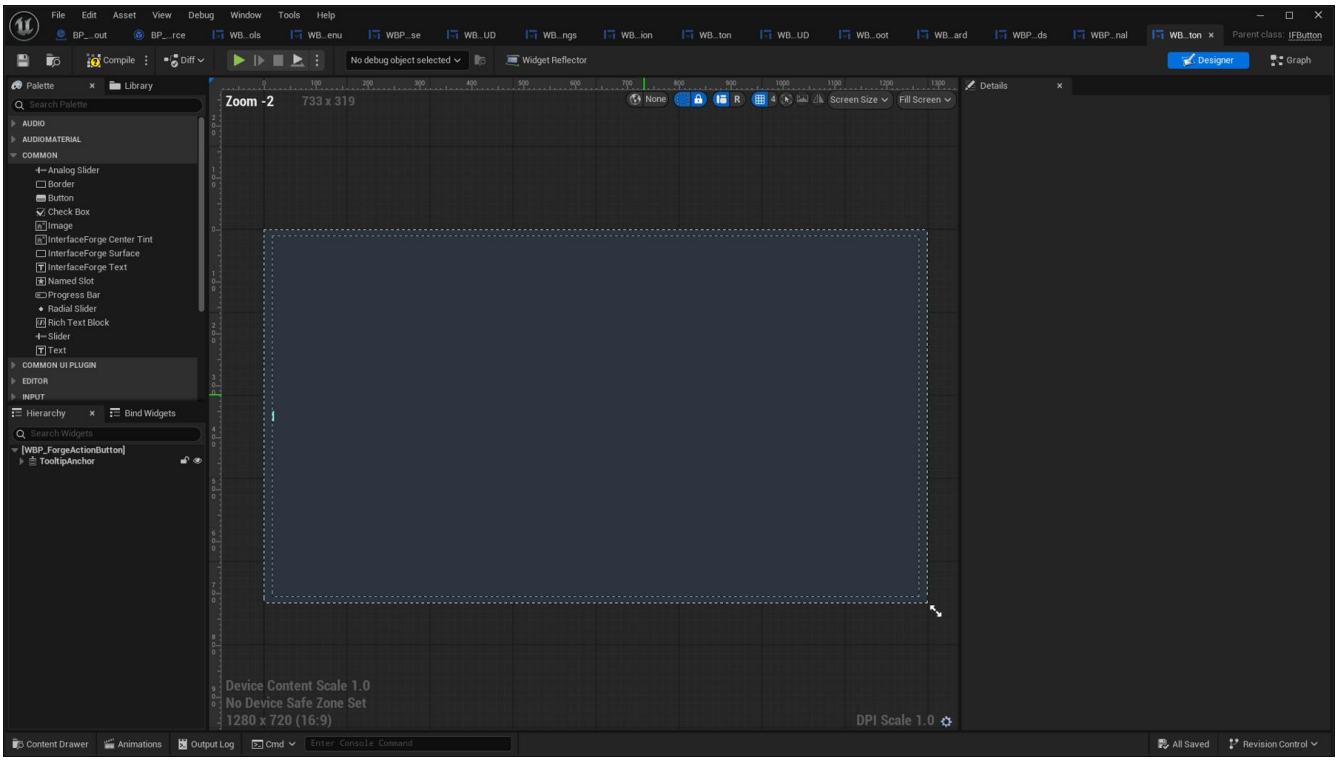
WBP_ForgeActionButton

`/Game/UI/Blueprint/WBP_ForgeActionButton.WBP_ForgeActionButton` → `IFButton` · button.



WBP_ForgeActionButton - button

Open full-size image - Isolated saved widget; entries use explicit sample presentation data. Layout previews do not replace the live-window images above.



WBP_ForgeActionButton in Unreal Designer

Open full-size image - Actual asset editor; design-time values can differ from runtime bindings.

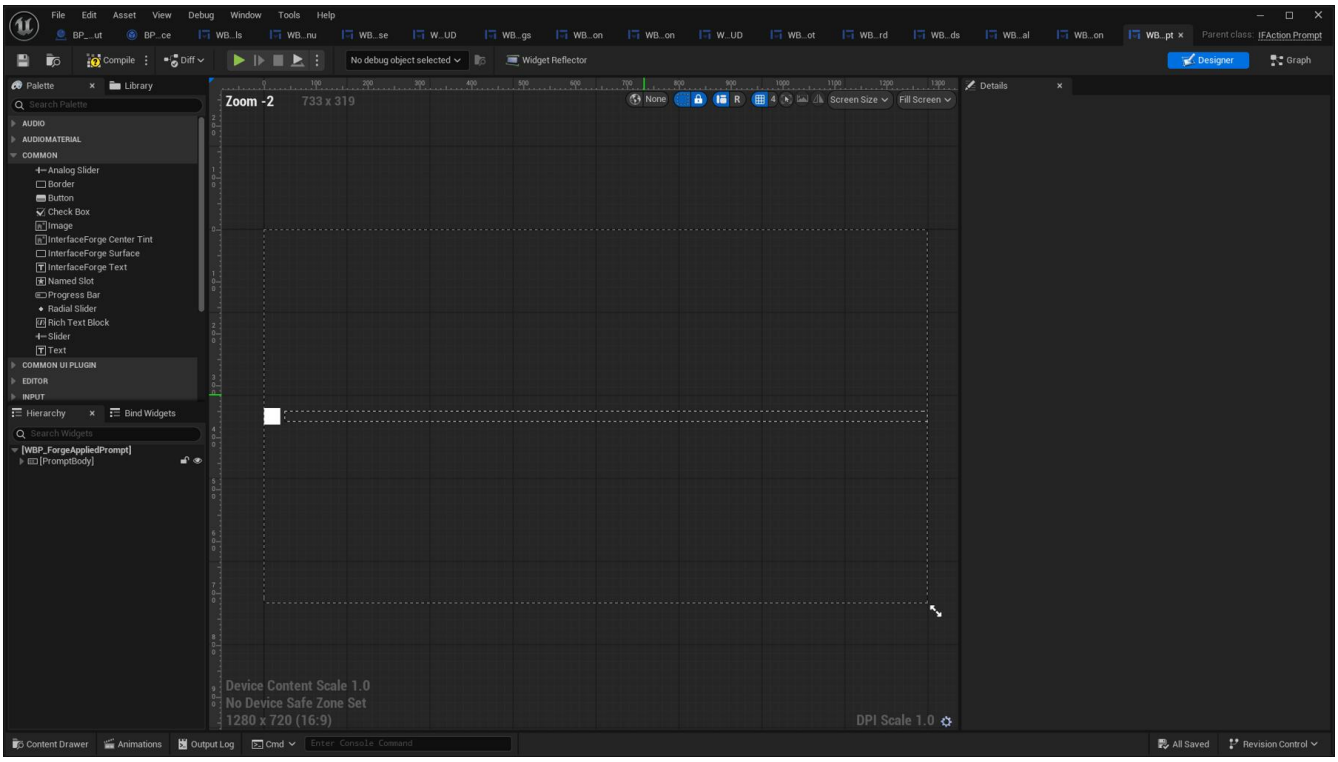
WBP_ForgeAppliedPrompt

`/Game/UI/Blueprint/WBP_ForgeAppliedPrompt.WBP_ForgeAppliedPrompt` → `IFActionPrompt` · widget.



WBP_ForgeAppliedPrompt - widget

Open full-size image - Isolated saved widget; entries use explicit sample presentation data. Layout previews do not replace the live-window images above.

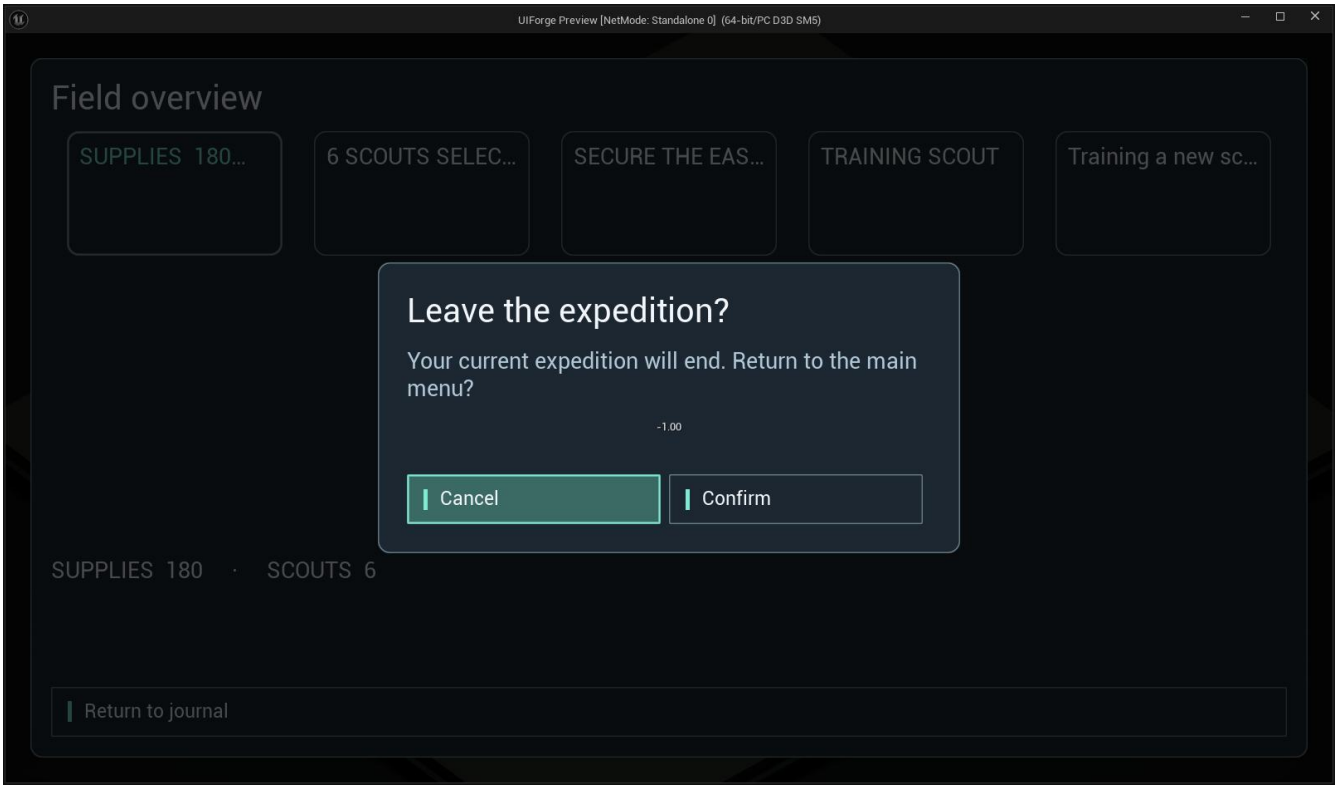


WBP_ForgeAppliedPrompt in Unreal Designer

Open full-size image - Actual asset editor; design-time values can differ from runtime bindings.

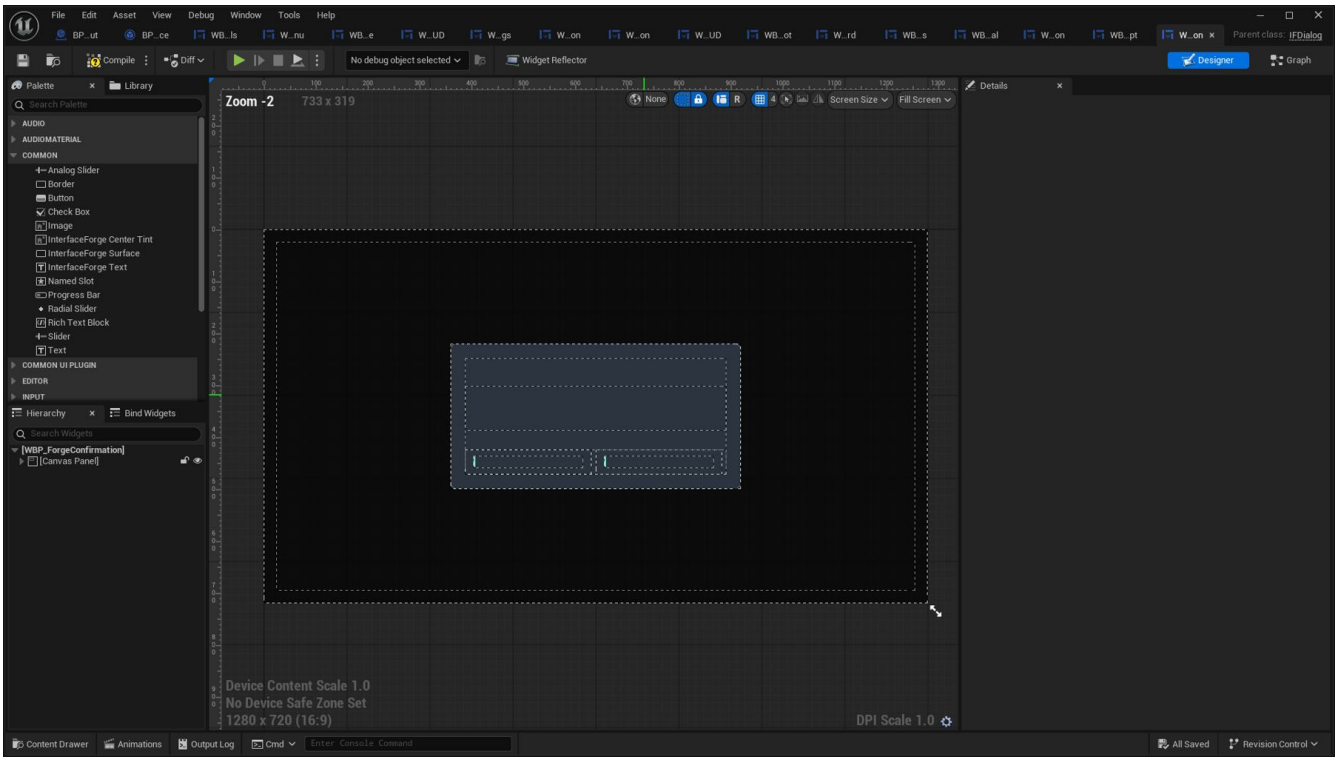
WBP_ForgeConfirmation

`/Game/UI/Blueprint/WBP_ForgeConfirmation.WBP_ForgeConfirmation` → `IFDialog` · widget.



WBP_ForgeConfirmation - widget

Open full-size image - Live composed interface.

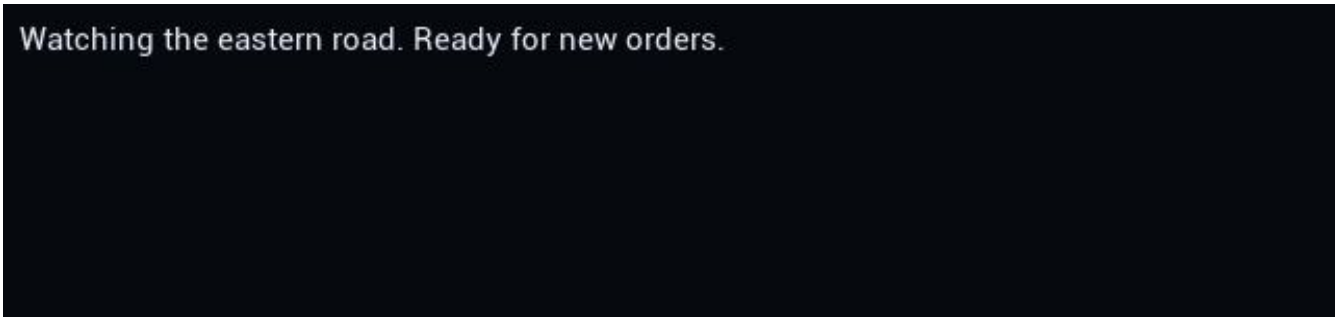


WBP_ForgeConfirmation in Unreal Designer

Open full-size image - Actual asset editor; design-time values can differ from runtime bindings.

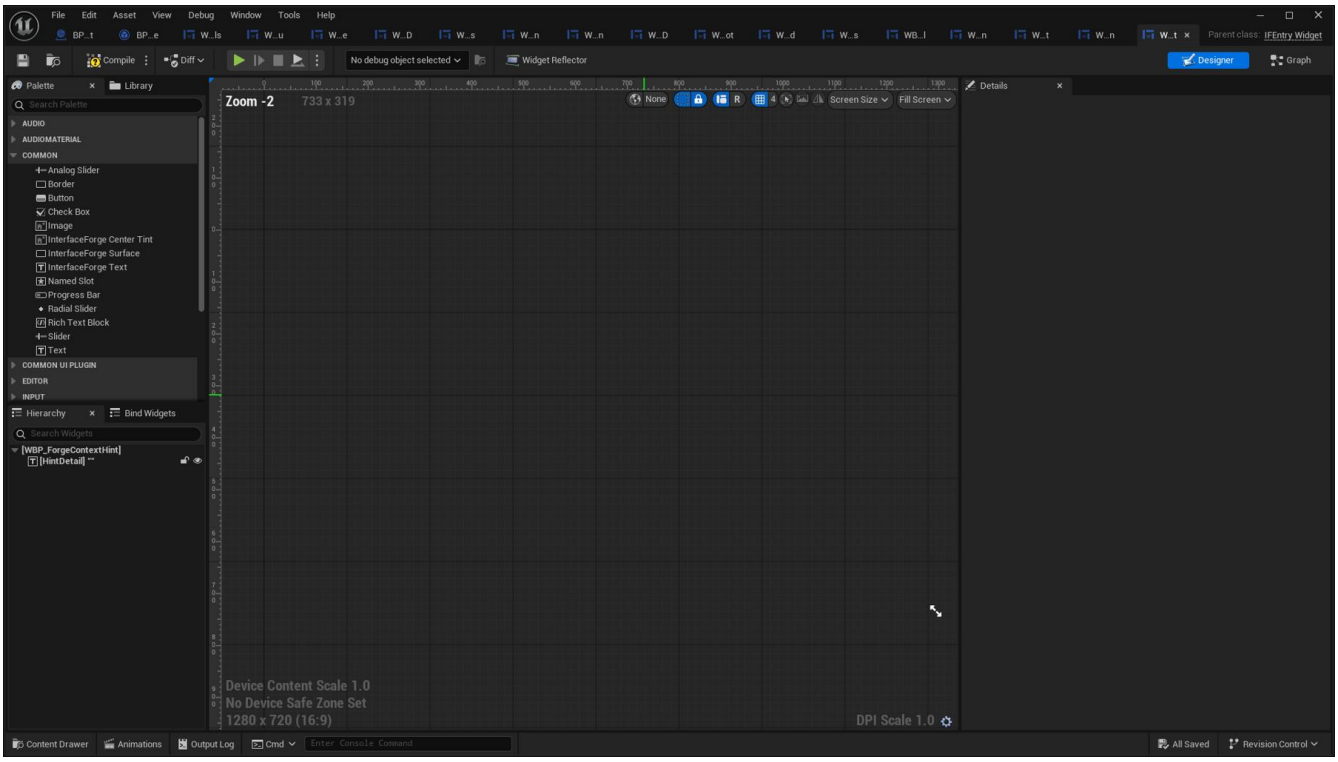
WBP_ForgeContextHint

`/Game/UI/Blueprint/WBP_ForgeContextHint.WBP_ForgeContextHint` → `IFEntryWidget` · entry.



WBP_ForgeContextHint - entry

Open full-size image - Isolated saved widget; entries use explicit sample presentation data. Layout previews do not replace the live-window images above.

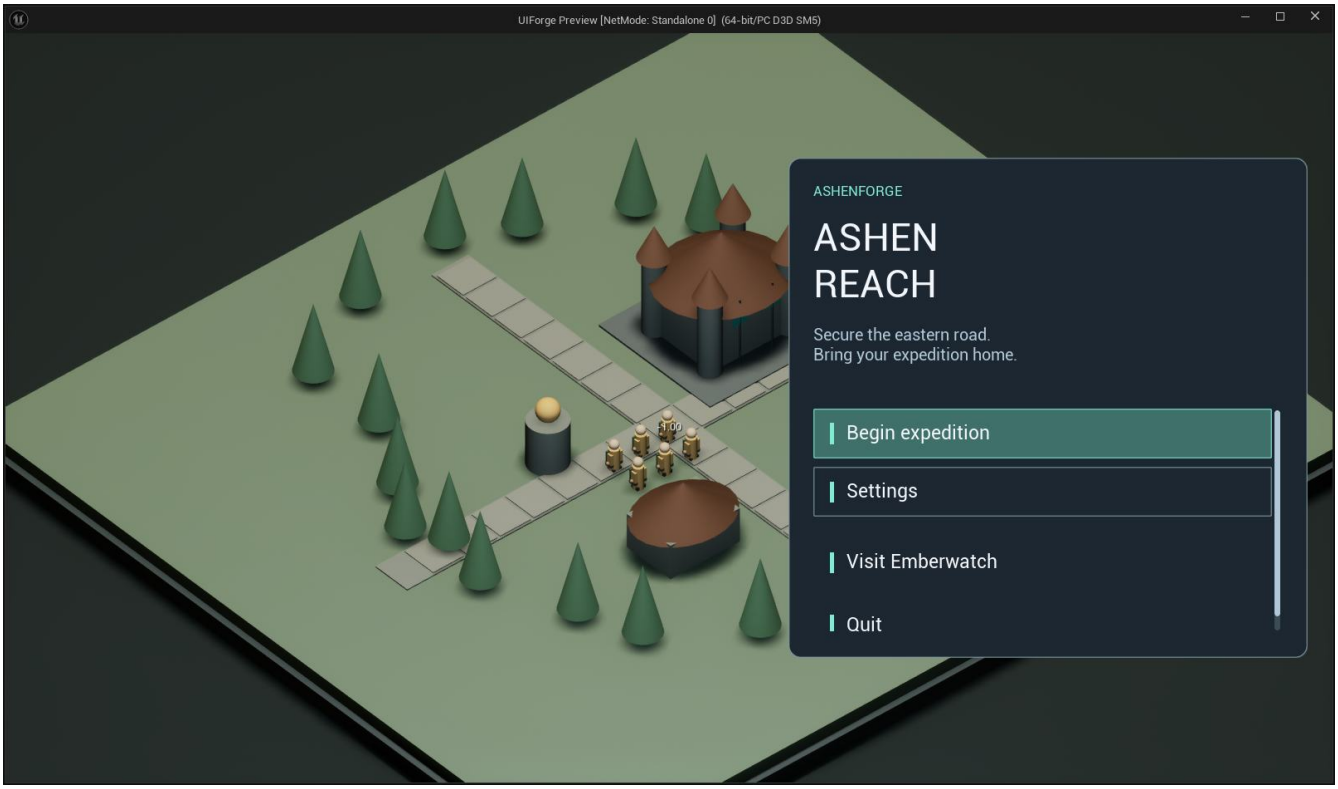


WBP_ForgeContextHint in Unreal Designer

Open full-size image - Actual asset editor; design-time values can differ from runtime bindings.

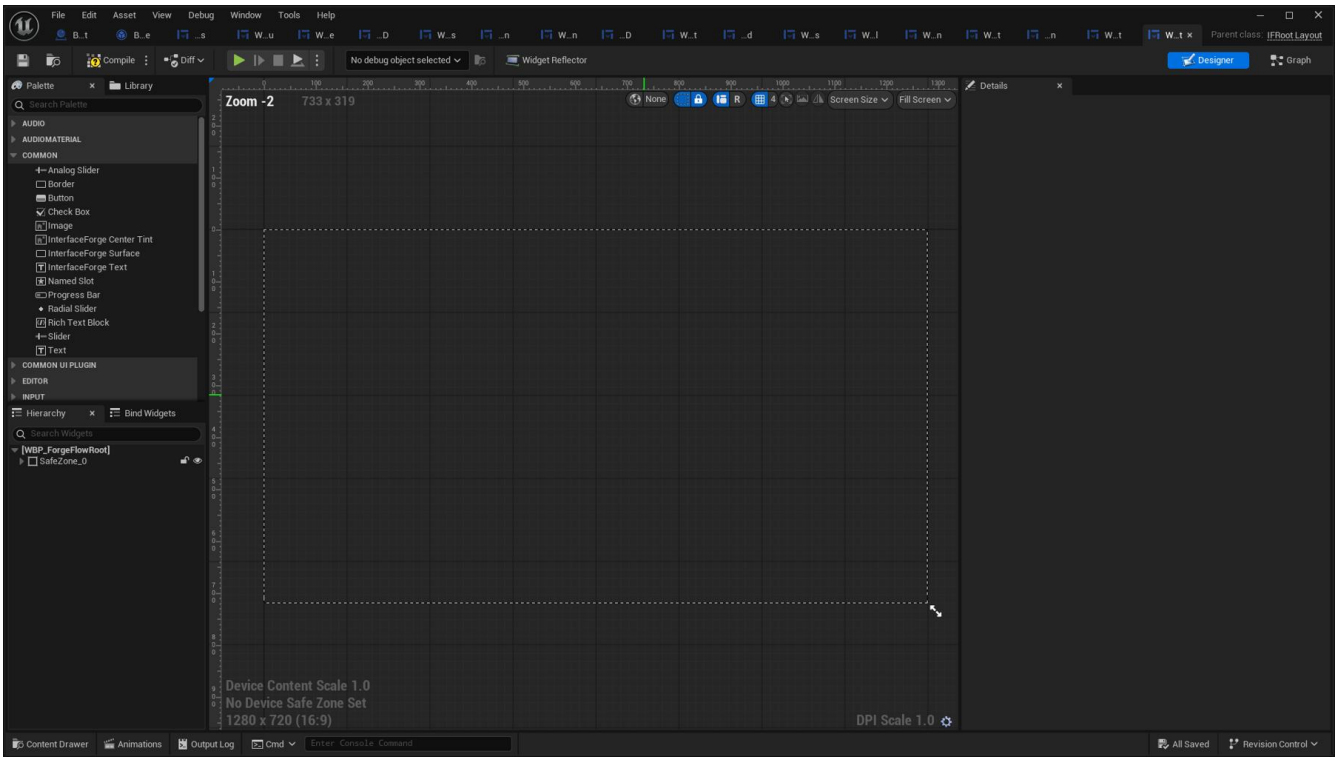
WBP_ForgeFlowRoot

`/Game/UI/Blueprint/WBP_ForgeFlowRoot.WBP_ForgeFlowRoot` → `IFRootLayout` · root. Developer-created Designer root registering HUD/Menu/Modal and notification layers.



WBP_ForgeFlowRoot - root

Open full-size image - Live composed interface.

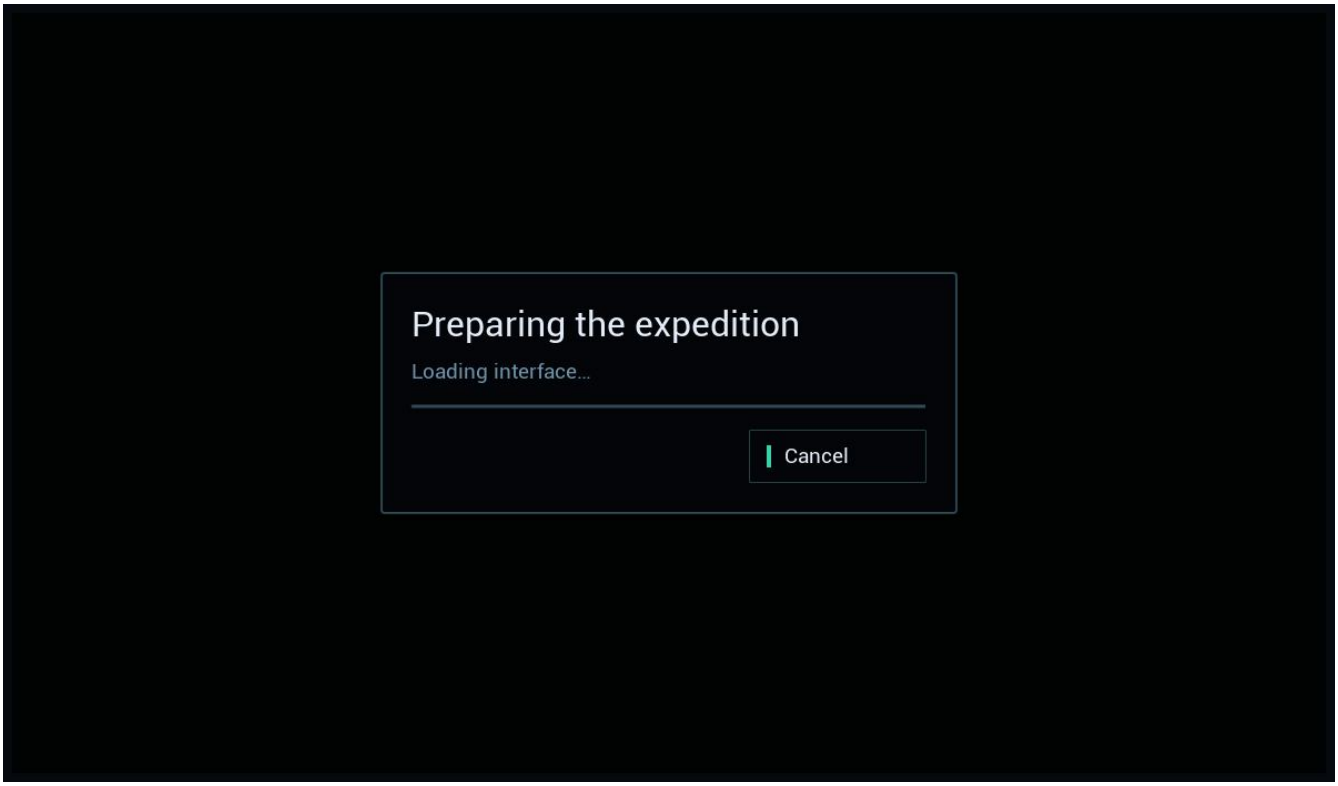


WBP_ForgeFlowRoot in Unreal Designer

Open full-size image - Actual asset editor; design-time values can differ from runtime bindings.

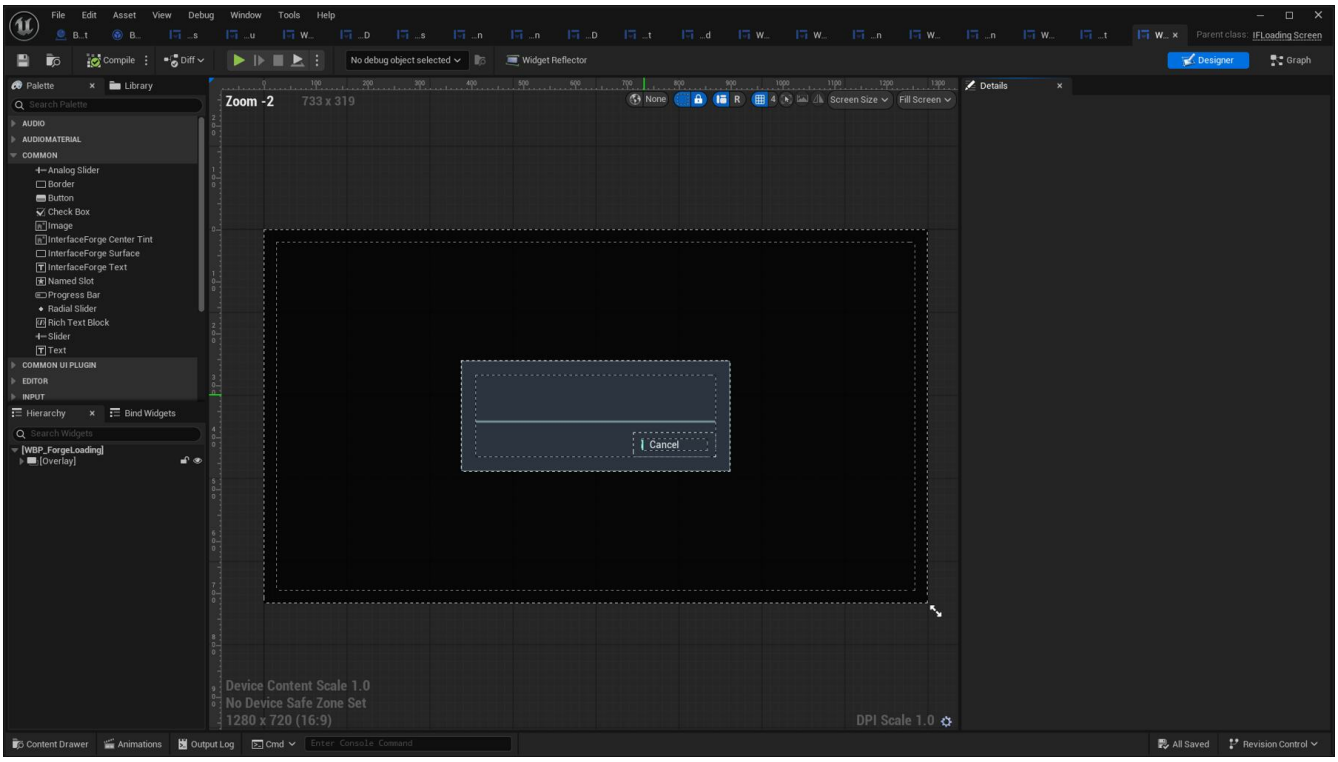
WBP_ForgeLoading

`/Game/UI/Blueprint/WBP_ForgeLoading.WBP_ForgeLoading` → `IFLoadingScreen` · widget.



WBP_ForgeLoading - widget

Open full-size image - Isolated saved widget; entries use explicit sample presentation data. Layout previews do not replace the live-window images above.



WBP_ForgeLoading in Unreal Designer

Open full-size image - Actual asset editor; design-time values can differ from runtime bindings.

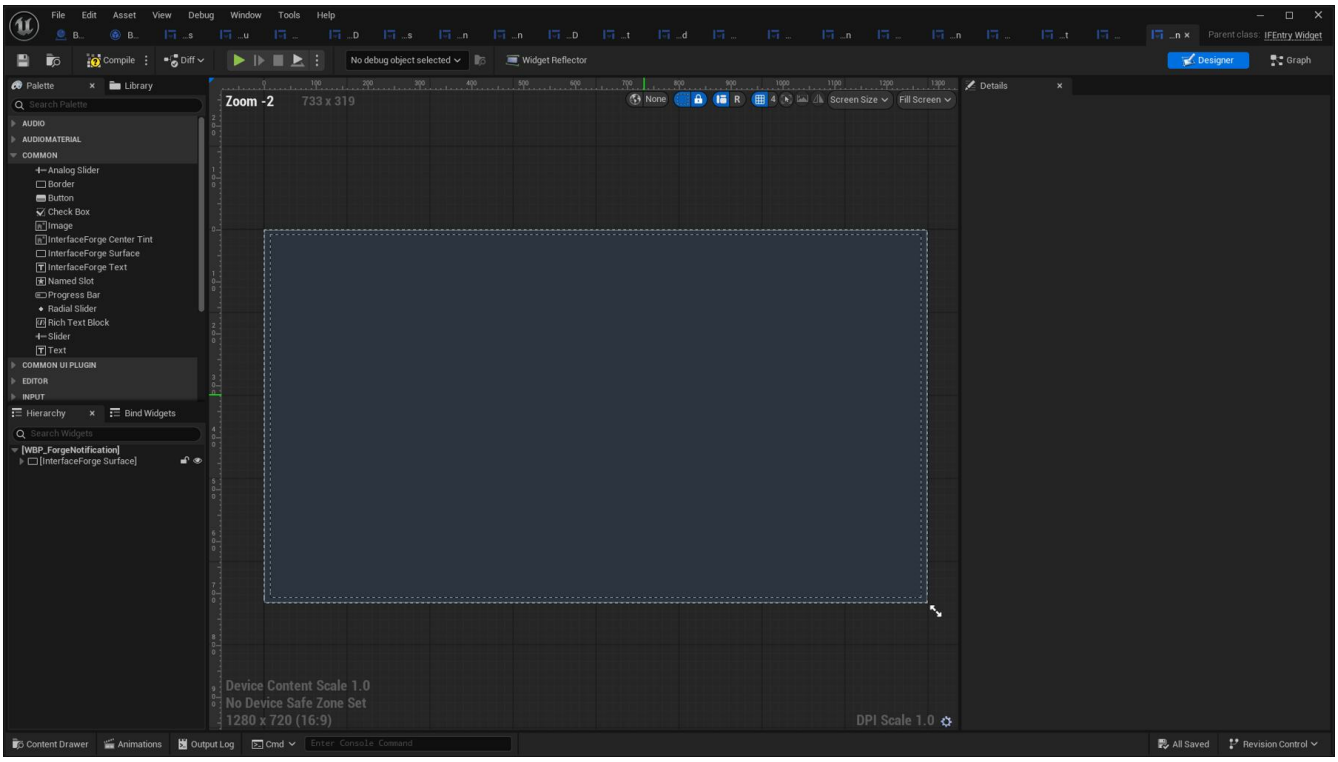
WBP_ForgeNotification

`/Game/UI/Blueprint/WBP_ForgeNotification.WBP_ForgeNotification` → `IFEntryWidget` · entry.



WBP_ForgeNotification - entry

Open full-size image - Isolated saved widget; entries use explicit sample presentation data. Layout previews do not replace the live-window images above.

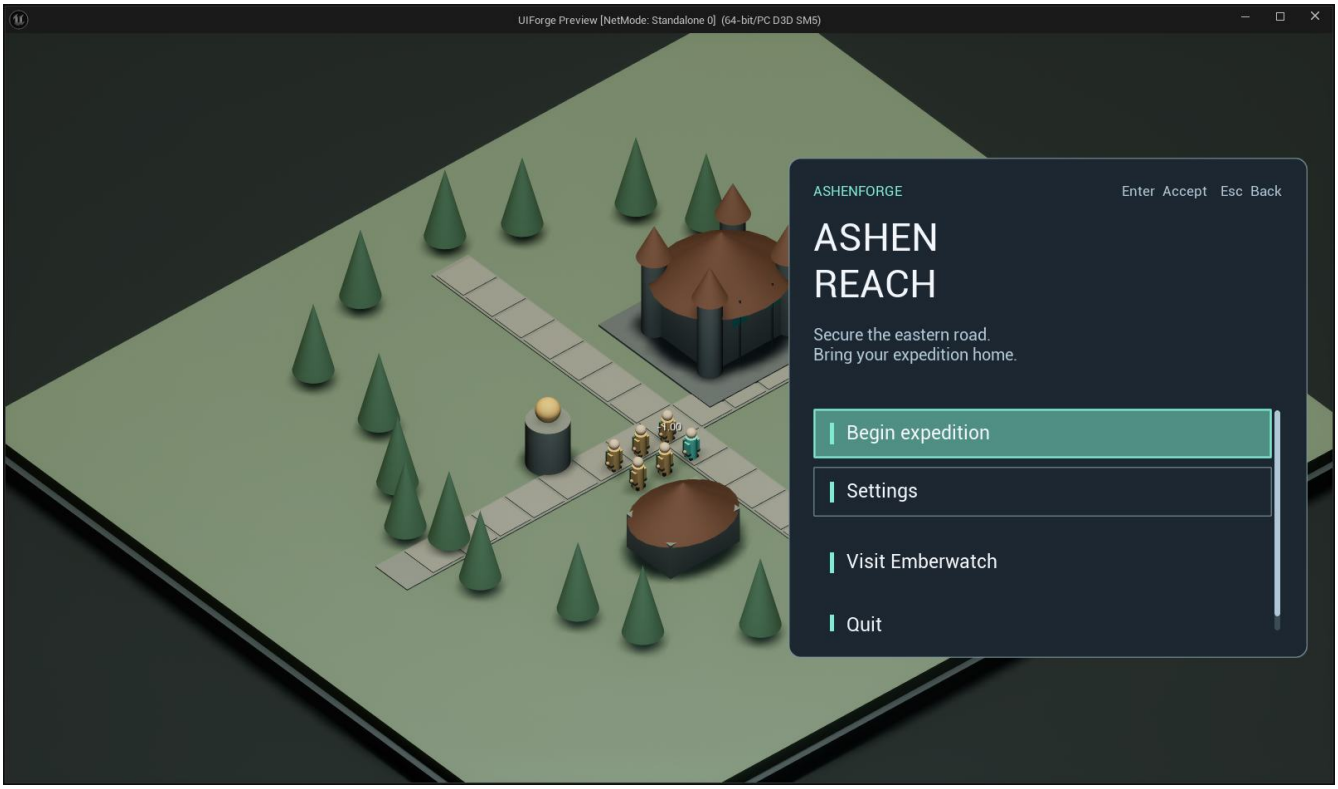


WBP_ForgeNotification in Unreal Designer

Open full-size image - Actual asset editor; design-time values can differ from runtime bindings.

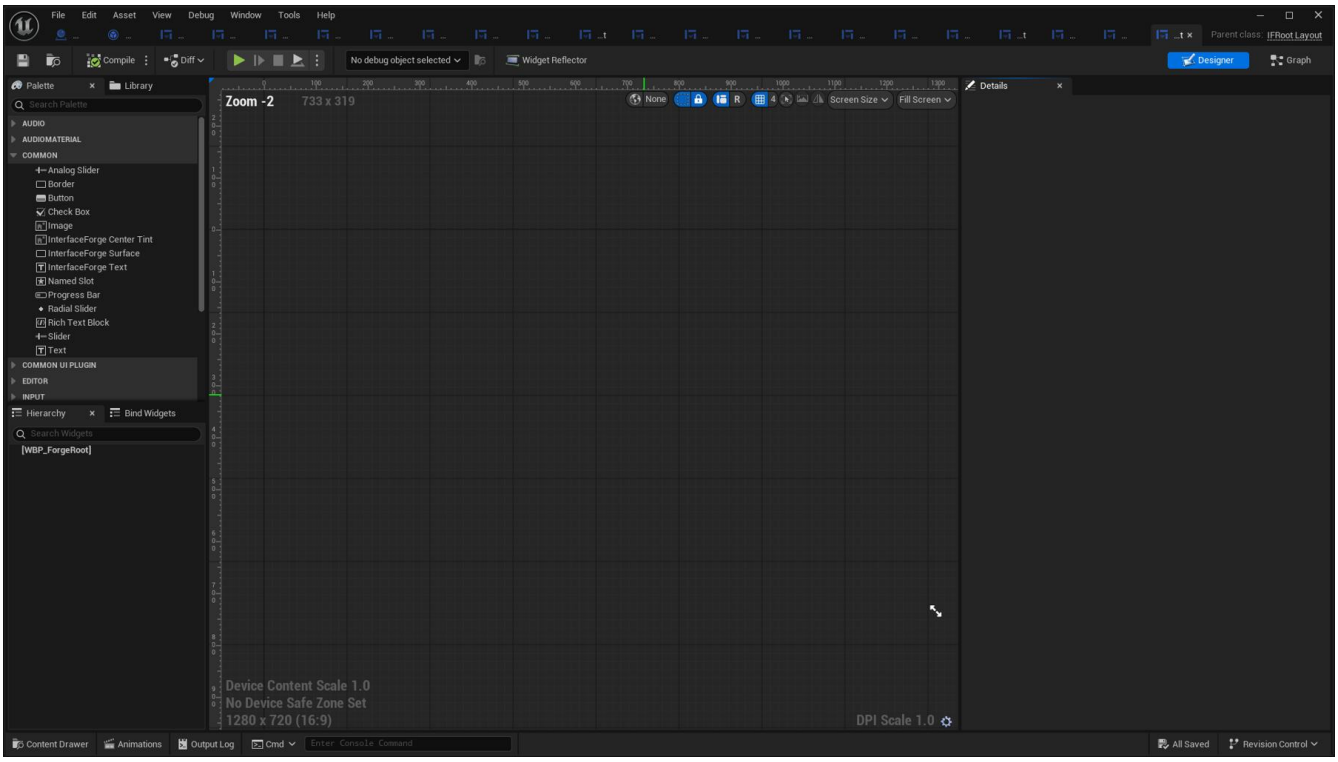
WBP_ForgeRoot

`/Game/UI/Blueprint/WBP_ForgeRoot.WBP_ForgeRoot` → `IFRootLayout` · root. Native root defaults with a Blueprint-configured theme/entry class.



WBP_ForgeRoot - root

Open full-size image - Live composed interface.



WBP_ForgeRoot in Unreal Designer

Open full-size image - Actual asset editor; design-time values can differ from runtime bindings.

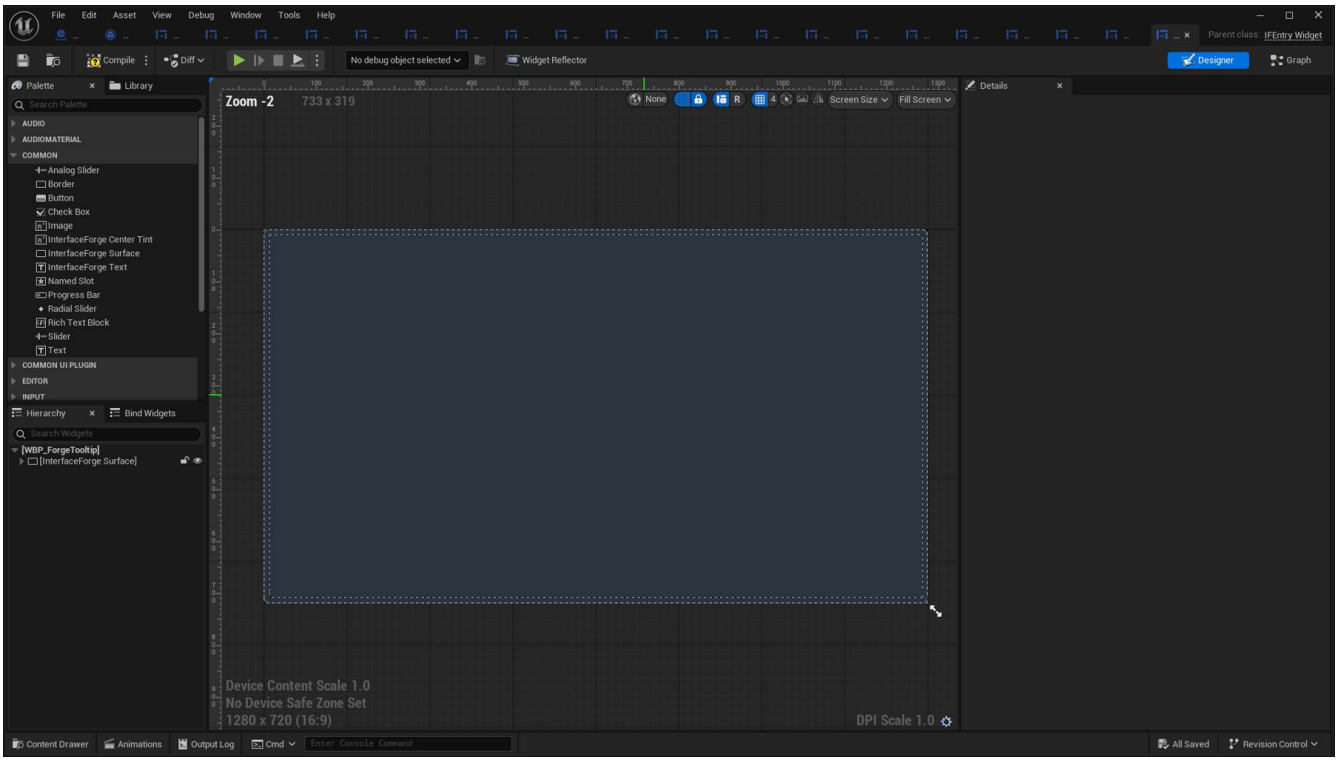
WBP_ForgeTooltip

`/Game/UI/Blueprint/WBP_ForgeTooltip.WBP_ForgeTooltip` → `IFEntryWidget` · entry.



WBP_ForgeTooltip - entry

Open full-size image - Isolated saved widget; entries use explicit sample presentation data. Layout previews do not replace the live-window images above.



WBP_ForgeTooltip in Unreal Designer

Open full-size image - Actual asset editor; design-time values can differ from runtime bindings.

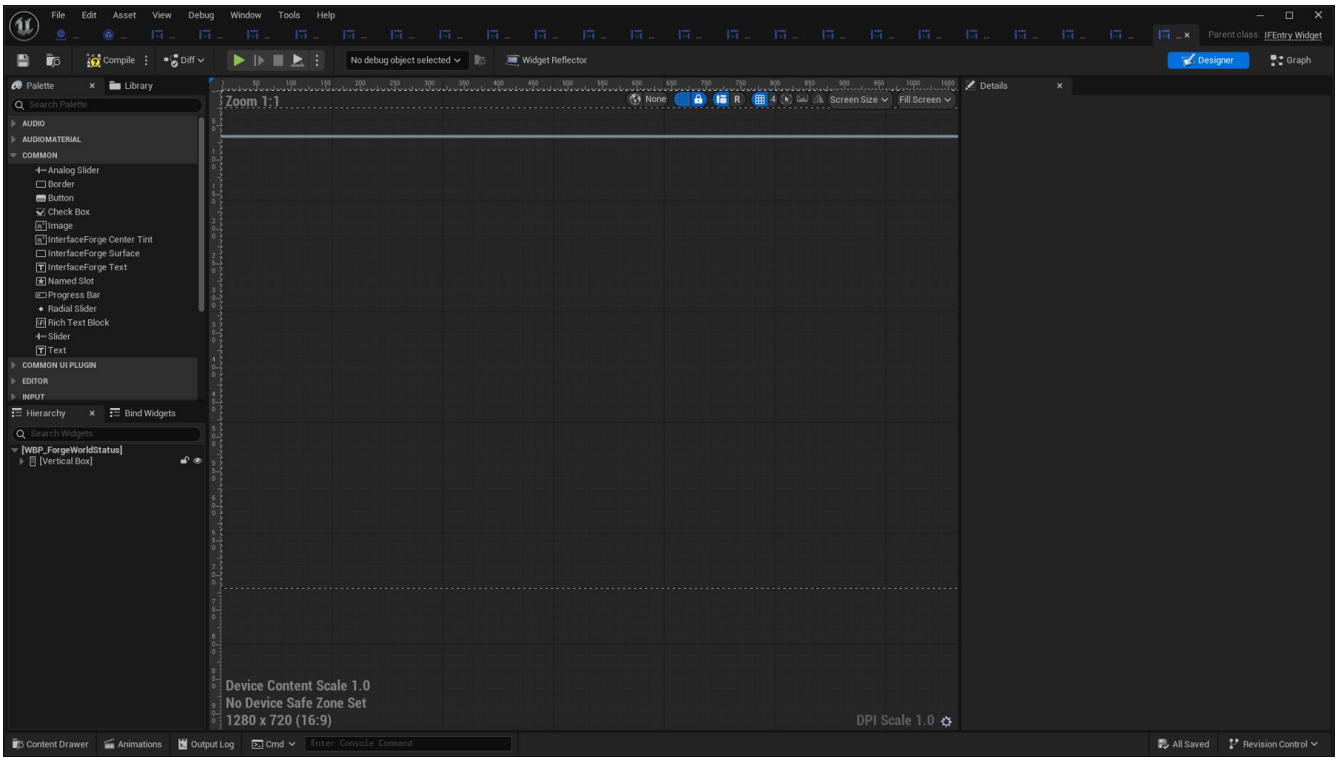
WBP_ForgeWorldStatus

`/Game/UI/Blueprint/WBP_ForgeWorldStatus.WBP_ForgeWorldStatus` → `IFEntryWidget` · entry.



WBP_ForgeWorldStatus - entry

Open full-size image - Isolated saved widget; entries use explicit sample presentation data. Layout previews do not replace the live-window images above.



WBP_ForgeWorldStatus in Unreal Designer

Open full-size image - Actual asset editor; design-time values can differ from runtime bindings.

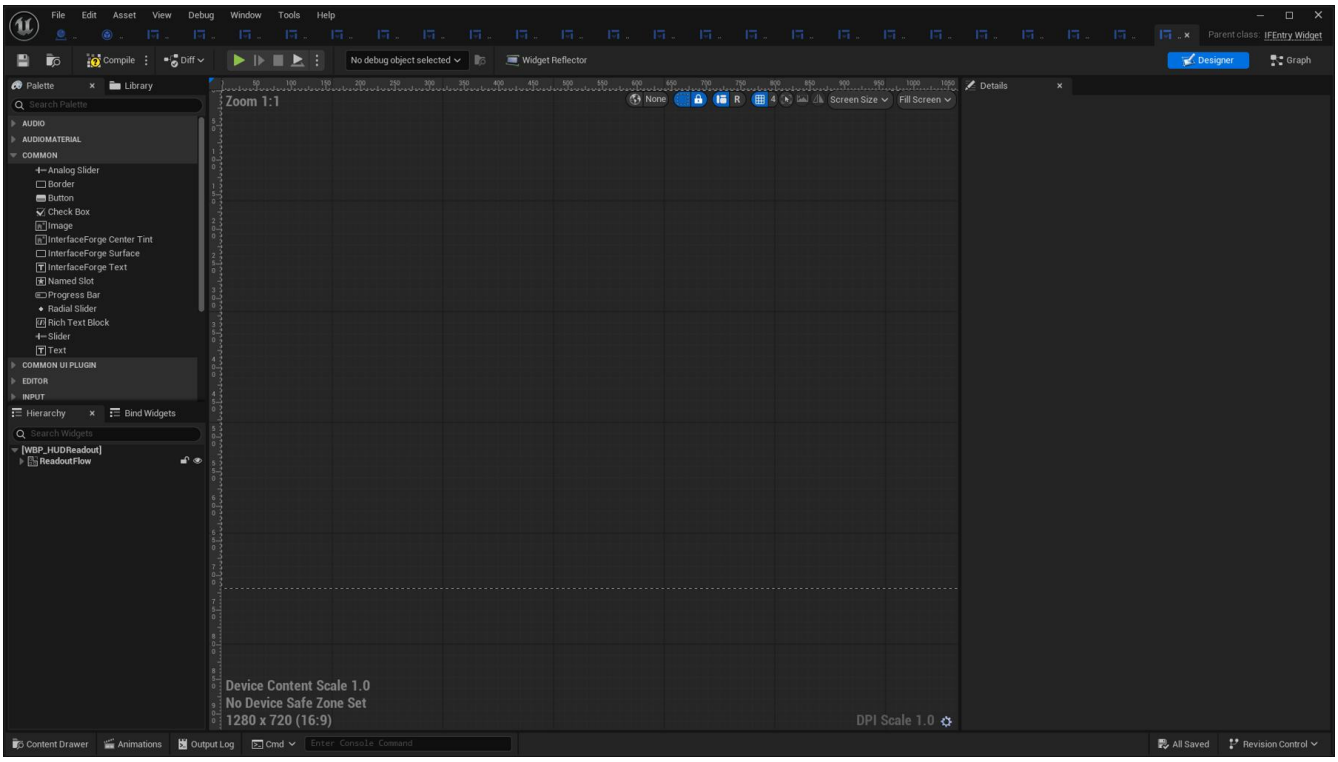
WBP_HUDReadout

`/Game/UI/Blueprint/WBP_HUDReadout.WBP_HUDReadout` → `IFEntryWidget` · entry.



WBP_HUDReadout - entry

Open full-size image - Isolated saved widget; entries use explicit sample presentation data. Layout previews do not replace the live-window images above.



WBP_HUDReadout in Unreal Designer

Open full-size image - Actual asset editor; design-time values can differ from runtime bindings.

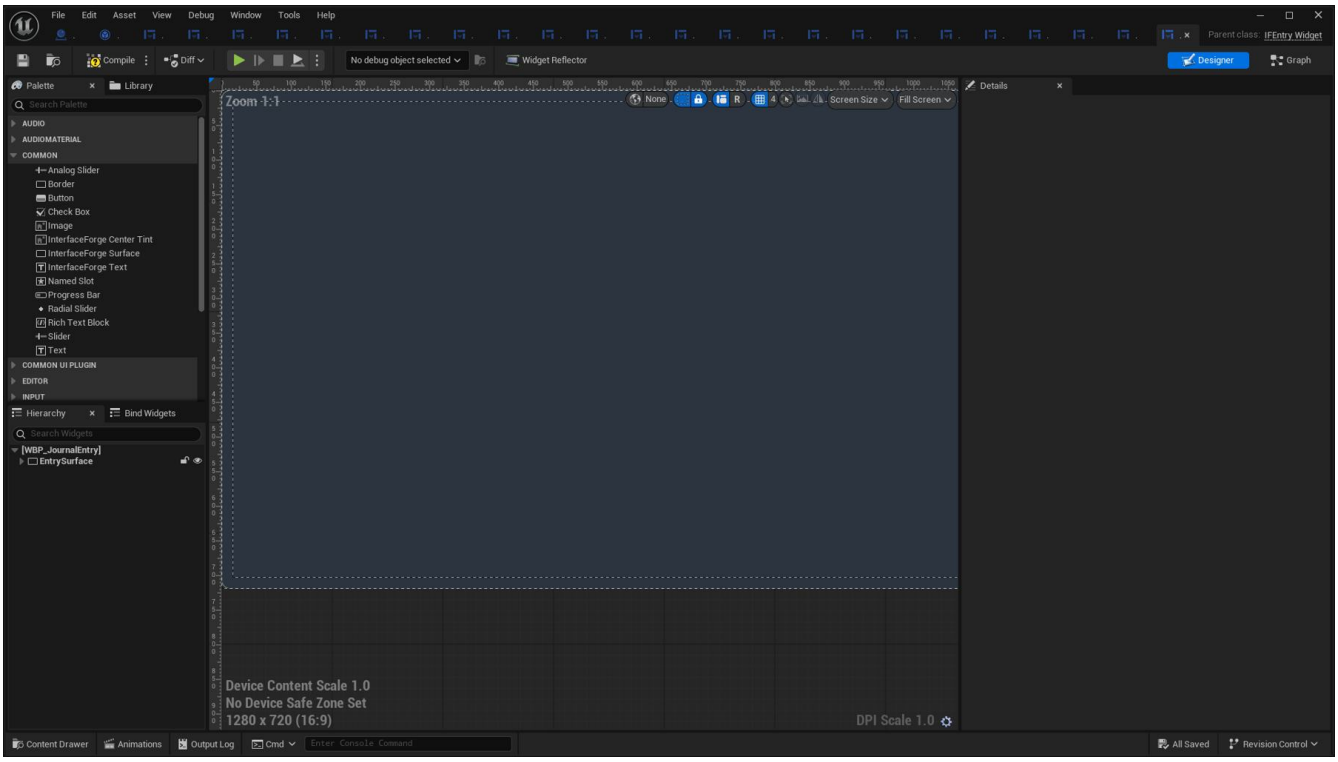
WBP_JournalEntry

`/Game/UI/Blueprint/WBP_JournalEntry.WBP_JournalEntry` → `IFEntryWidget` · entry.



WBP_JournalEntry - entry

Open full-size image - Isolated saved widget; entries use explicit sample presentation data. Layout previews do not replace the live-window images above.



WBP_JournalEntry in Unreal Designer

Open full-size image - Actual asset editor; design-time values can differ from runtime bindings.

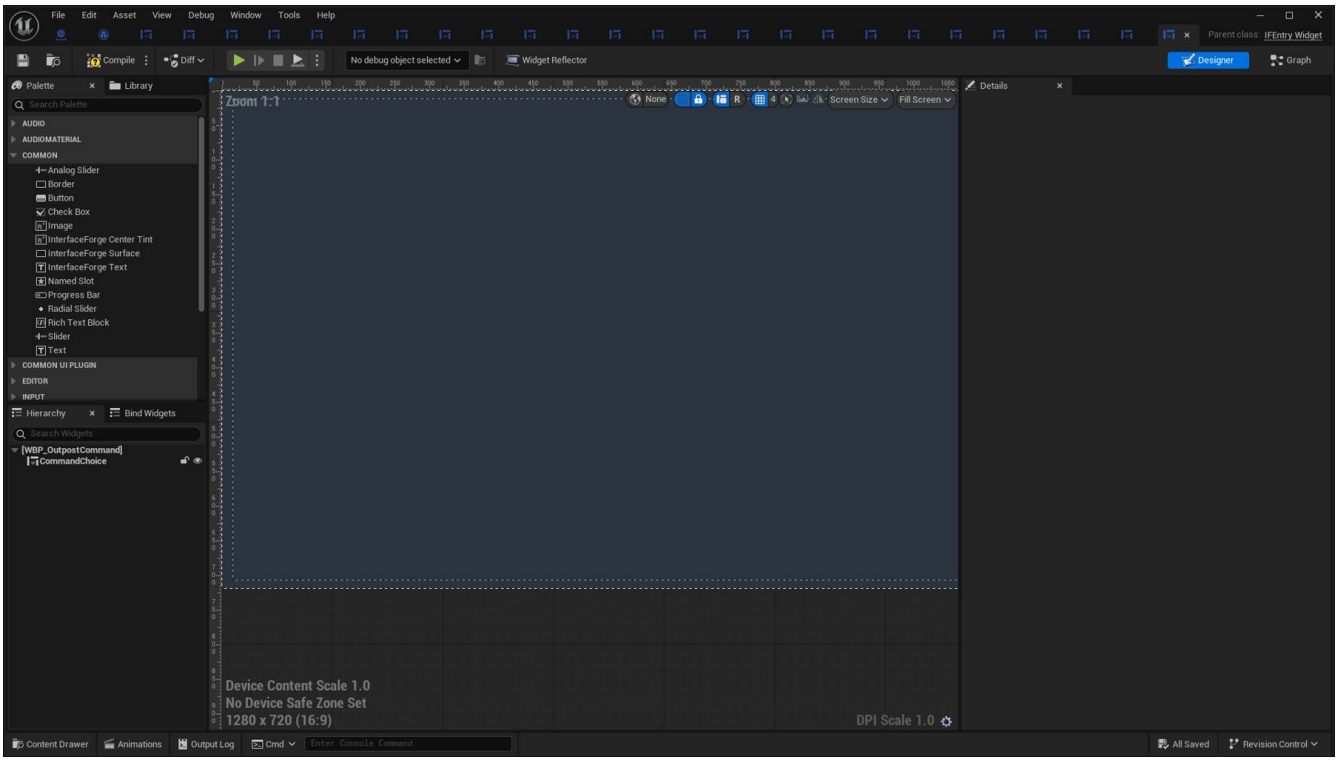
WBP_OutpostCommand

`/Game/UI/Blueprint/WBP_OutpostCommand.WBP_OutpostCommand` → `IFEntryWidget` · entry.



WBP_OutpostCommand - entry

Open full-size image - Isolated saved widget; entries use explicit sample presentation data. Layout previews do not replace the live-window images above.



WBP_OutpostCommand in Unreal Designer

Open full-size image - Actual asset editor; design-time values can differ from runtime bindings.

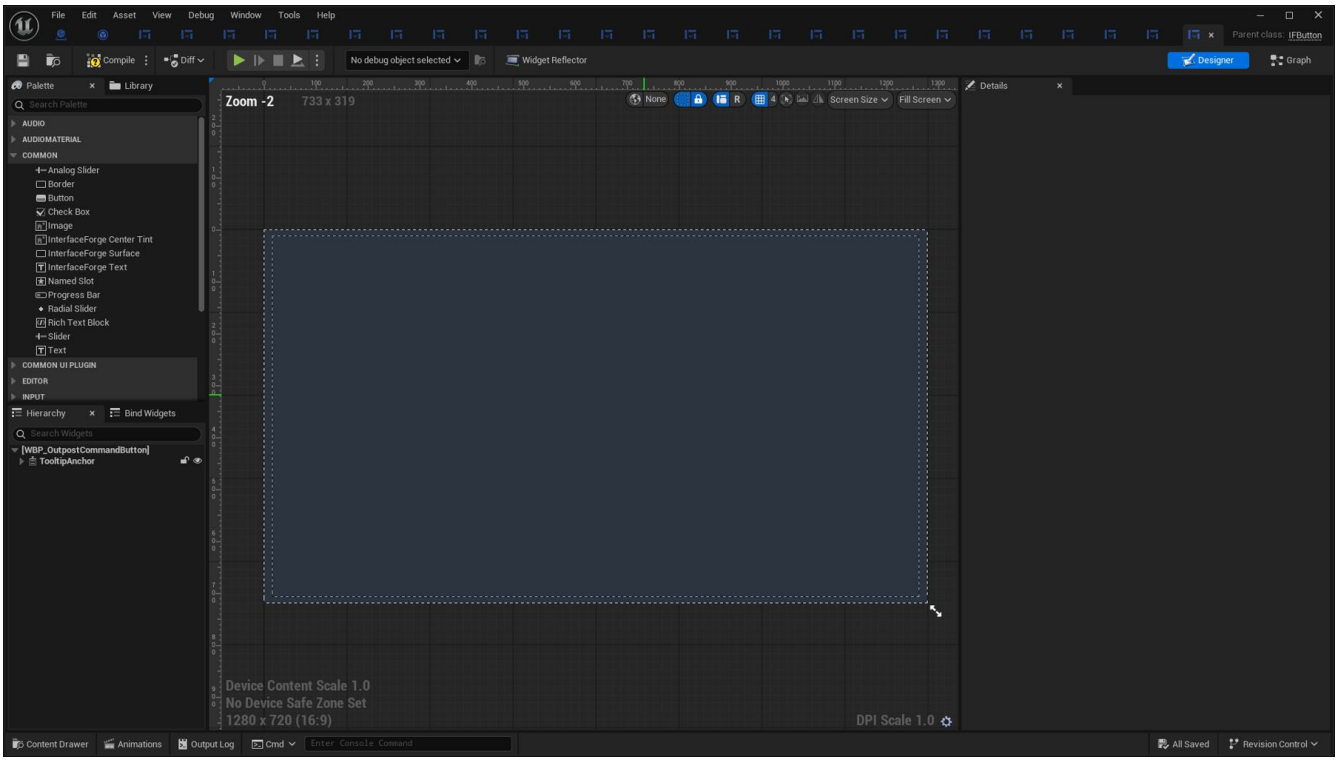
WBP_OutpostCommandButton

`/Game/UI/Blueprint/WBP_OutpostCommandButton.WBP_OutpostCommandButton` → `IFButton` · button.



WBP_OutpostCommandButton - button

Open full-size image - Isolated saved widget; entries use explicit sample presentation data. Layout previews do not replace the live-window images above.

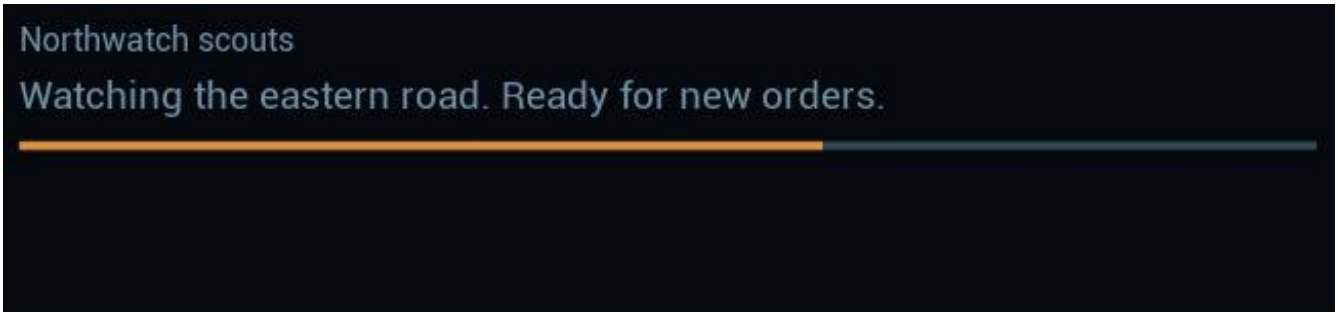


WBP_OutpostCommandButton in Unreal Designer

Open full-size image - Actual asset editor; design-time values can differ from runtime bindings.

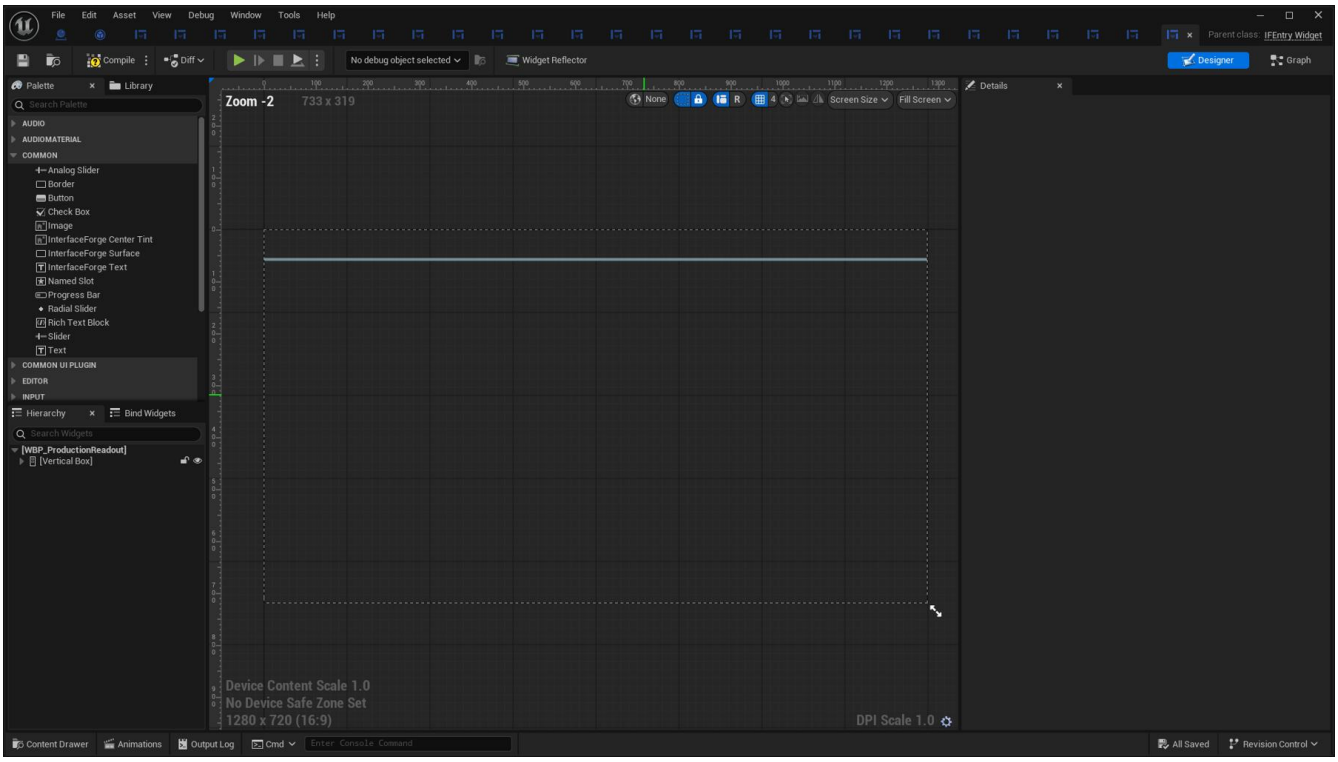
WBP_ProductionReadout

`/Game/UI/Blueprint/WBP_ProductionReadout.WBP_ProductionReadout` → `IFEntryWidget` · entry.



WBP_ProductionReadout - entry

Open full-size image - Isolated saved widget; entries use explicit sample presentation data. Layout previews do not replace the live-window images above.



WBP_ProductionReadout in Unreal Designer

Open full-size image - Actual asset editor; design-time values can differ from runtime bindings.

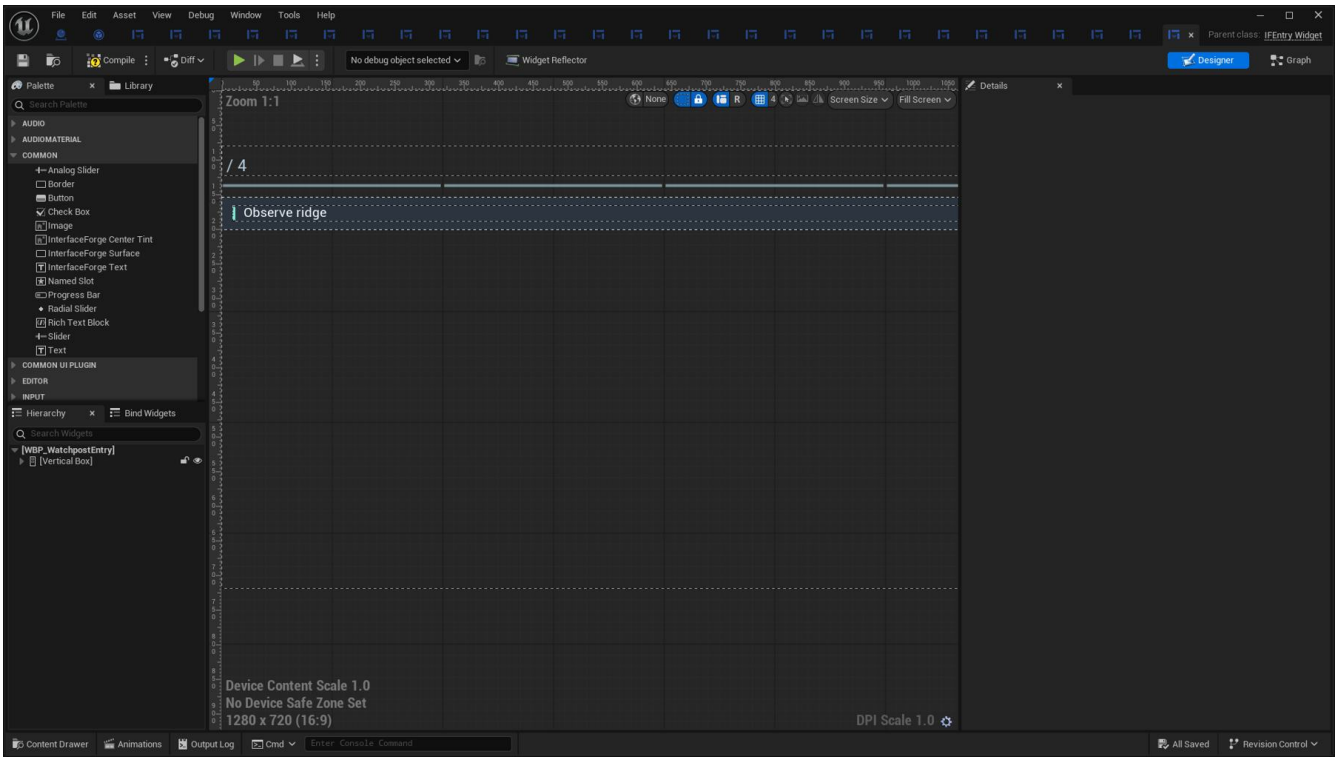
WBP_WatchpostEntry

`/Game/UI/Blueprint/WBP_WatchpostEntry.WBP_WatchpostEntry` → `IFEntryWidget` · entry.



WBP_WatchpostEntry - entry

Open full-size image - Isolated saved widget; entries use explicit sample presentation data. Layout previews do not replace the live-window images above.

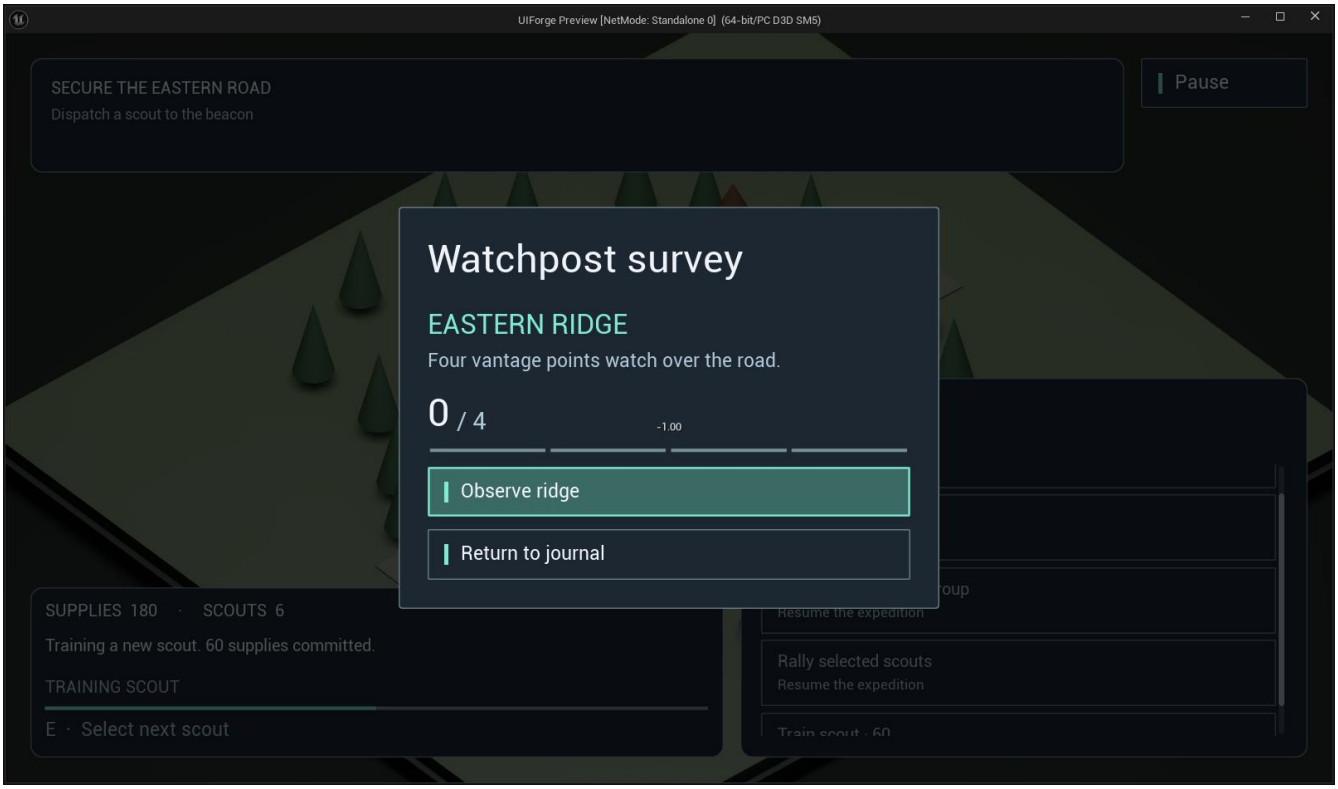


WBP_WatchpostEntry in Unreal Designer

Open full-size image - Actual asset editor; design-time values can differ from runtime bindings.

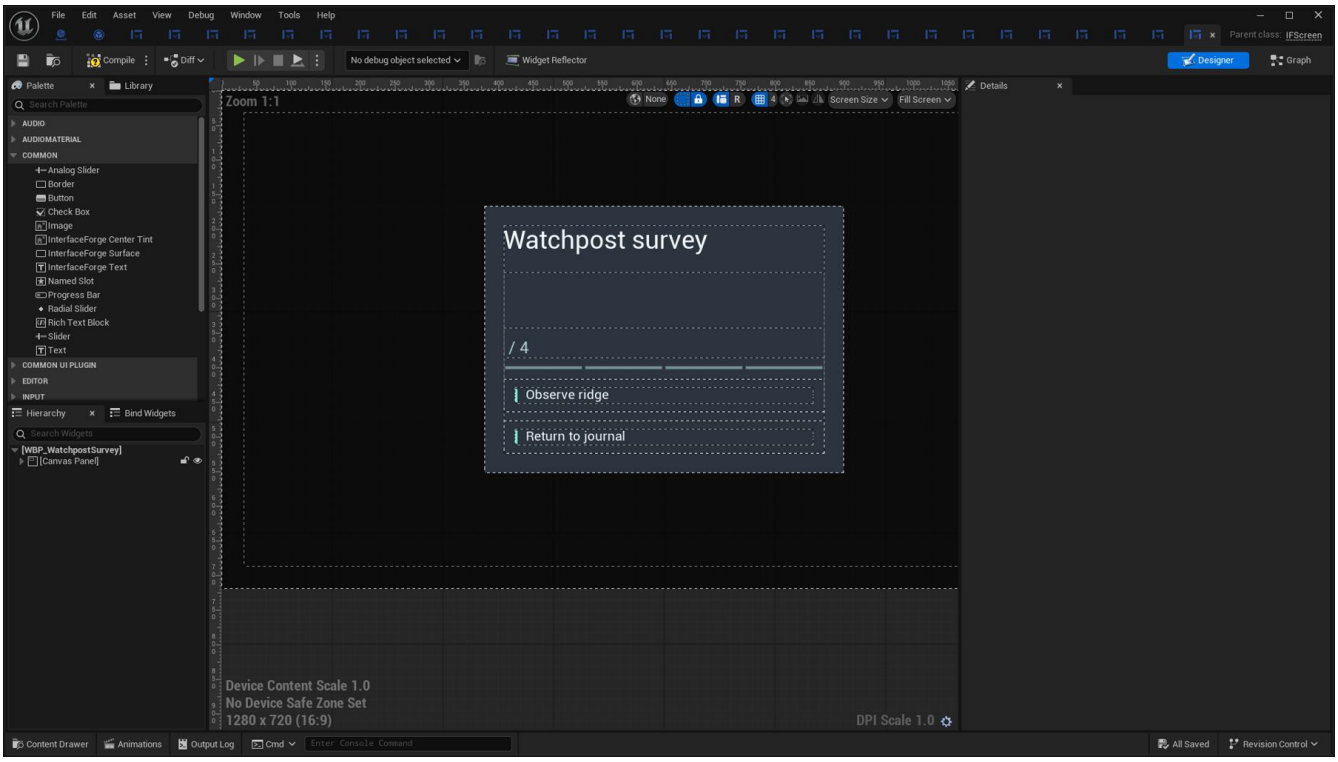
WBP_WatchpostSurvey

`/Game/UI/Blueprint/WBP_WatchpostSurvey.WBP_WatchpostSurvey` → `IFScreen` · screen.



WBP_WatchpostSurvey - screen

Open full-size image - Live composed interface.



WBP_WatchpostSurvey in Unreal Designer

Open full-size image - Actual asset editor; design-time values can differ from runtime bindings.

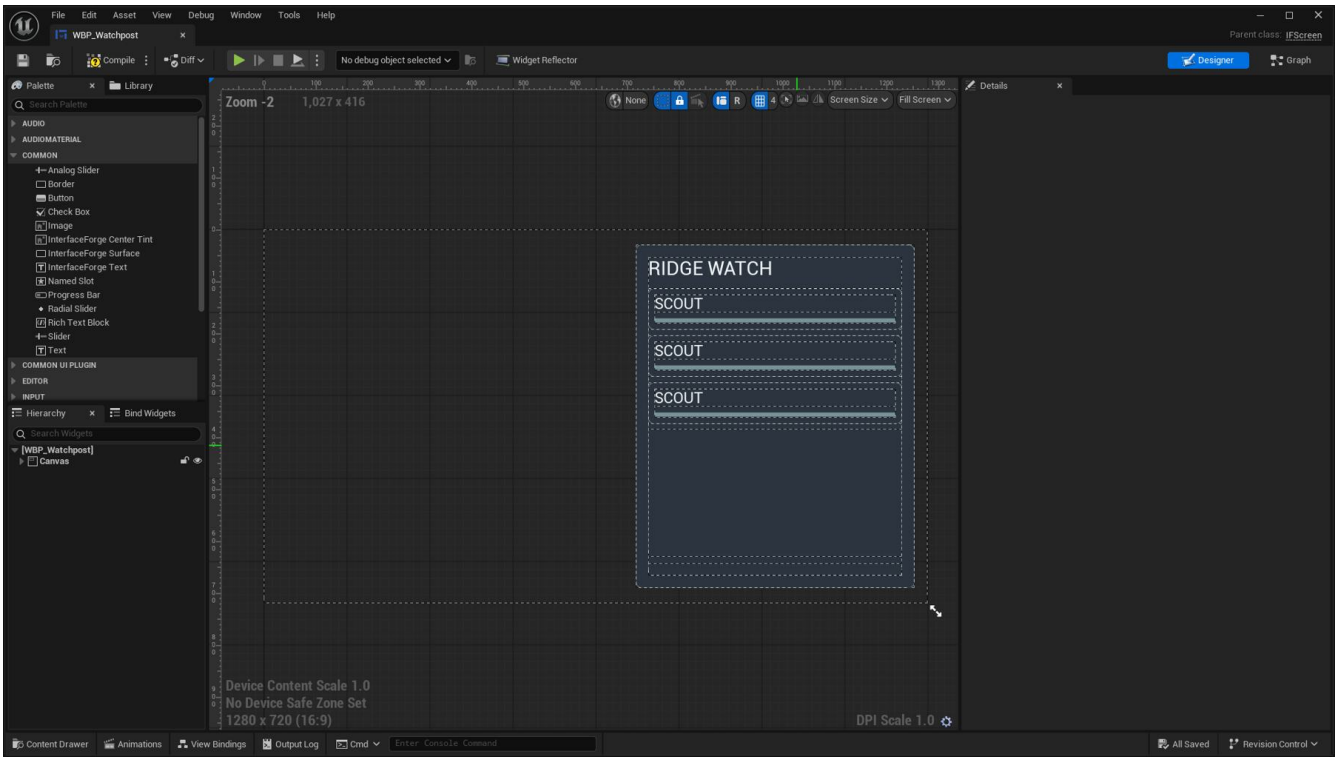
WBP_Watchpost

`/Game/UI/WBP_Watchpost.WBP_Watchpost` → `IFScreen` · screen. Original optional example UI, with explicit preview data in UE5.8.3; no MassForge backend run.



WBP_Watchpost - screen

Open full-size image - Live composed interface.

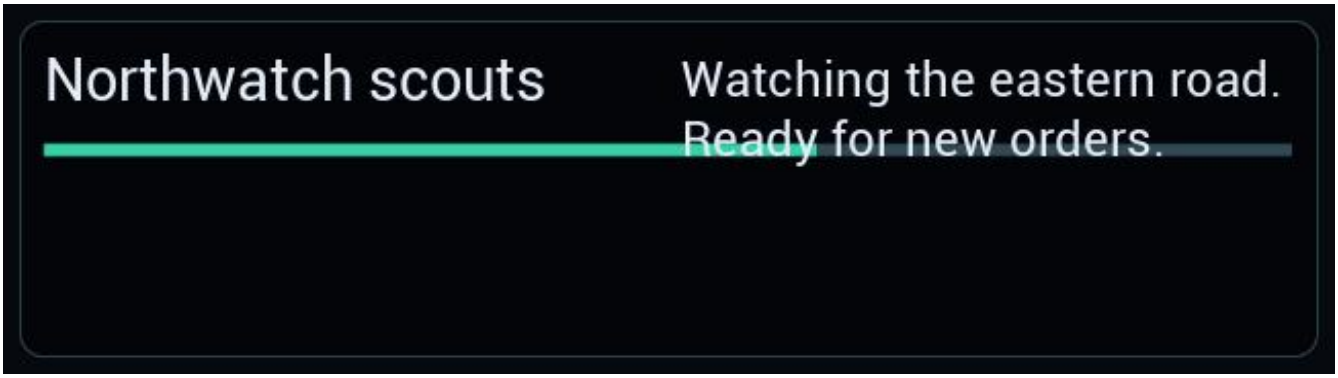


WBP_Watchpost in Unreal Designer

Open full-size image - Actual asset editor; design-time values can differ from runtime bindings.

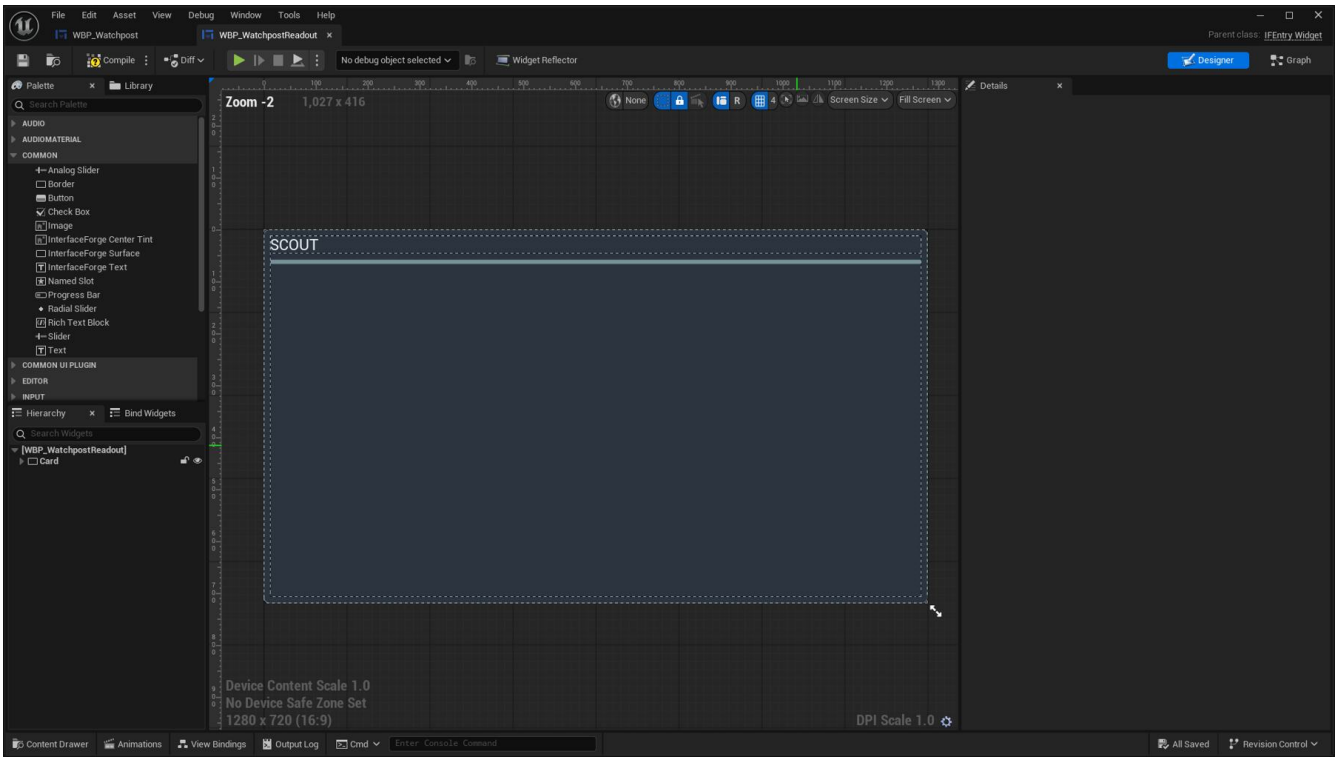
WBP_WatchpostReadout

`/Game/UI/WBP_WatchpostReadout.WBP_WatchpostReadout` → `IFEntryWidget` · entry.



WBP_WatchpostReadout - entry

Open full-size image - Isolated saved widget; entries use explicit sample presentation data. Layout previews do not replace the live-window images above.



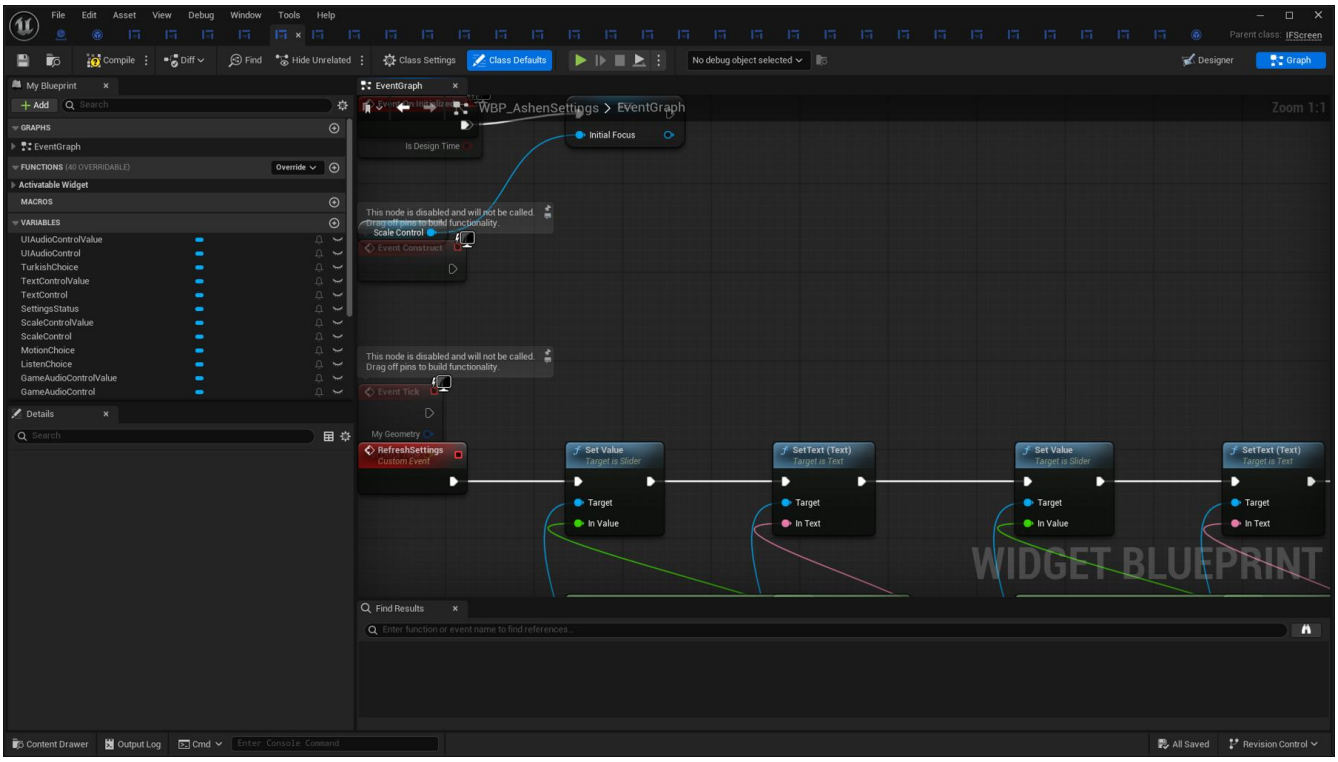
WBP_WatchpostReadout in Unreal Designer

Open full-size image - Actual asset editor; design-time values can differ from runtime bindings.

Blueprint graph wiring

Whole-graph overviews show structure. Detail images show readable node names and connections. The first document and JSON index retain every node, pin/default and wire endpoint, including graphs too large to fit one screenshot.

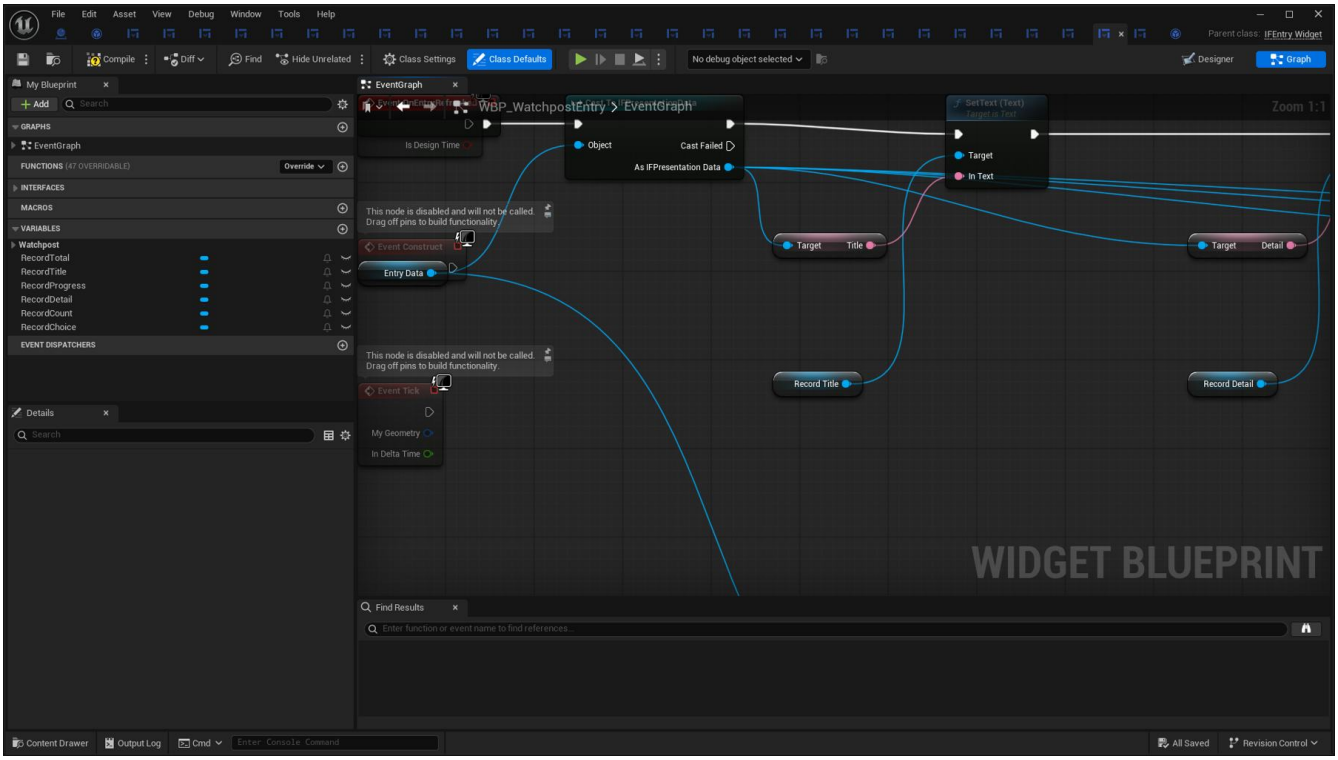
WBP_AshenSettings - overview



WBP_AshenSettings Actual Unreal Event Graph

Open full-size image

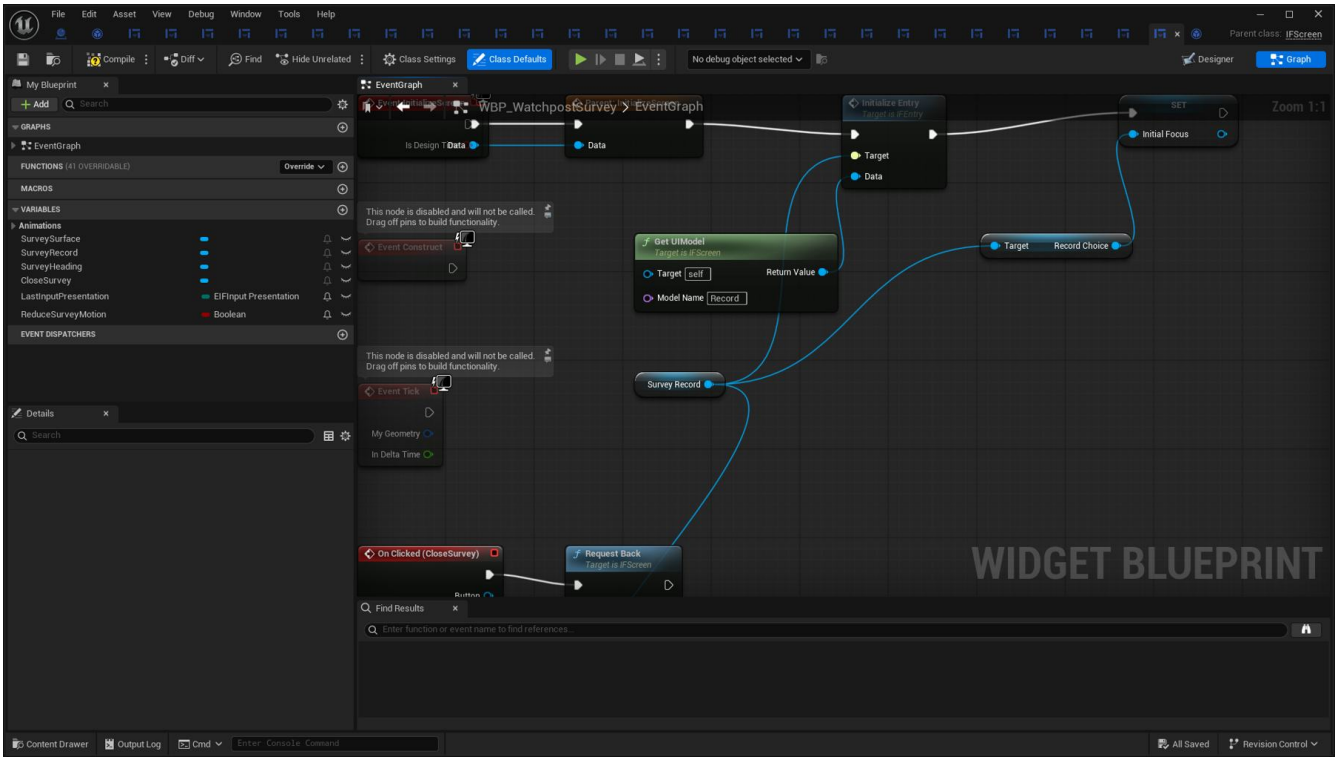
WBP_WatchpostEntry - overview



WBP_WatchpostEntry Actual Unreal Event Graph

Open full-size image

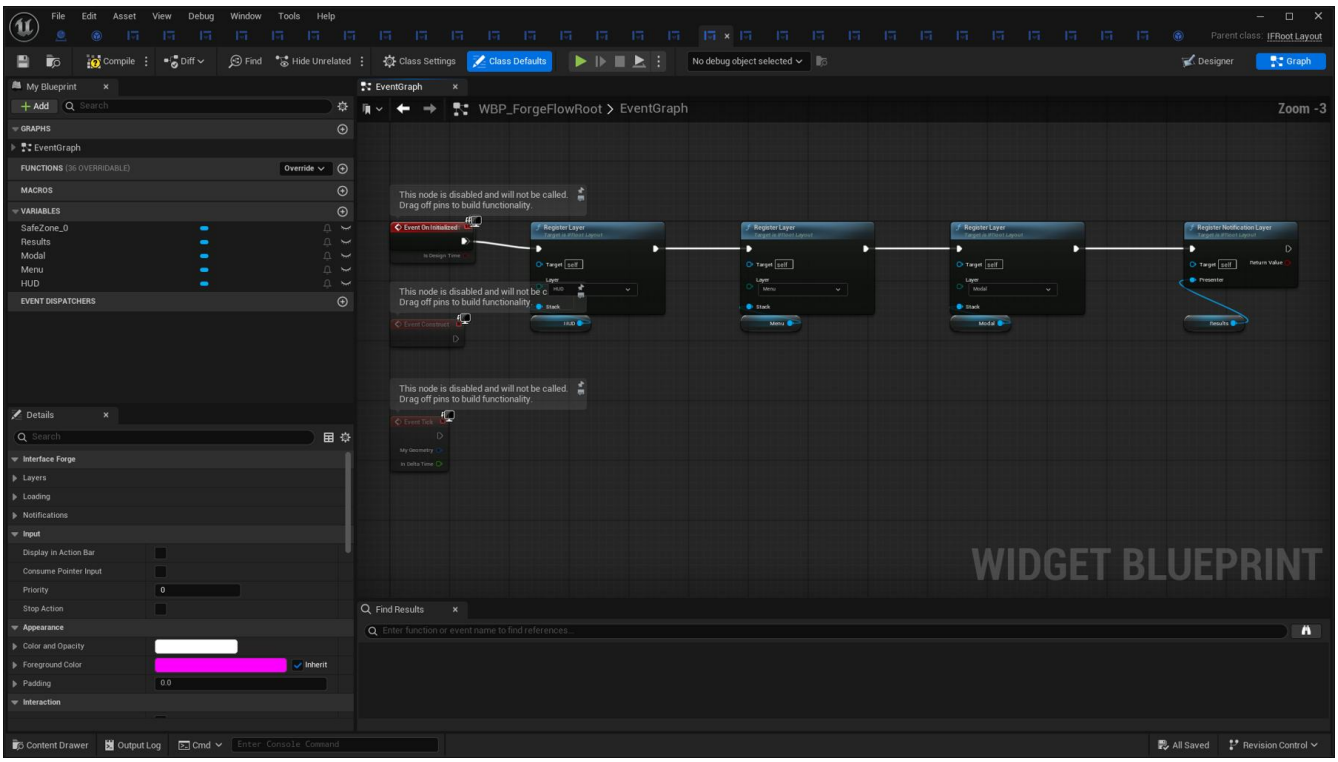
WBP_WatchpostSurvey - overview



WBP_WatchpostSurvey Actual Unreal Event Graph

Open full-size image

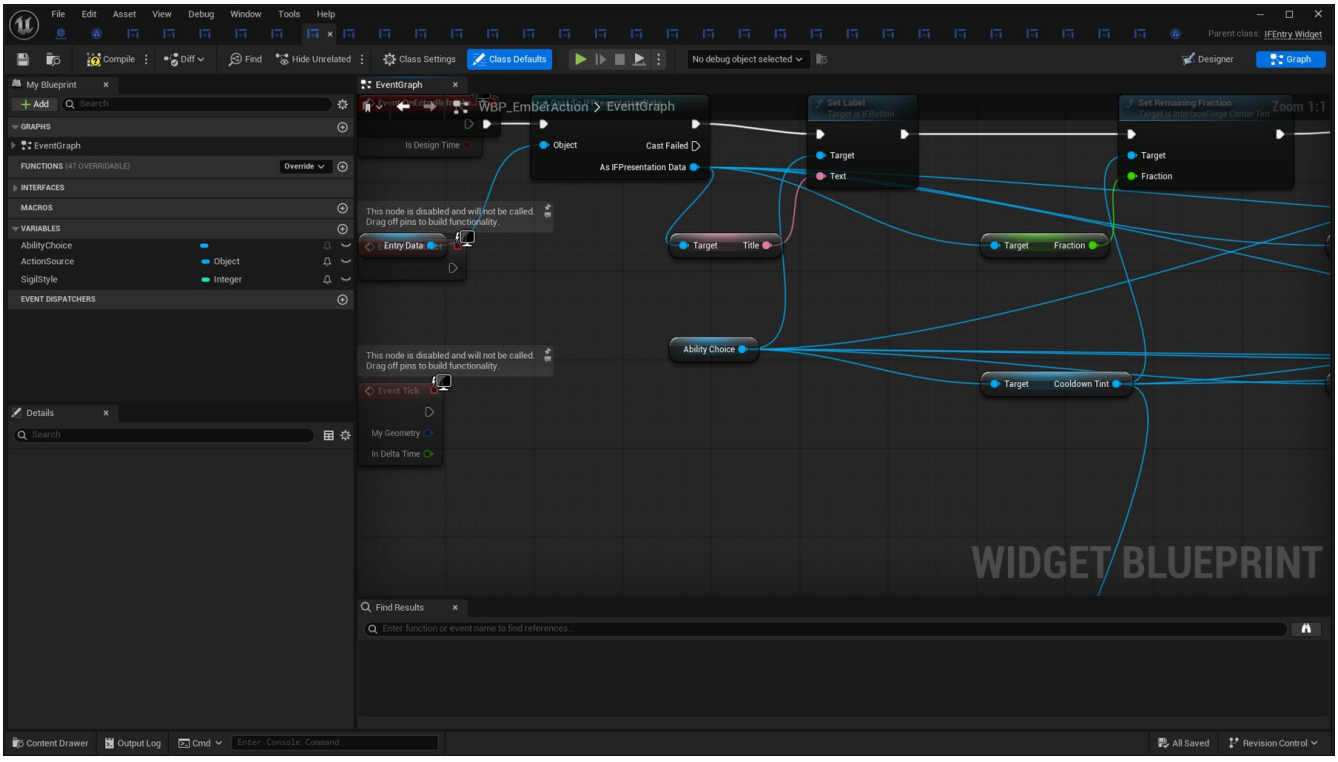
WBP_ForgeFlowRoot - overview



WBP_ForgeFlowRoot Actual Unreal Event Graph

Open full-size image

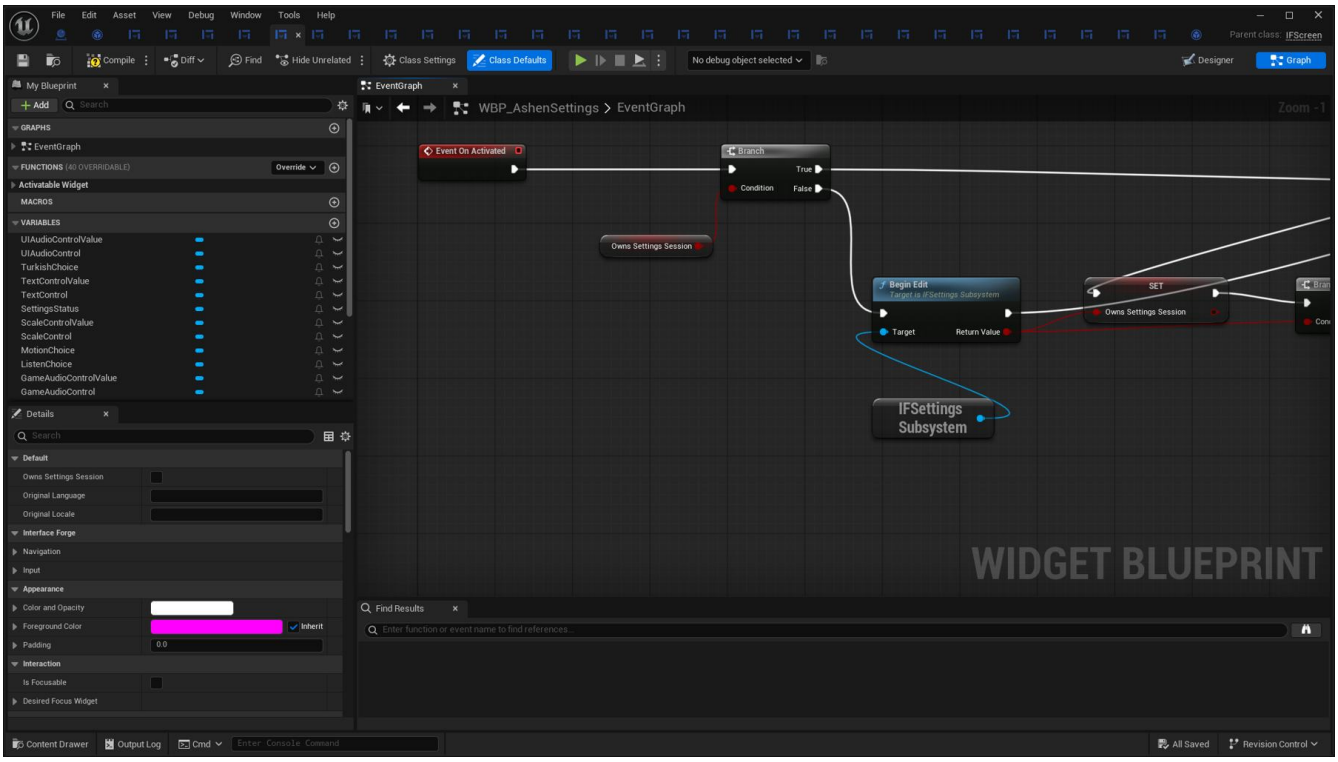
WBP_EmberAction - overview



WBP_EmberAction Actual Unreal Event Graph

Open full-size image

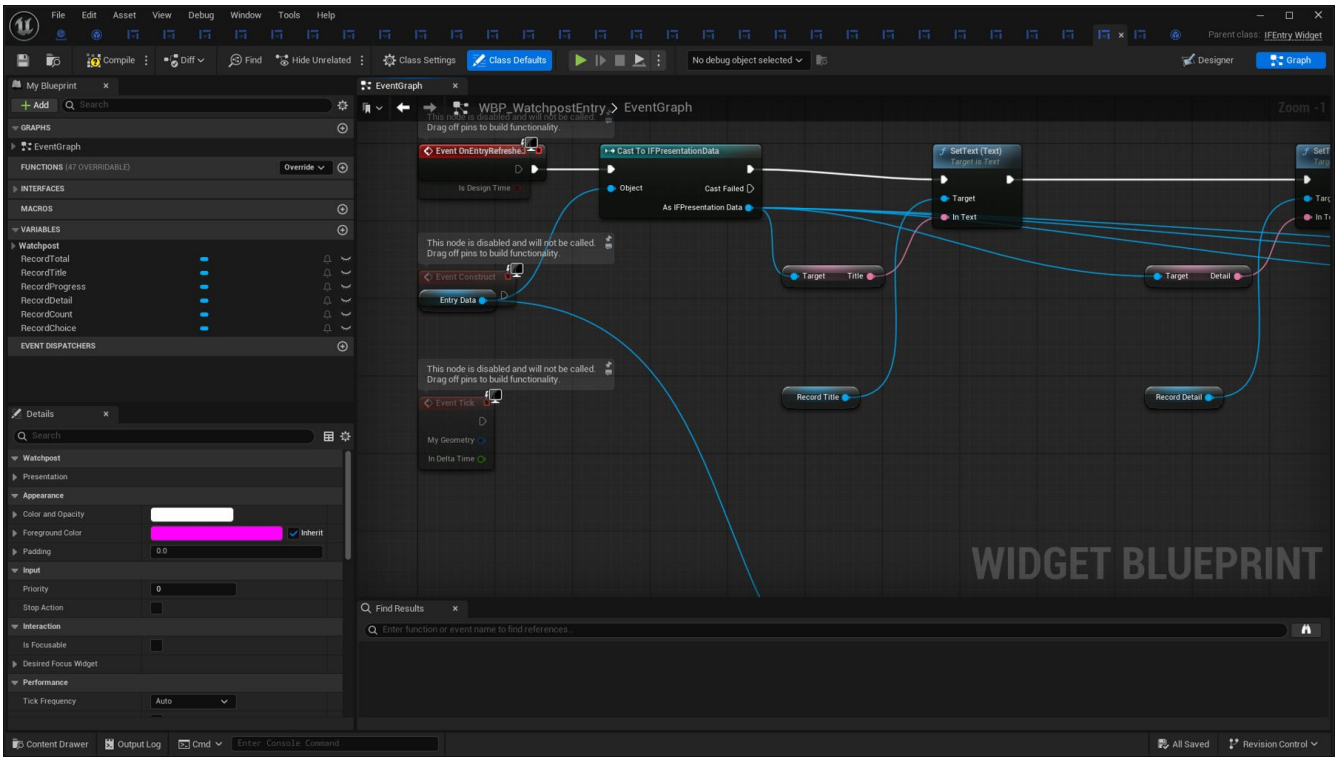
WBP_AshenSettings - detail



WBP_AshenSettings Actual Unreal graph detail

Open full-size image - Event On Activated

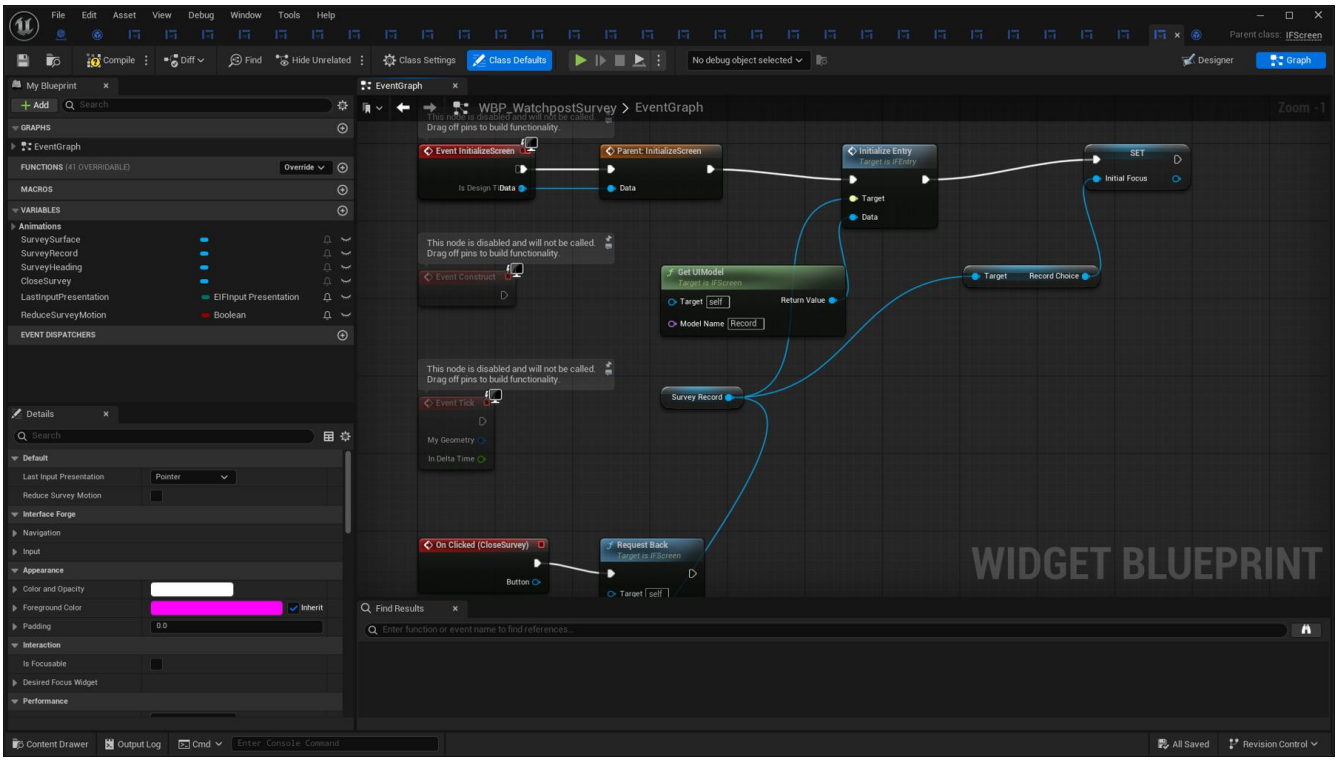
WBP_WatchpostEntry - detail



WBP_WatchpostEntry Actual Unreal graph detail

Open full-size image - Event OnEntryRefreshed

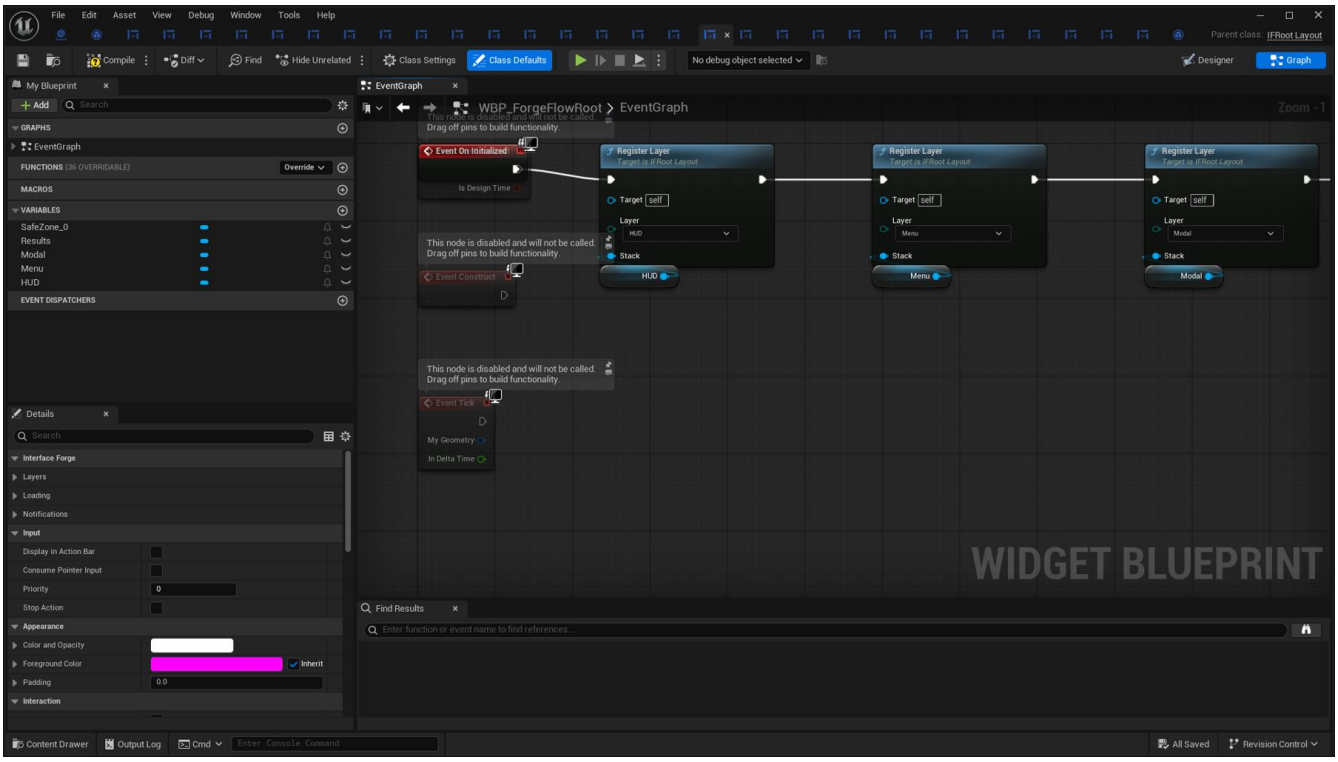
WBP_WatchpostSurvey - detail



WBP_WatchpostSurvey Actual Unreal graph detail

Open full-size image - Event InitializeScreen

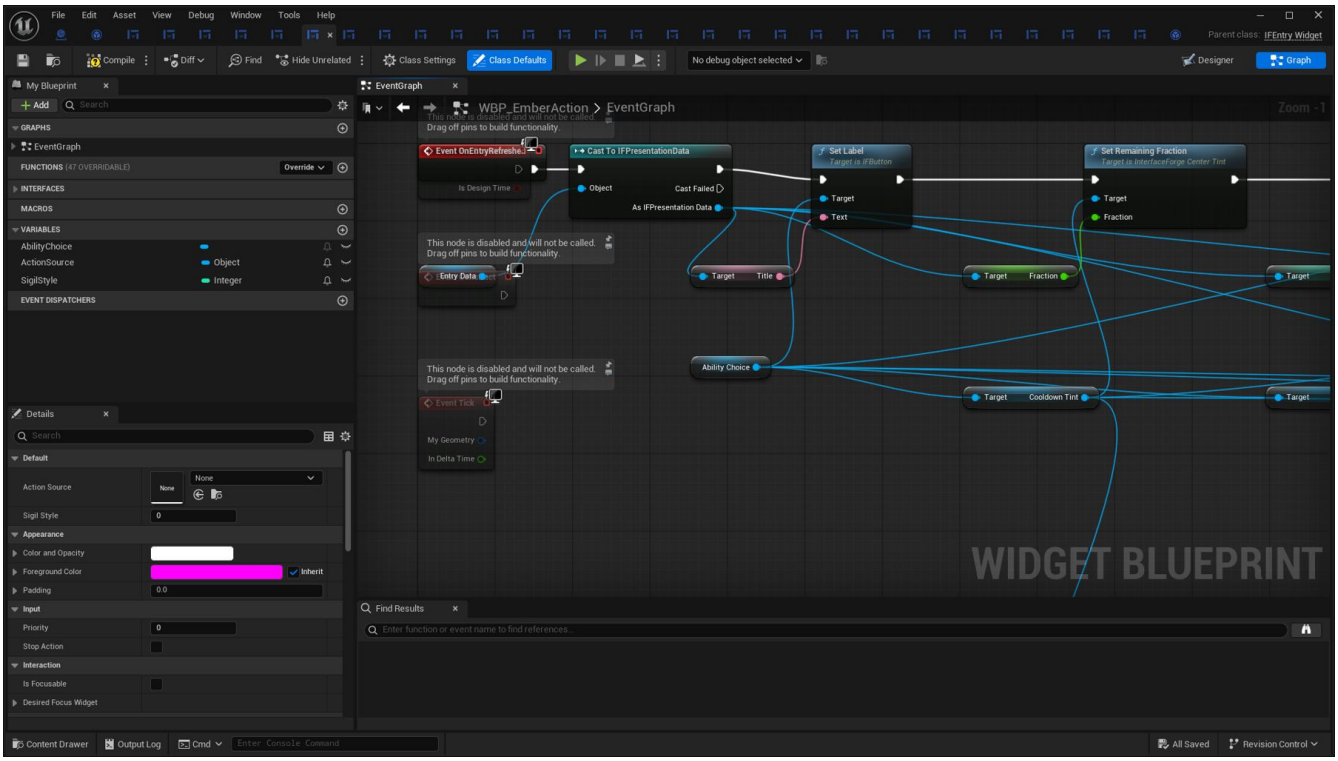
WBP_ForgeFlowRoot - detail



WBP_ForgeFlowRoot Actual Unreal graph detail

Open full-size image - Event On Initialized

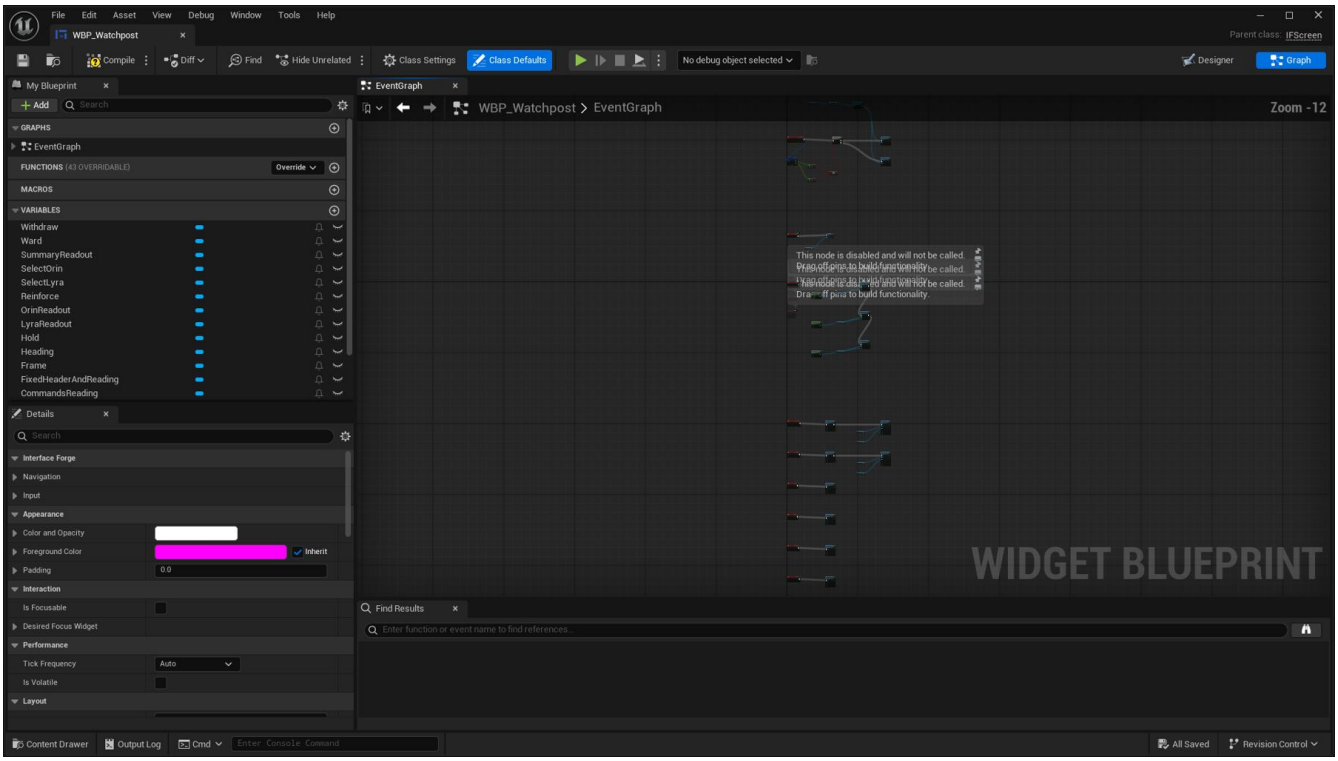
WBP_EmberAction - detail



WBP_EmberAction Actual Unreal graph detail

Open full-size image - Event OnEntryRefreshed

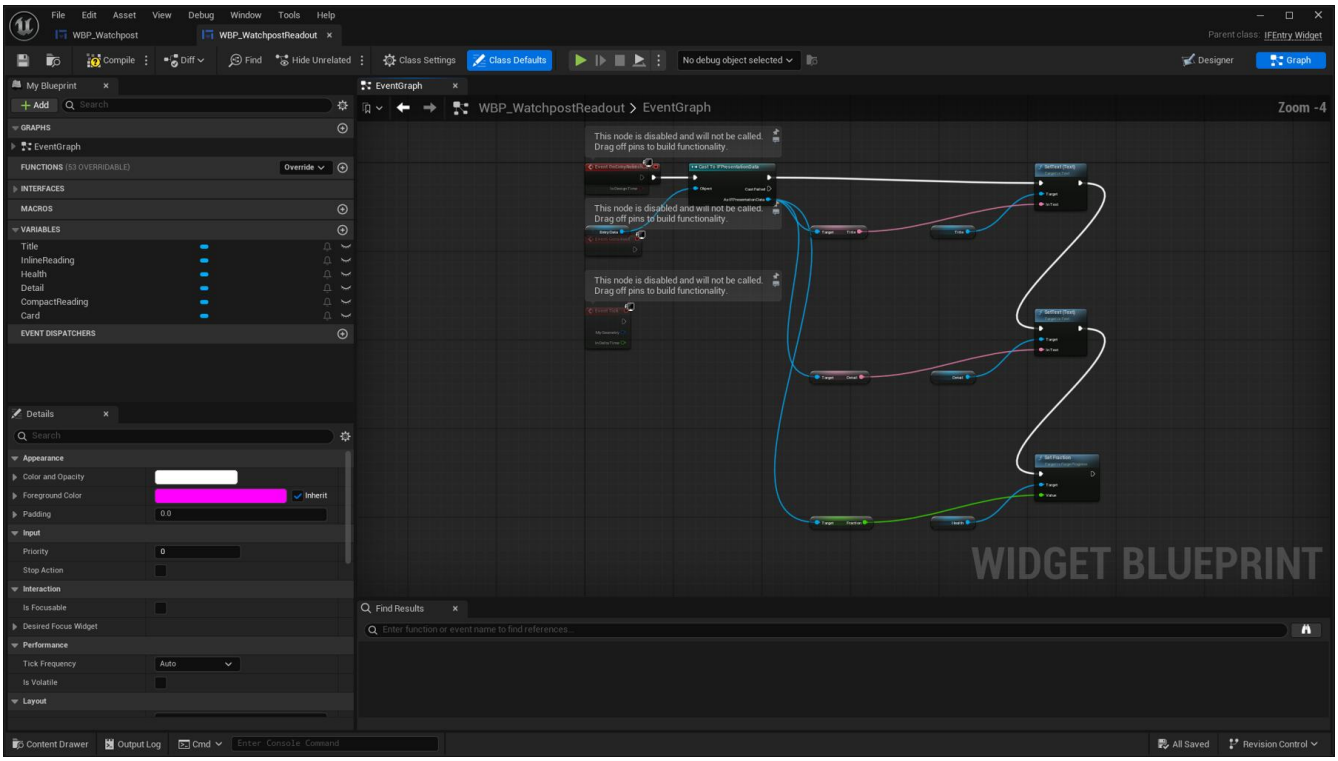
WBP_Watchpost - overview



WBP_Watchpost Actual UE5.8 Event Graph

Open full-size image

WBP_WatchpostReadout - overview



WBP_WatchpostReadout Actual UE5.8 Event Graph

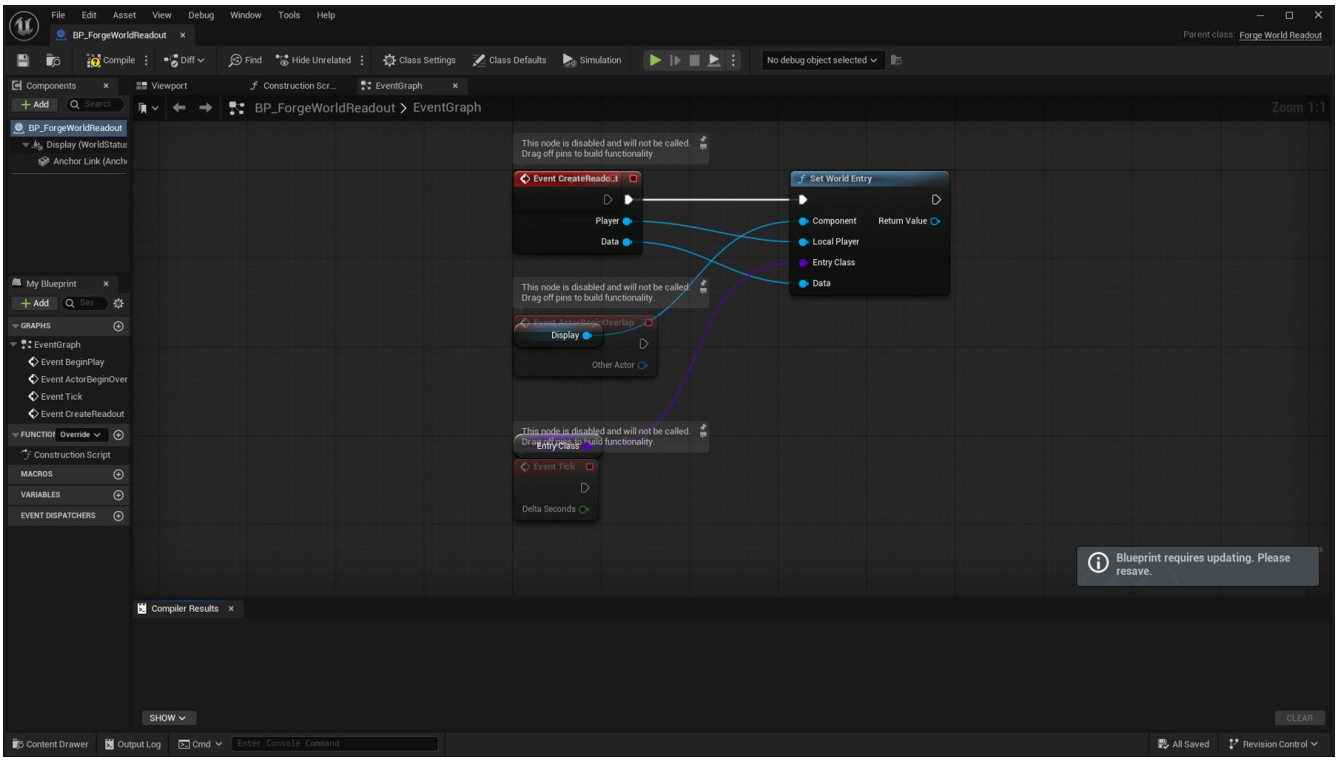
Open full-size image

Non-widget Blueprint assets

These objects are not standalone UI windows. Their editor images show the actor/provider/controller configuration used by the interfaces.

BP_ForgeWorldReadout

Parent: `/Script/ForgeExamples.ForgeWorldReadout`.

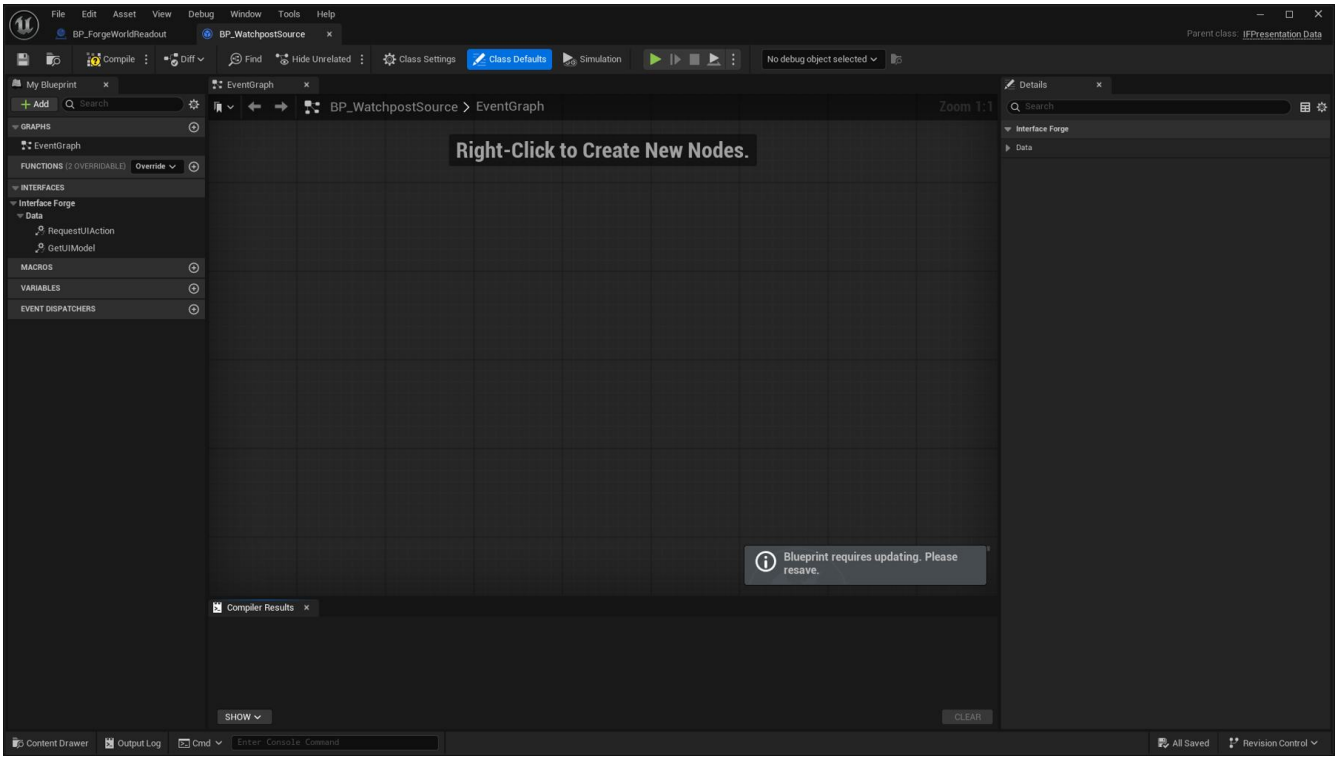


BP_ForgeWorldReadout actual Blueprint editor

Open full-size image

BP_WatchpostSource

Parent: `/Script/InterfaceForge.IFPresentationData`.

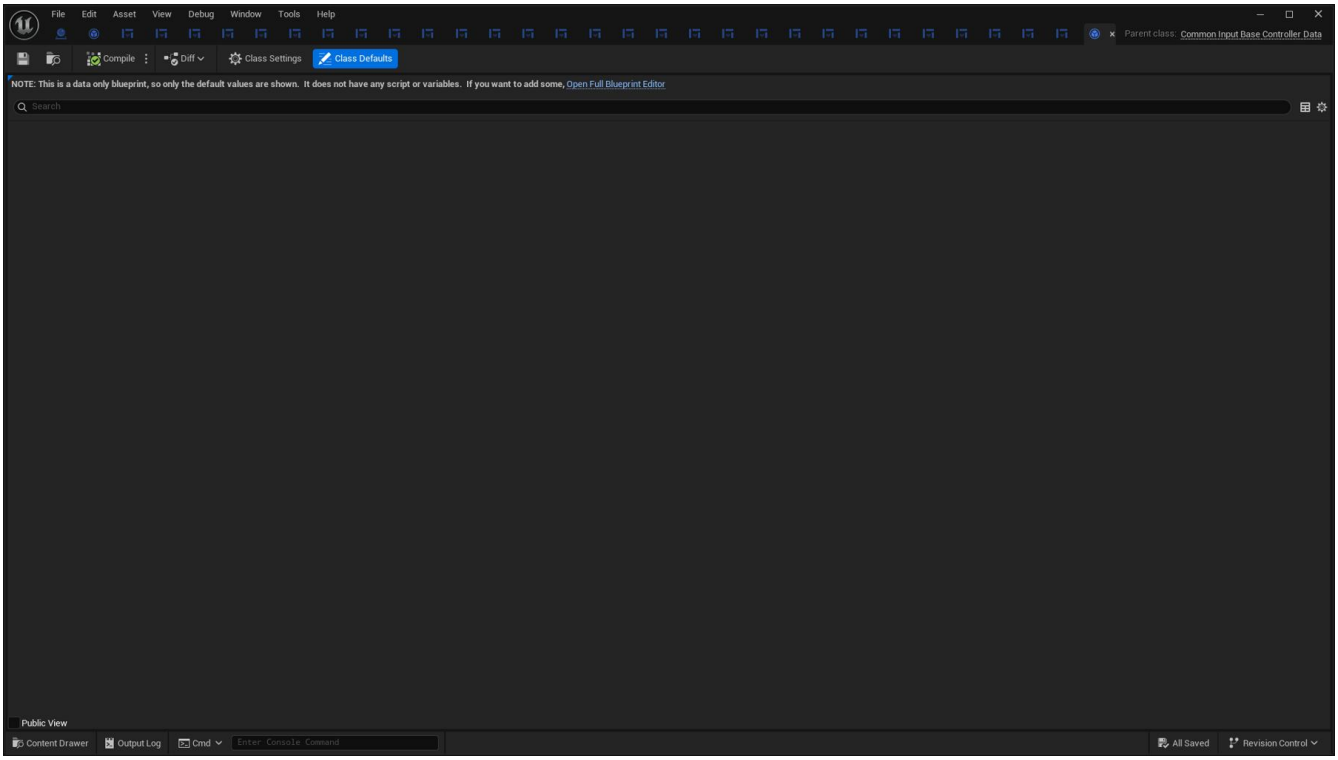


BP_WatchpostSource actual Blueprint editor

Open full-size image

BP_IFGenericController

Parent: `/Script/CommonInput.CommonInputBaseControllerData`.



BP_IFGenericController actual Blueprint editor

Open full-size image

Native widgets and fallback bodies

Native fallbacks are included so the catalog does not hide code-created UI. Isolated dialogs/loading widgets receive typed public payloads. Surface/scroll examples contain explicit sample text to make otherwise empty containers visible.

IFButton



/Script/InterfaceForge.IFButton native visual preview

Open full-size image - Native visual body, independent of the Designer Blueprint replacement.

IFTooltipCard



/Script/InterfaceForge.IFTooltipCard native visual preview

Open full-size image - Native visual body, independent of the Designer Blueprint replacement.

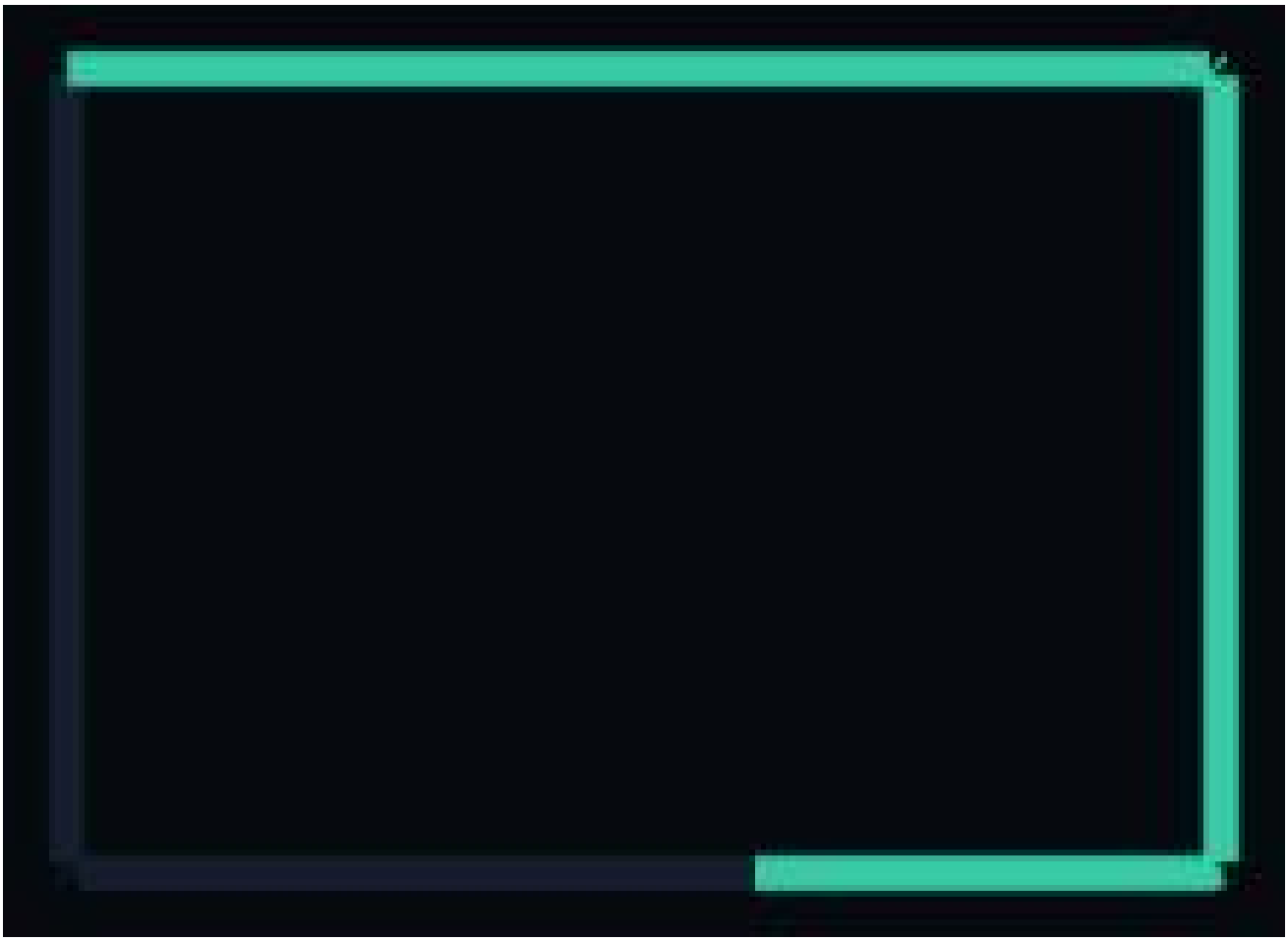
IFText



/Script/InterfaceForge.IFText native visual preview

Open full-size image - Native visual body, independent of the Designer Blueprint replacement.

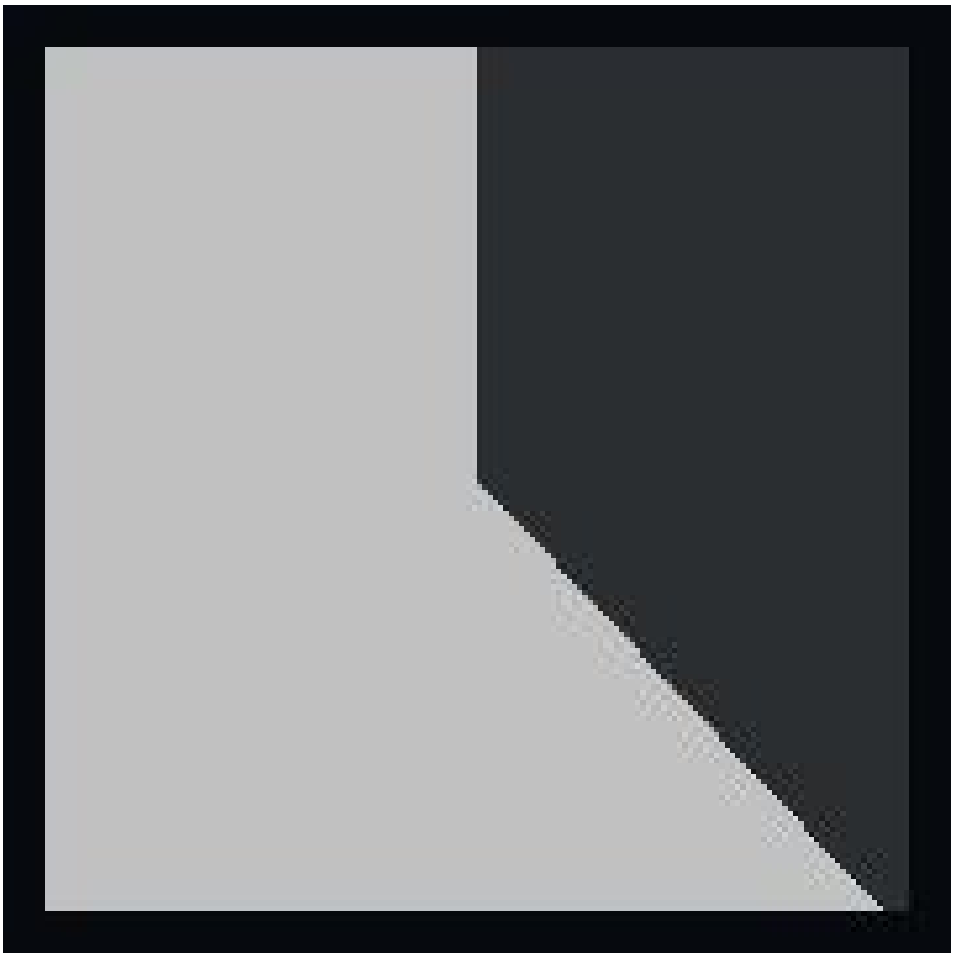
IFProgressWidget



/Script/InterfaceForge.IFProgressWidget native visual preview

Open full-size image - Native visual body, independent of the Designer Blueprint replacement.

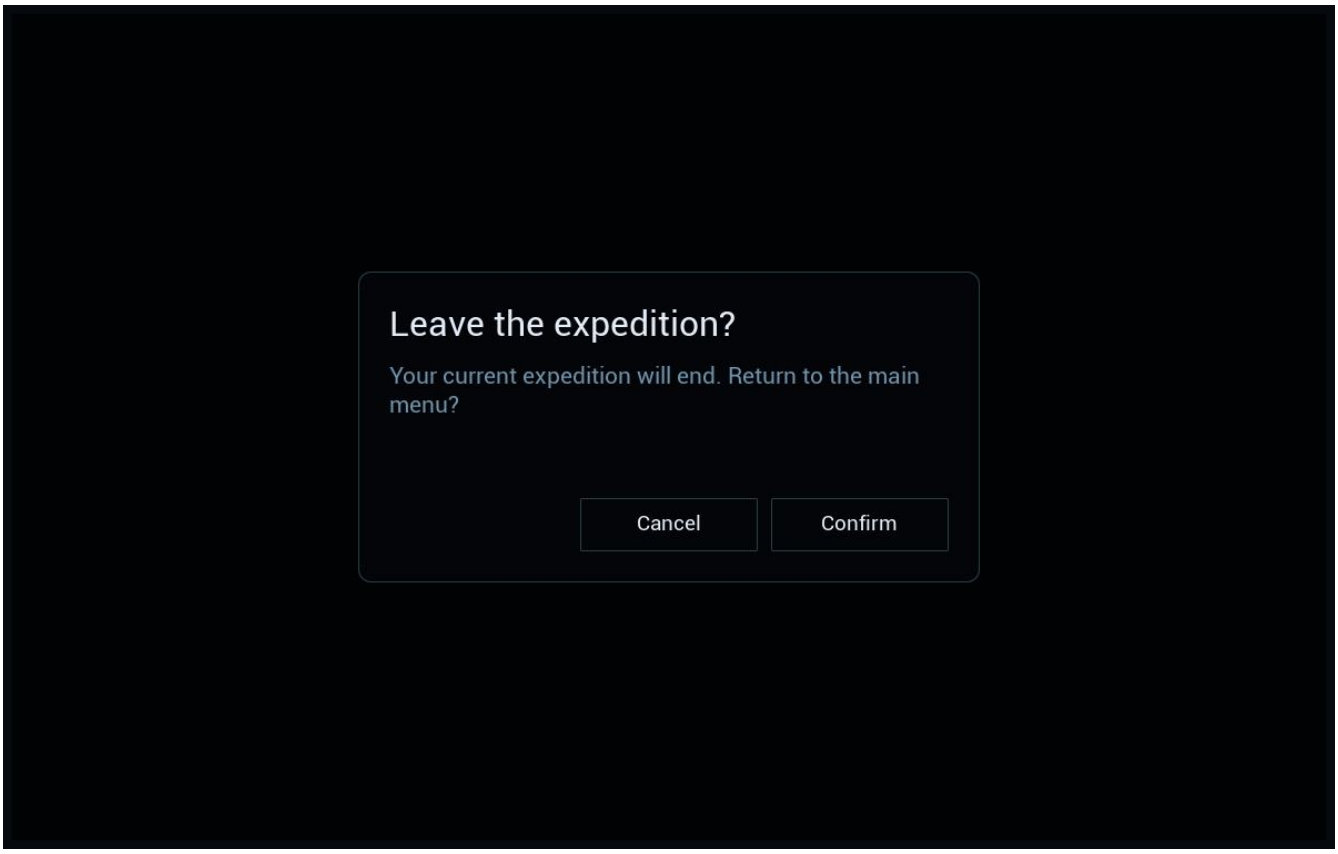
IFCenterTint



/Script/InterfaceForge.IFCenterTint native visual preview

Open full-size image - Native visual body, independent of the Designer Blueprint replacement.

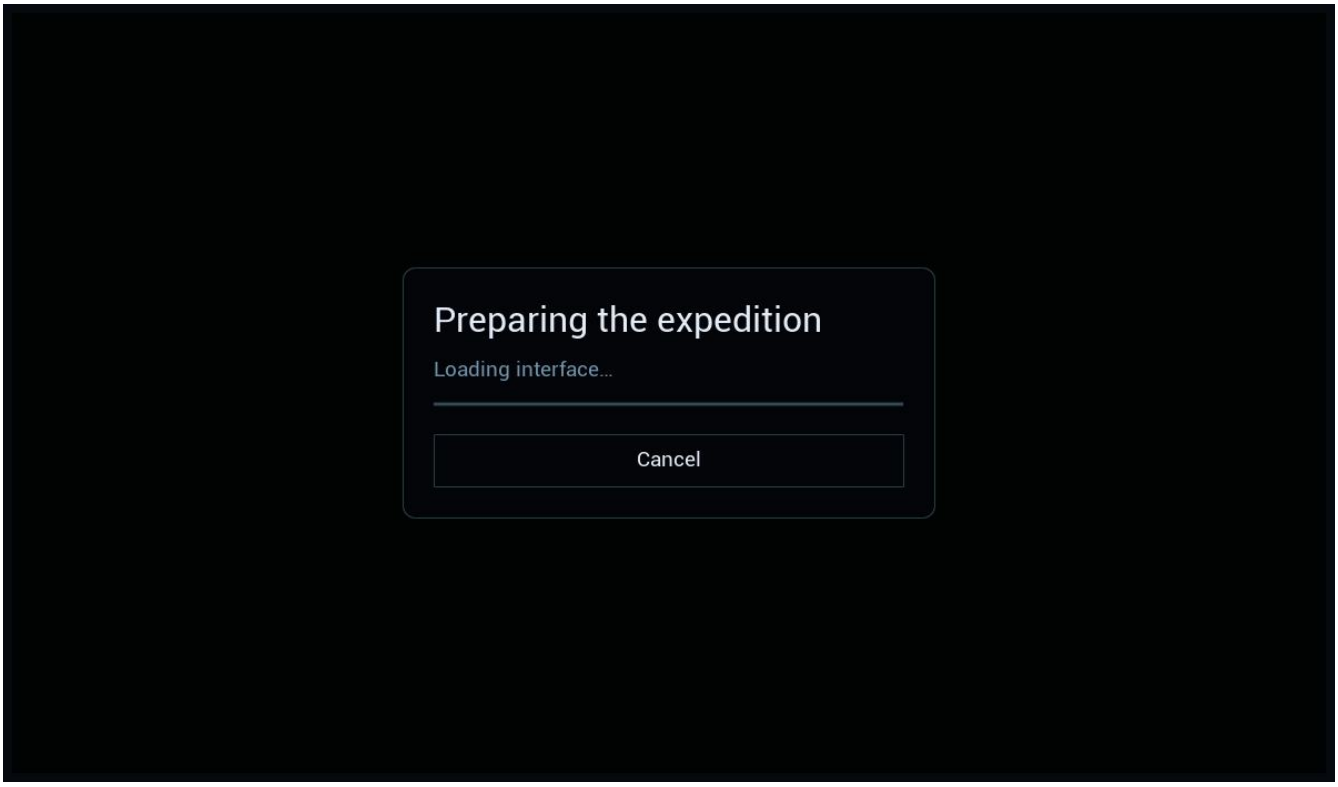
IFDialog



/Script/InterfaceForge.IFDialog native visual preview

Open full-size image - Typed public presentation/dialog/loading payload; no gameplay backend is implied.

IFLoadingScreen



/Script/InterfaceForge.IFLoadingScreen native visual preview

Open full-size image - Typed public presentation/dialog/loading payload; no gameplay backend is implied.

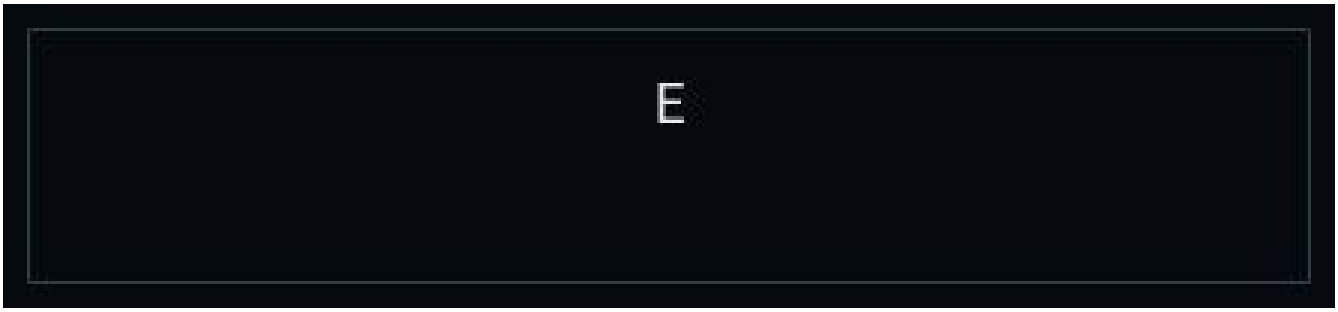
IFActionPrompt



/Script/InterfaceForge.IFActionPrompt native visual preview

Open full-size image - Typed public presentation/dialog/loading payload; no gameplay backend is implied.

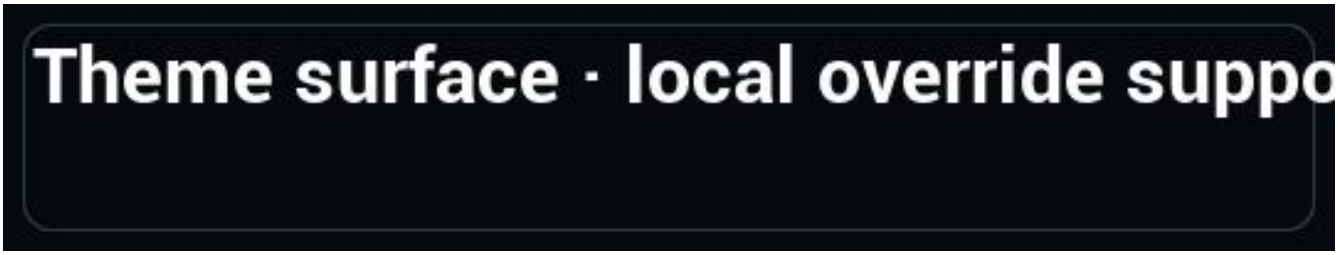
IFKeySelector



/Script/InterfaceForge.IFKeySelector native visual preview

Open full-size image - Typed public presentation/dialog/loading payload; no gameplay backend is implied.

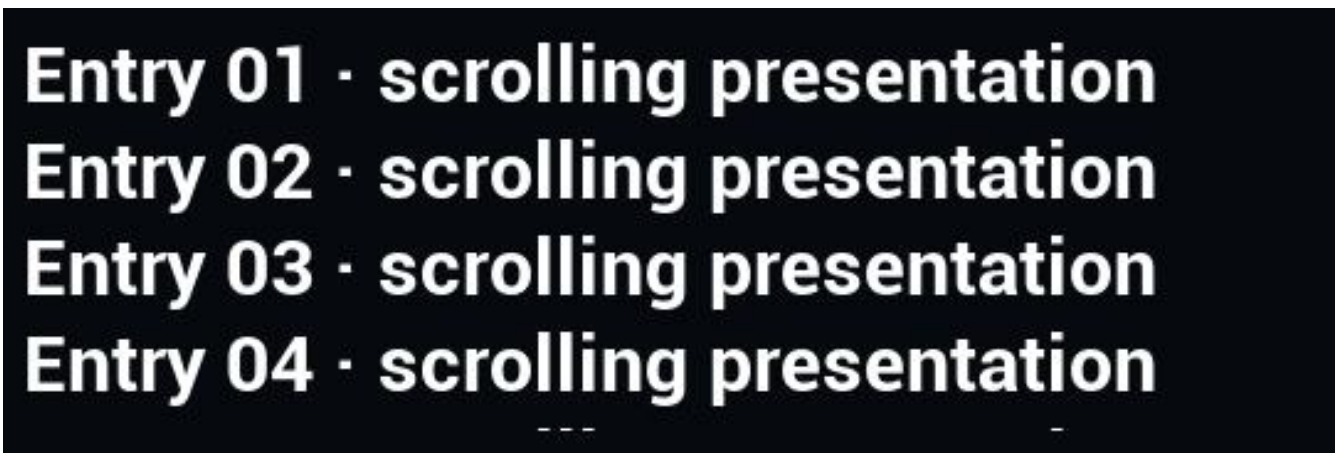
IFSurface



/Script/InterfaceForge.IFSurface native visual preview

Open full-size image - Typed public presentation/dialog/loading payload; no gameplay backend is implied.

IFScrollBox



/Script/InterfaceForge.IFScrollBox native visual preview

Open full-size image - Typed public presentation/dialog/loading payload; no gameplay backend is implied.

IFNotificationLayer



/Script/InterfaceForge.IFNotificationLayer native visual preview

Open full-size image - Typed public presentation/dialog/loading payload; no gameplay backend is implied.

Containers and contracts

Type	Visual representation
IFRootLayout	Root compositions above; owns screen layers rather than a standalone panel.
IFScreen	Base activatable contract; concrete screen images are above.
IFEntryWidget / IFEntry	Abstract entry lifecycle; each concrete entry is pictured above.
IFTooltipAnchor	Popup/focus routing container; its visible card is pictured as IFTooltipCard and WBP_ForgeTooltip.

Progress and cooldown material variants

Linear; fraction 0.62



Linear; fraction 0.62

Open full-size image - /Script/InterfaceForge.IFProgressWidget

Radial; fraction 0.62



Radial; fraction 0.62

Open full-size image - /Script/InterfaceForge.IFProgressWidget

SquarePerimeter; fraction 0.62



SquarePerimeter; fraction 0.62

Open full-size image - /Script/InterfaceForge.IFProgressWidget

Segmented; fraction 0.62



Segmented; fraction 0.62

Open full-size image - /Script/InterfaceForge.IFProgressWidget

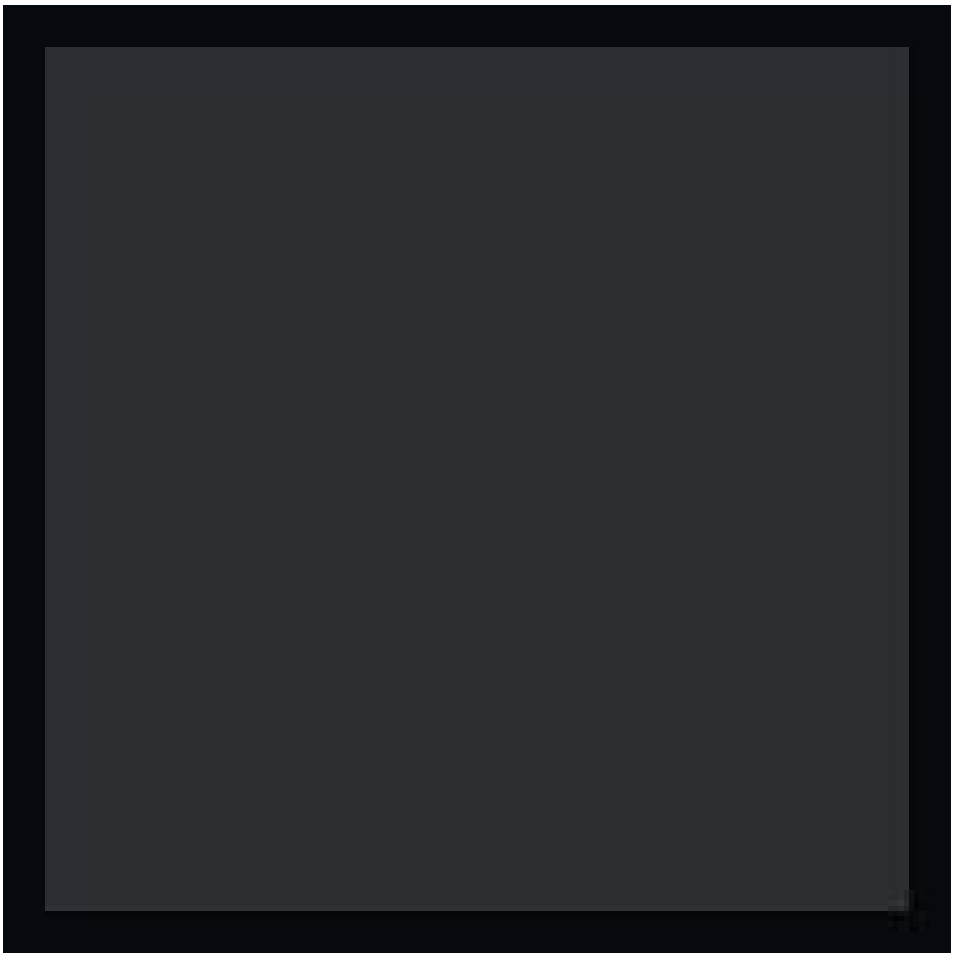
Charges; fraction 0.62



Charges; fraction 0.62

Open full-size image - /Script/InterfaceForge.IFProgressWidget

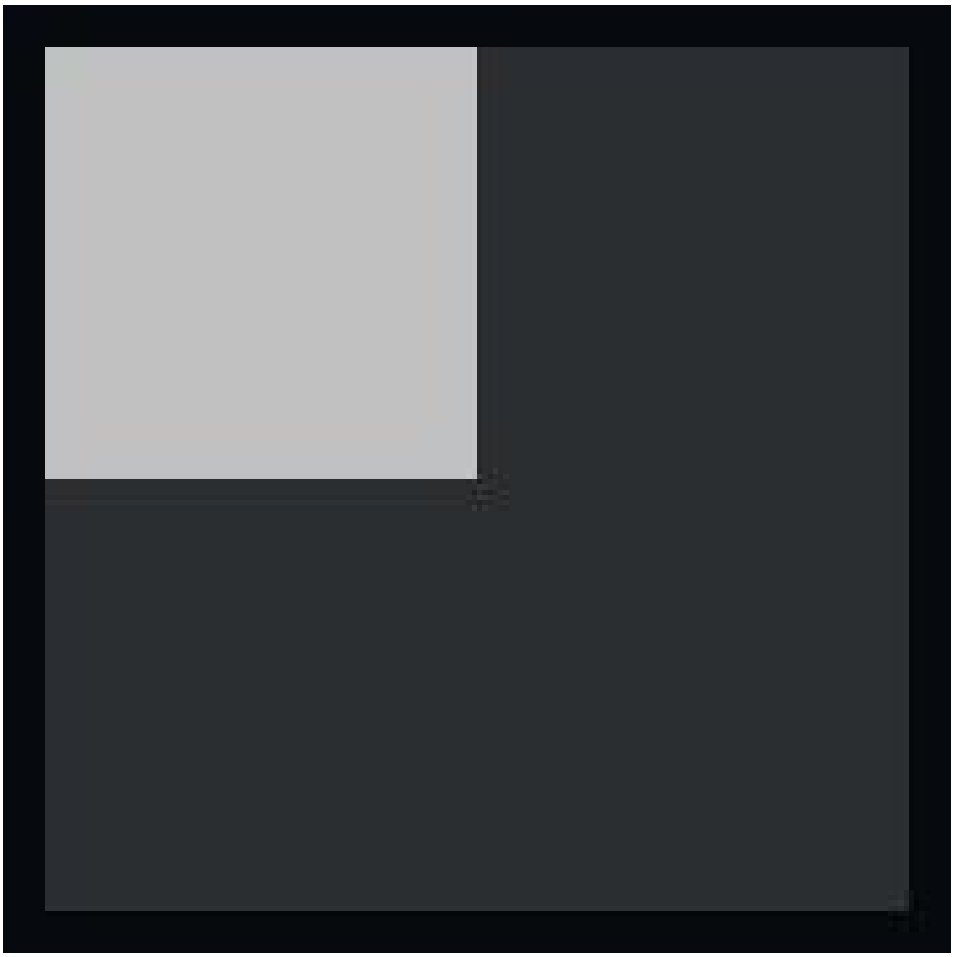
Remaining fraction 0



Remaining fraction 0

Open full-size image - /Script/InterfaceForge.IFCenterTint

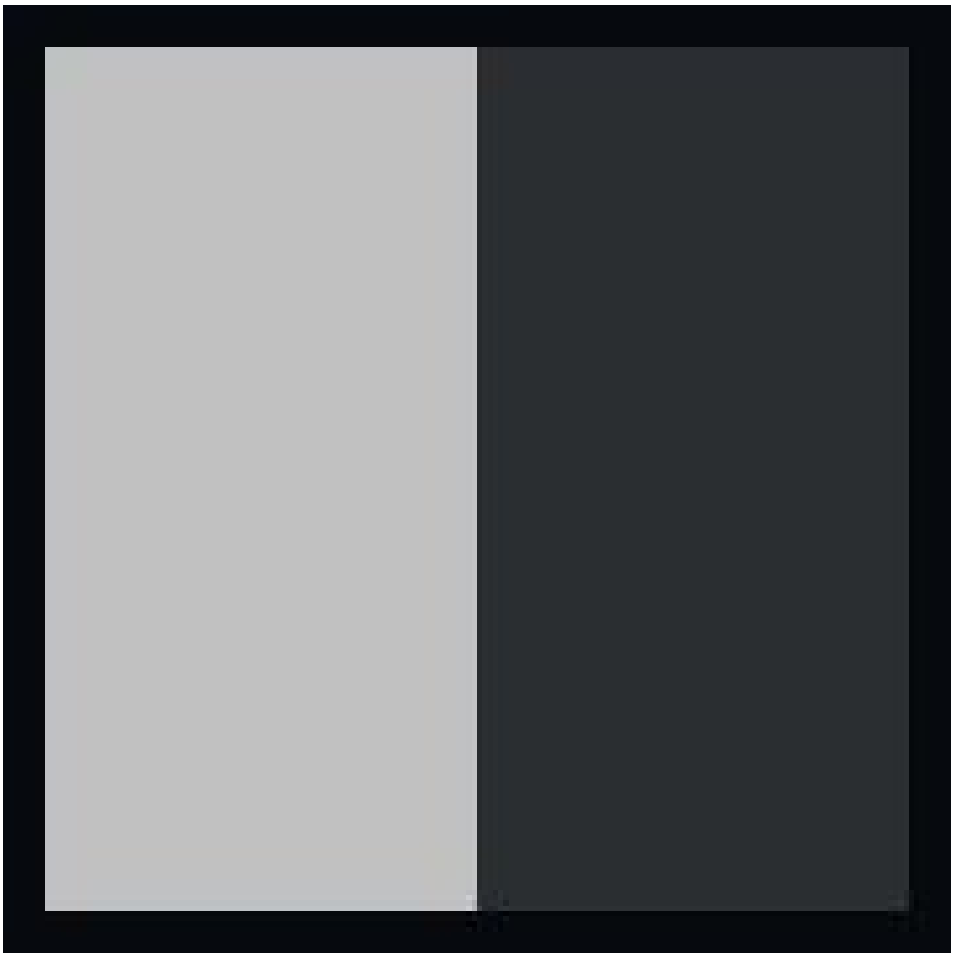
Remaining fraction 0.25



Remaining fraction 0.25

Open full-size image - /Script/InterfaceForge.IFCenterTint

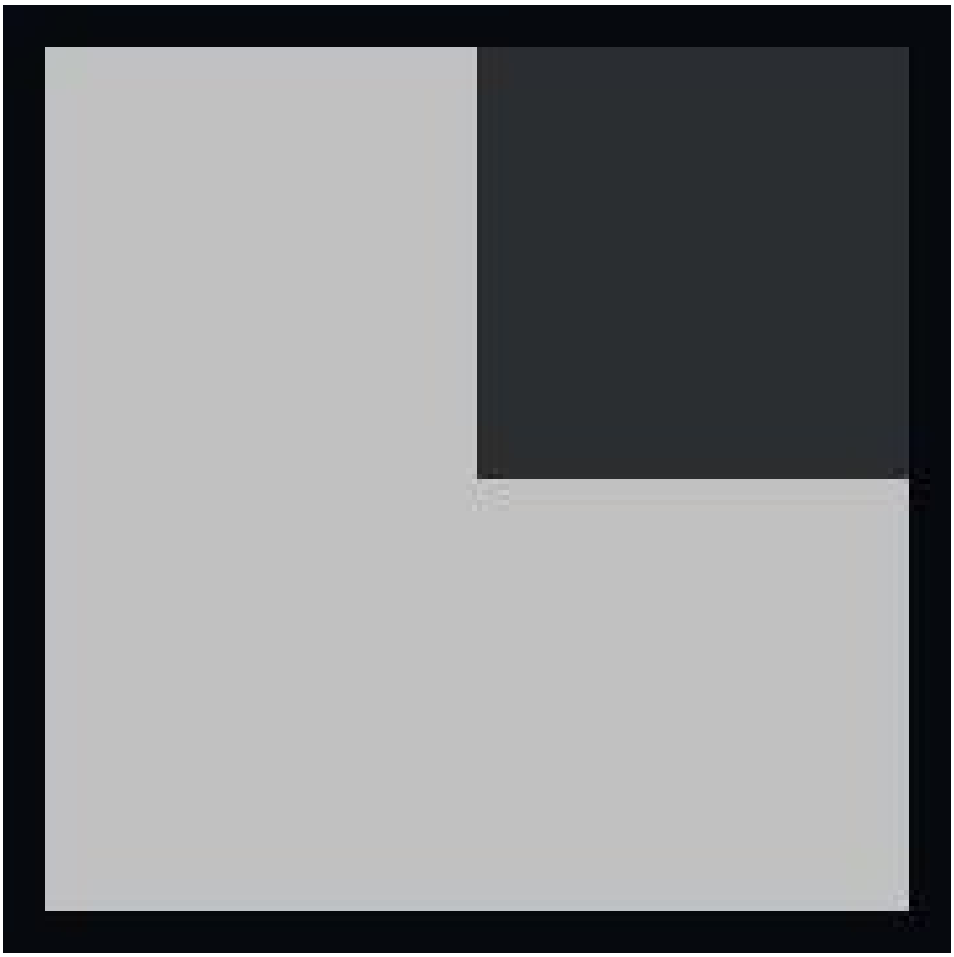
Remaining fraction 0.5



Remaining fraction 0.5

Open full-size image - /Script/InterfaceForge.IFCenterTint

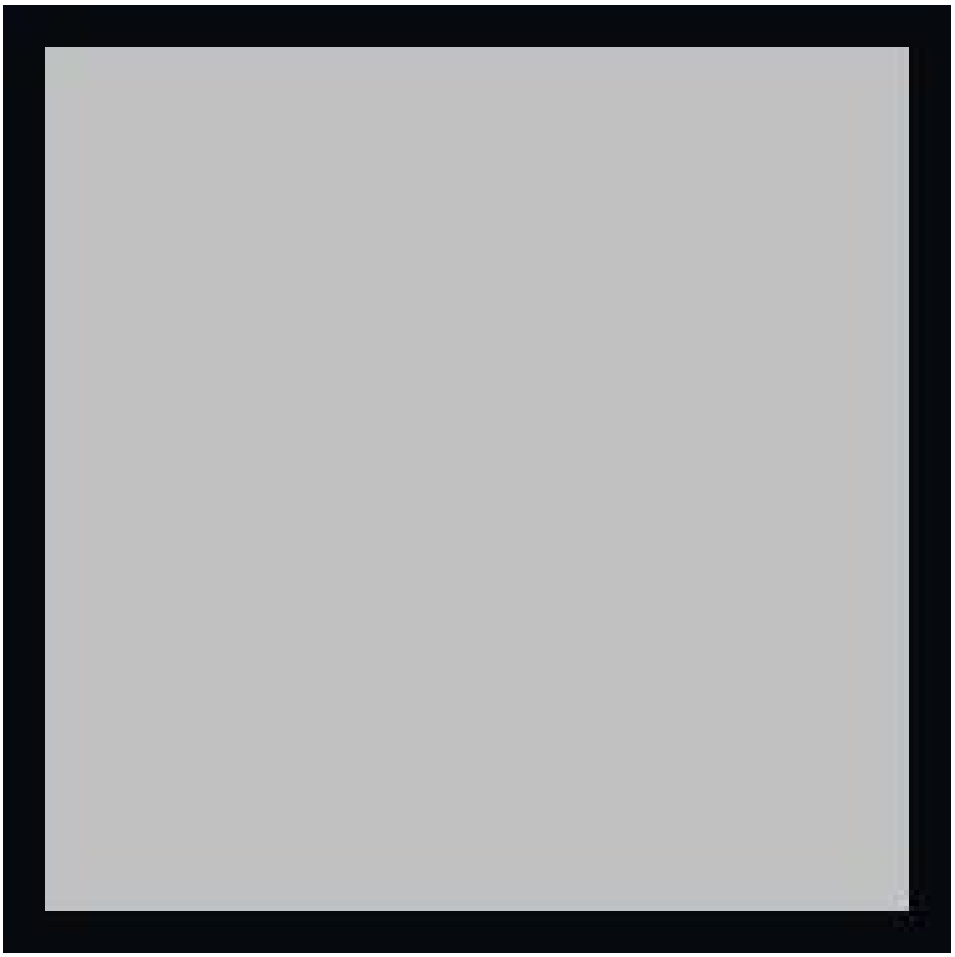
Remaining fraction 0.75



Remaining fraction 0.75

Open full-size image - /Script/InterfaceForge.IFCenterTint

Remaining fraction 1



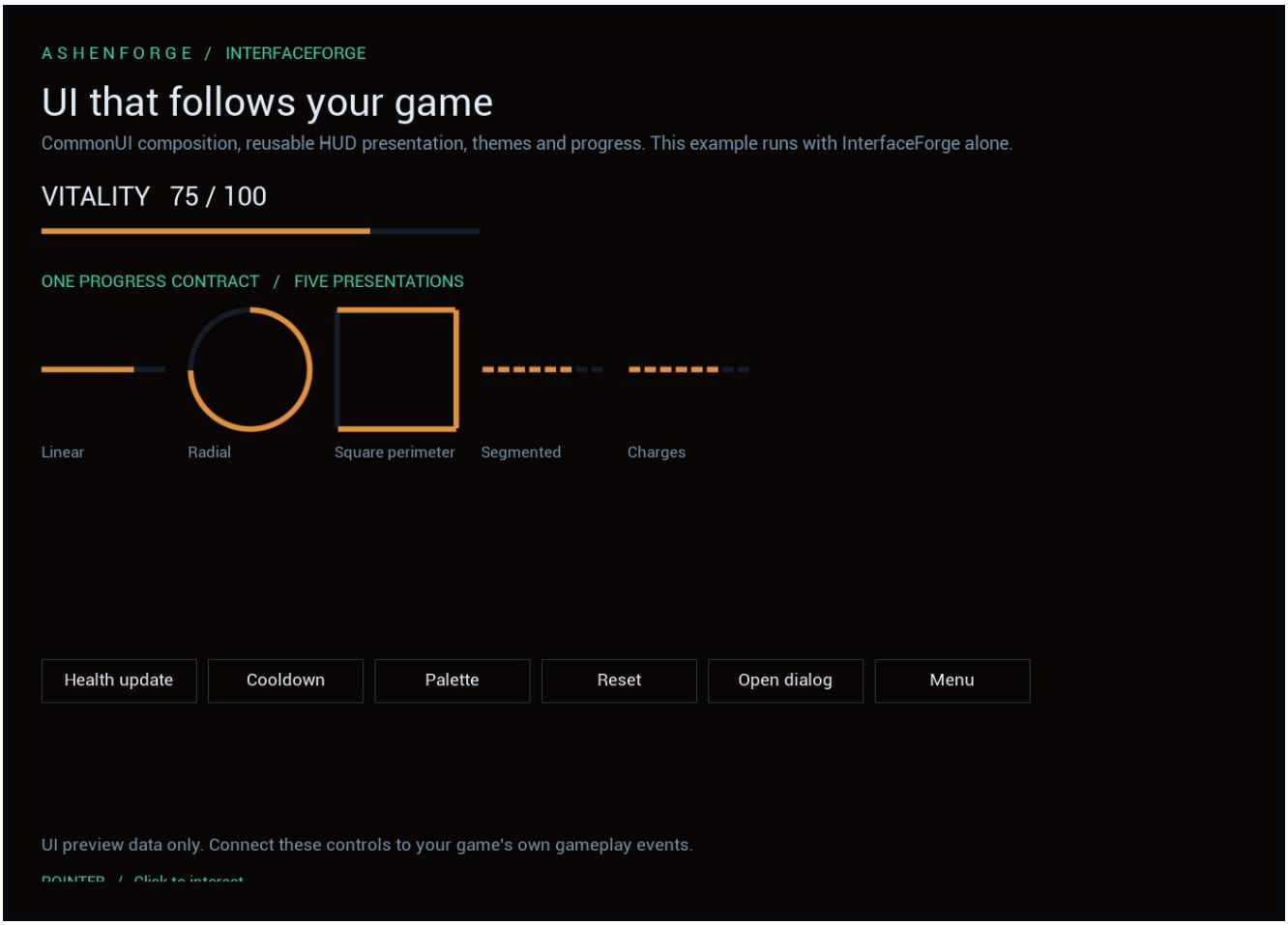
Remaining fraction 1

Open full-size image - /Script/InterfaceForge.IFCenterTint

Native example fallbacks and legacy prototypes

These code-created interfaces still exist in source. They are separate from the saved Blueprint menus/HUDs used by the active examples. The workshop/tint panels are old prototype/QA surfaces, not additional finished genres.

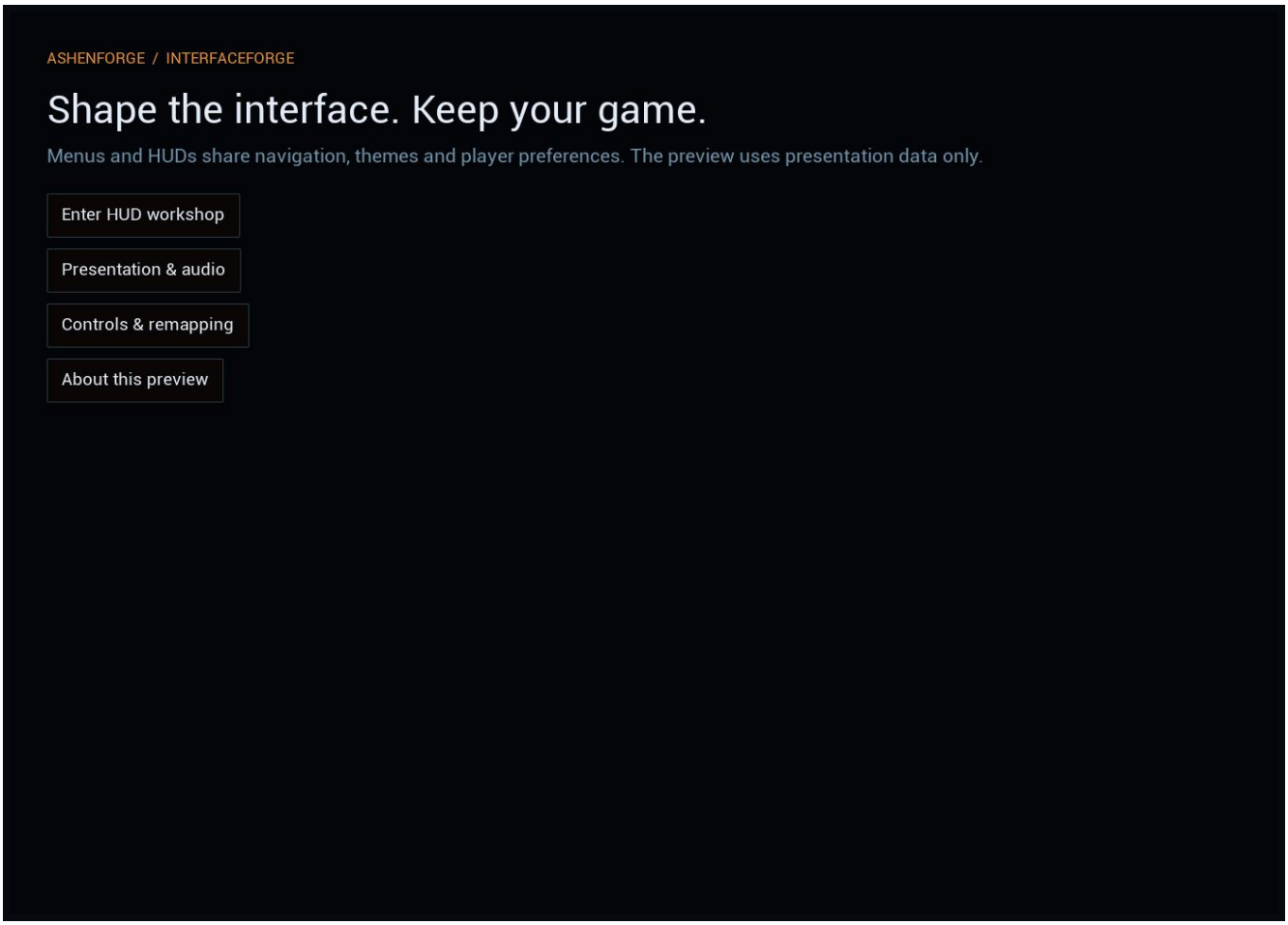
ForgeWorkshopWidget



/Script/ForgeExamples.ForgeWorkshopWidget native example body

Open full-size image

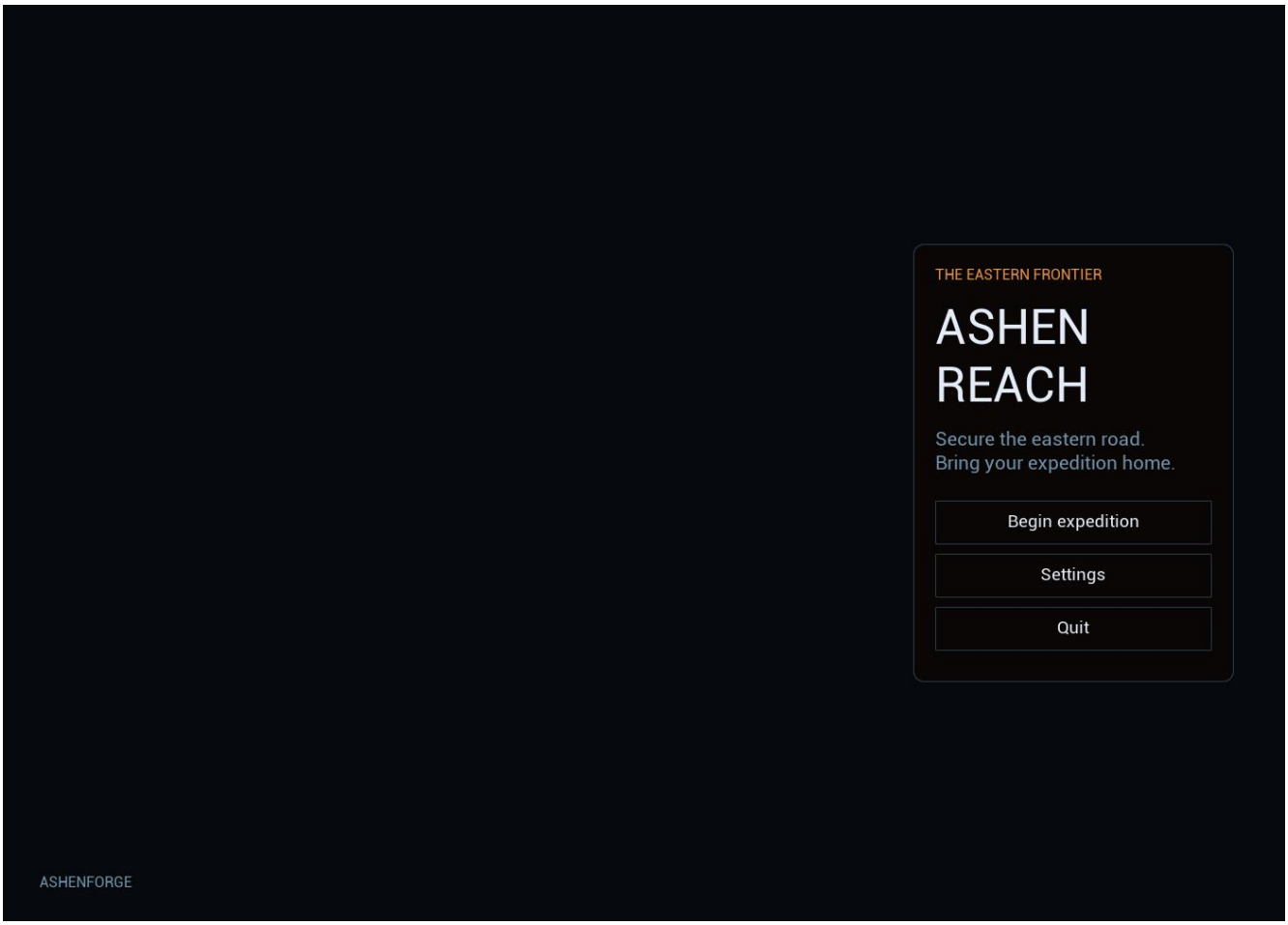
ForgeMenuScreen



/Script/ForgeExamples.ForgeMenuScreen native example body

Open full-size image

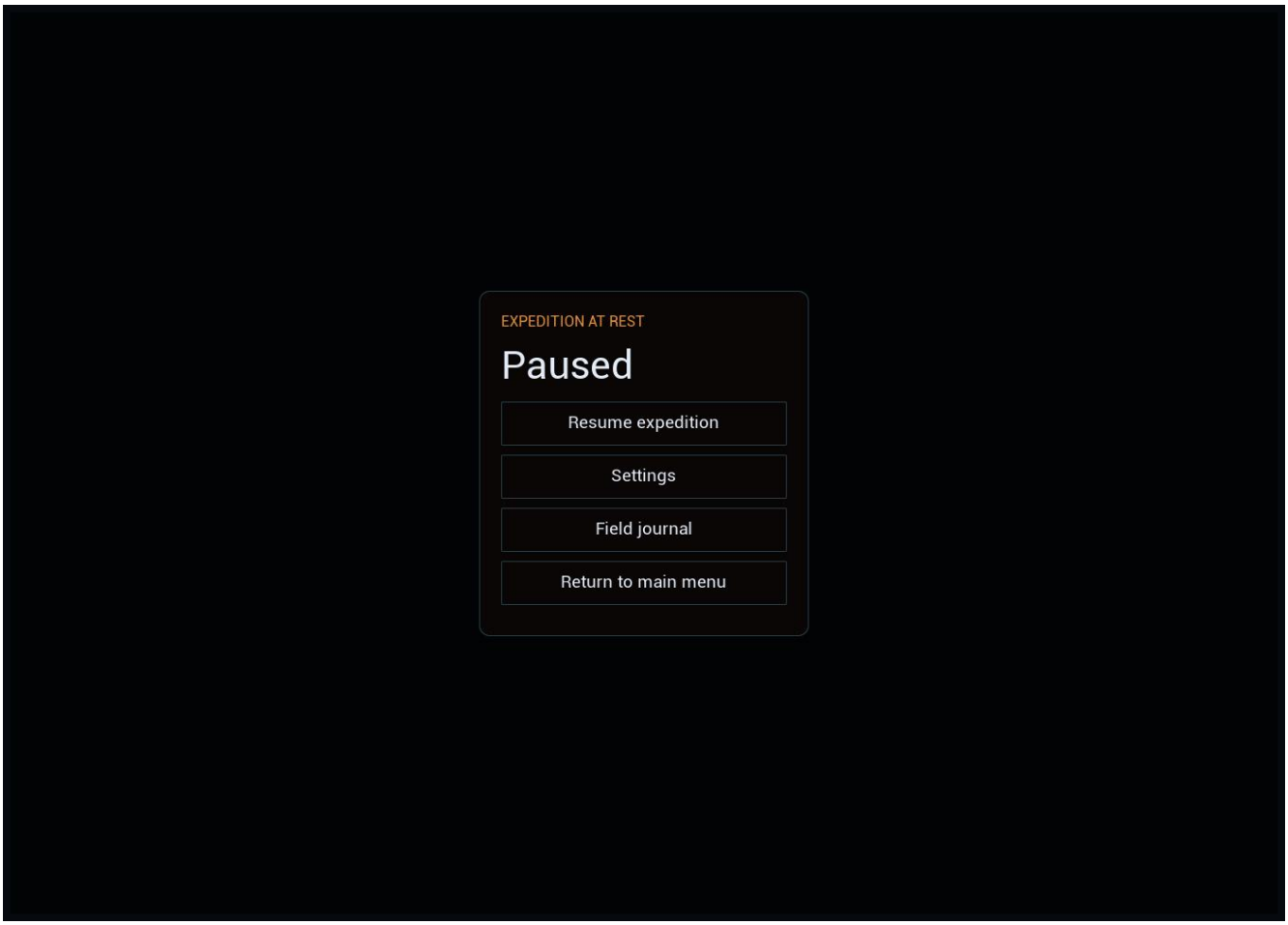
ForgeMainScreen



/Script/ForgeExamples.ForgeMainScreen native example body

Open full-size image

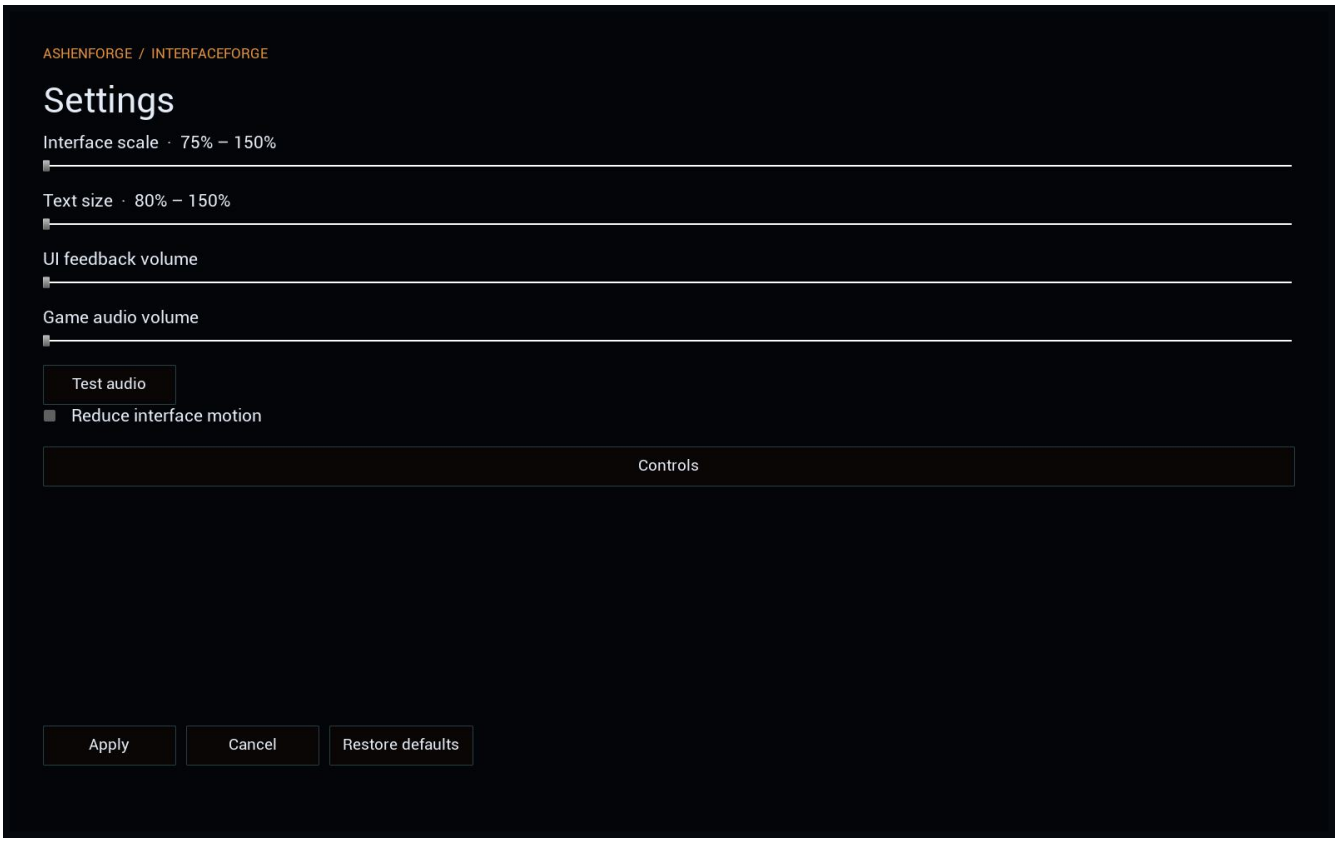
ForgePauseScreen



/Script/ForgeExamples.ForgePauseScreen native example body

Open full-size image

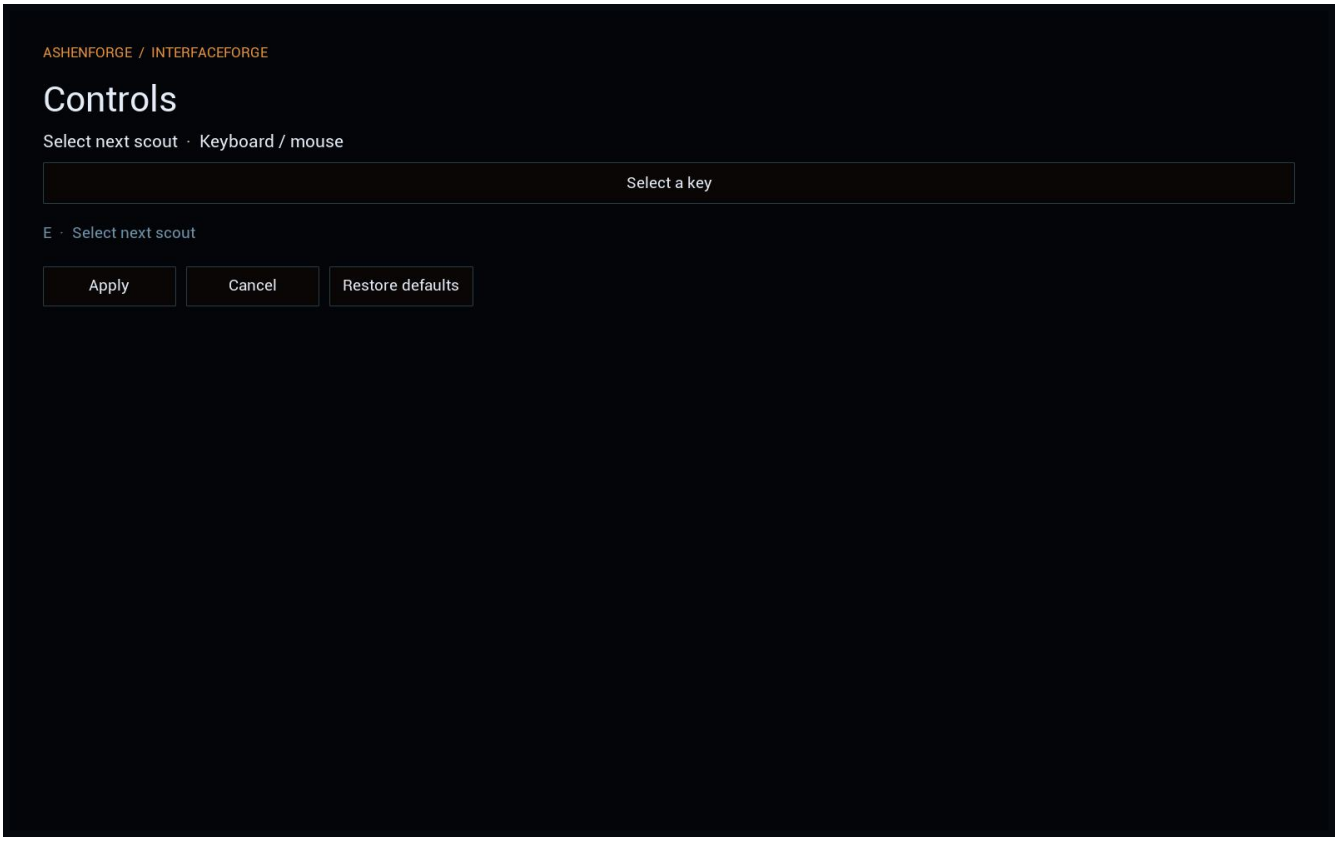
ForgeSettingsScreen



/Script/ForgeExamples.ForgeSettingsScreen native example body

Open full-size image

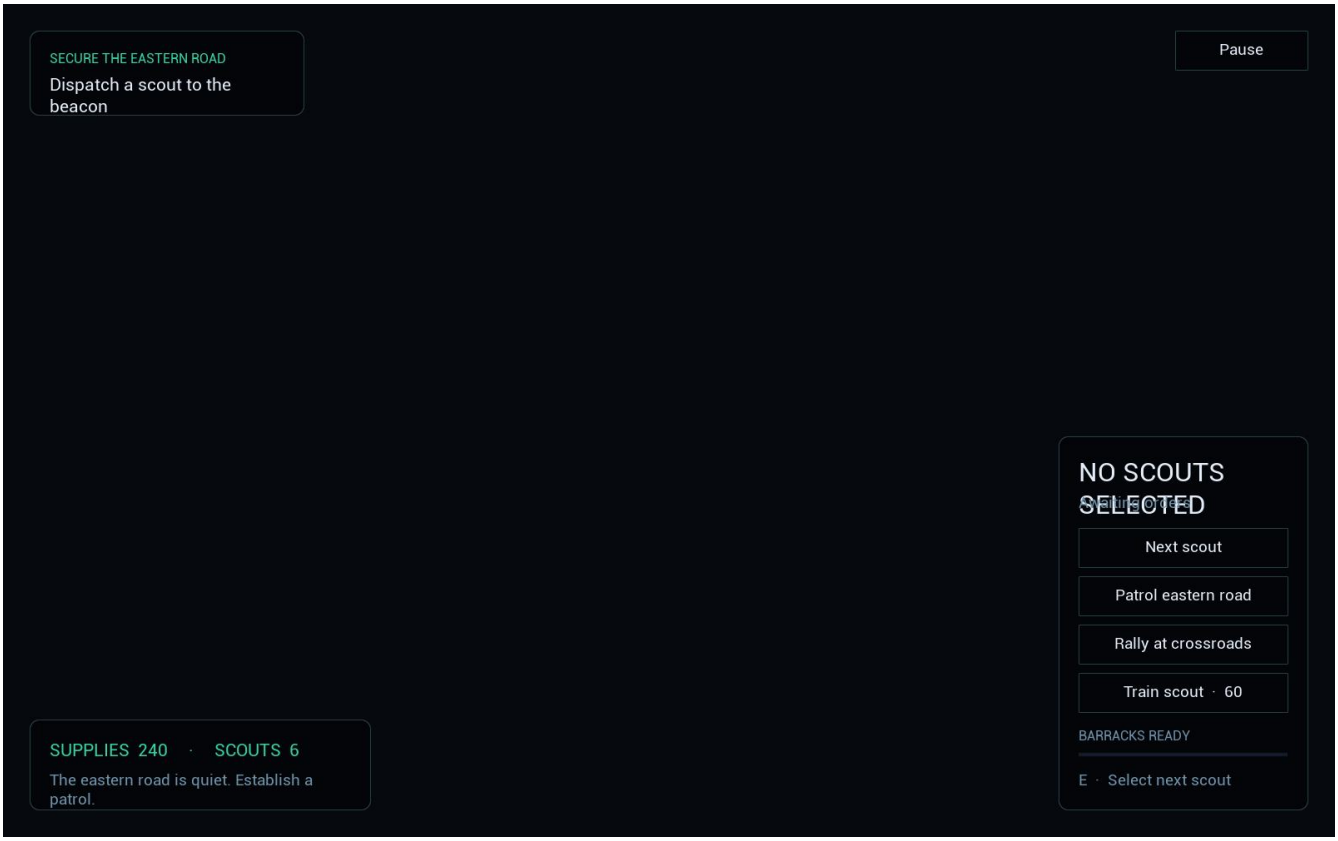
ForgeControlsScreen



/Script/ForgeExamples.ForgeControlsScreen native example body

Open full-size image

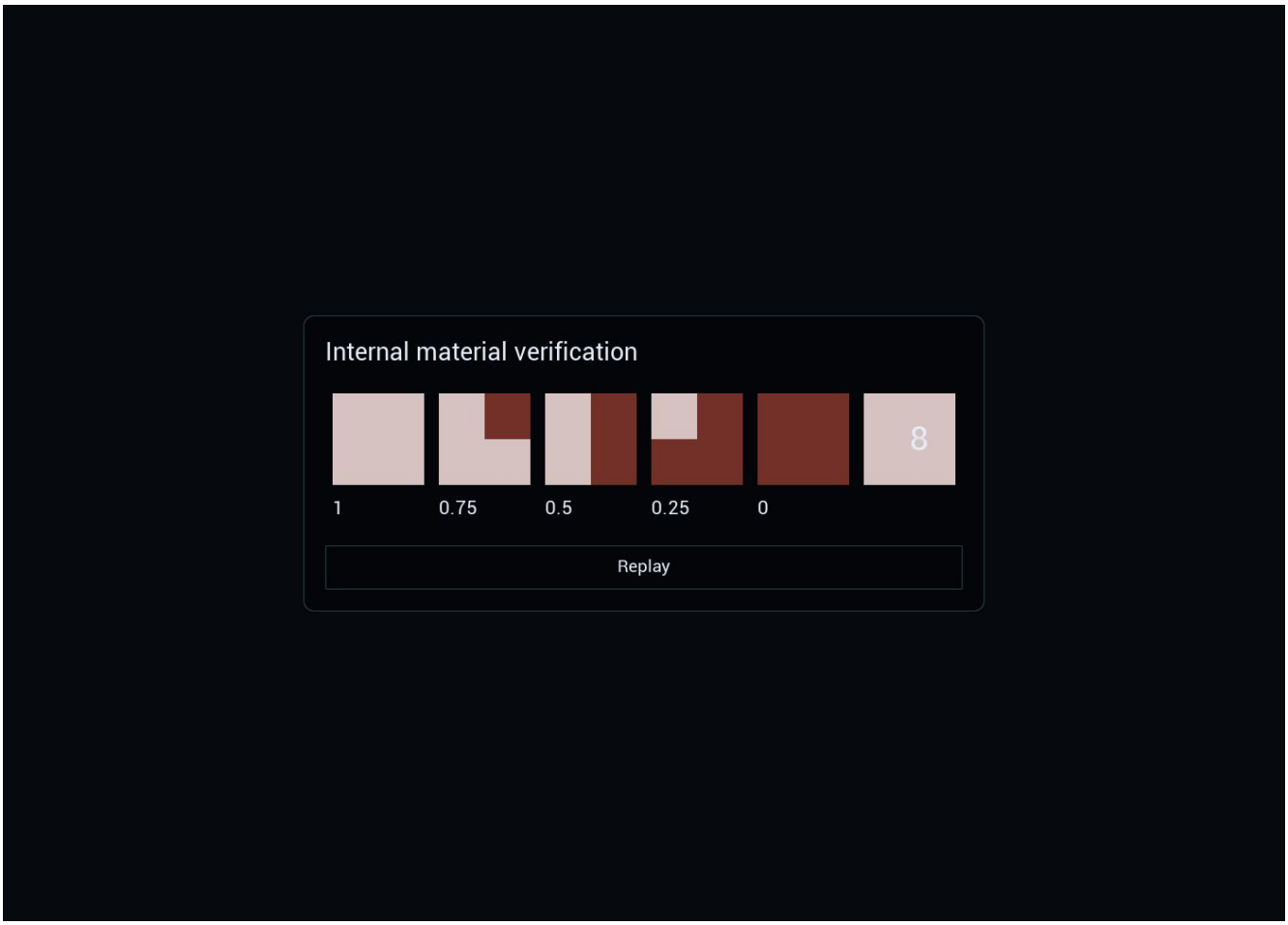
ForgeFieldHUD



/Script/ForgeExamples.ForgeFieldHUD native example body

Open full-size image

ForgeTintReview



/Script/ForgeExamples.ForgeTintReview native example body

Open full-size image

Coverage checklist

Saved Widget Blueprint	Parent	Render	Designer
WBP_AshenControls	IFScreen	Image	Image
WBP_AshenMainMenu	IFScreen	Image	Image
WBP_AshenPause	IFScreen	Image	Image
WBP_AshenRTSHUD	IFScreen	Image	Image
WBP_AshenSettings	IFScreen	Image	Image
WBP_EmberAction	IFEntryWidget	Image	Image
WBP_EmberActionButton	IFButton	Image	Image
WBP_EmberRPGHUD	IFScreen	Image	Image
WBP_EmberRoot	IFRootLayout	Image	Image
WBP_FieldCard	IFEntryWidget	Image	Image
WBP_FieldCards	IFScreen	Image	Image
WBP_FieldJournal	IFScreen	Image	Image
WBP_ForgeActionButton	IFButton	Image	Image
WBP_ForgeAppliedPrompt	IFActionPrompt	Image	Image
WBP_ForgeConfirmation	IFDialog	Image	Image
WBP_ForgeContextHint	IFEntryWidget	Image	Image
WBP_ForgeFlowRoot	IFRootLayout	Image	Image
WBP_ForgeLoading	IFLoadingScreen	Image	Image
WBP_ForgeNotification	IFEntryWidget	Image	Image
WBP_ForgeRoot	IFRootLayout	Image	Image
WBP_ForgeTooltip	IFEntryWidget	Image	Image
WBP_ForgeWorldStatus	IFEntryWidget	Image	Image
WBP_HUDReadout	IFEntryWidget	Image	Image
WBP_JournalEntry	IFEntryWidget	Image	Image
WBP_OutpostCommand	IFEntryWidget	Image	Image
WBP_OutpostCommandButton	IFButton	Image	Image
WBP_ProductionReadout	IFEntryWidget	Image	Image
WBP_WatchpostEntry	IFEntryWidget	Image	Image
WBP_WatchpostSurvey	IFScreen	Image	Image
WBP_Watchpost	IFScreen	Image	Image
WBP_WatchpostReadout	IFEntryWidget	Image	Image

Additional complete-widget renders

Standalone screen renders below omit the 3D background and may have no activation-bound model. Prefer the live windows above to judge actual composition and content. They are retained so every captured widget image remains accessible.

Image	Context
Widget-WBP_AshenControls.png	Isolated widget/editor capture; see primary gallery for populated live composition.
Widget-WBP_AshenMainMenu.png	Isolated widget/editor capture; see primary gallery for populated live composition.
Widget-WBP_AshenPause.png	Isolated widget/editor capture; see primary gallery for populated live composition.
Widget-WBP_AshenRTSHUD.png	Isolated widget/editor capture; see primary gallery for populated live composition.
Widget-WBP_AshenSettings.png	Isolated widget/editor capture; see primary gallery for populated live composition.
Widget-WBP_EmberRPGHUD.png	Isolated widget/editor capture; see primary gallery for populated live composition.
Widget-WBP_FieldCards.png	Isolated widget/editor capture; see primary gallery for populated live composition.
Widget-WBP_FieldJournal.png	Isolated widget/editor capture; see primary gallery for populated live composition.
Widget-WBP_ForgeConfirmation.png	Isolated widget/editor capture; see primary gallery for populated live composition.
Widget-WBP_WatchpostSurvey.png	Isolated widget/editor capture; see primary gallery for populated live composition.

What these documents do not claim

The redesigned example hub, richer MMO-style HUD, minimap, horror interface and injury/poison/camera feedback requested for the new goal have not been implemented by this documentation task. They are not represented as existing features. Physical controller/touch testing remains unverified. These images are intended to let the current implementation be inspected honestly.

VERSION AND VERIFICATION

UIForge competitive quality pass

Verified local implementation pass, 2026-10-27. Previous functional receipts remain useful regression evidence; they do not establish visual parity or approval. The benchmark is Evil's RPG & MMO UI 6. Reference art is viewed remotely, never imported into the product. No image generation or copied competitor graphics.

Evidence and limits

The listing was read in a browser, including its expanded description. Gallery images 3, 5, 6 and 12-21 were inspected, with original remote images used when the lazy-loaded gallery stalled. Its linked preview video plays in the listing. Initial observed sequences include login, character selection/deletion confirmation and appearance selection. Exact animation curves/durations and underlying implementation are not established by screenshots. Later sampled video sequences also show the HUD and windows/tabs. Their motion is reference observation, not measured animation parity. The linked WebGL demo fails DNS resolution; linked documentation is currently inaccessible. These failures do not prevent independent development.

Seller claims include a scalable, recolorable art kit with reusable scripted controls, windows, overlays, tooltips, tabs and tweening. That is a claim about the Unity product, not evidence of Unreal compatibility, gamepad quality, multiplayer correctness or its internal performance.

The strongest visible result is a shared visual grammar: decorative headers, fine separators, consistent close buttons/tabs, restrained dark content planes, brighter colored actions, beveled resource/slot treatments and a recognizable palette across dense windows. Its HUD differentiates portraits, target/player resources, party, statuses, cast information, objectives, map, action rows and transient announcements. It also includes inventory/vendor/talent illustrations. Those do not expand UIForge into inventory, trade, equipment or progression mechanics.

The native UIForge baseline was inspected in UE5.8.3. Its working menu has a flat tan primary button, generic text actions, a single small frame treatment and a bright floor-heavy backdrop. Combat slots/resources are readable but visually thin; map, objectives and party still read as unrelated small overlays. Existing passing tests do not excuse those presentation gaps.

Feature and gap matrix

Need / evidence	Existing foundation to retain	Required finished result	Cost / proof
Coherent color variants (gallery 5)	Project/local themes, surface keys, fonts	Two genuinely complete theme variants; shared button/state/line/icon tokens, no widget-tree duplication	Medium; live theme swap across HUD, windows, prompts and settings
Distinct action states (video and menus)	CommonUI buttons, availability reasons, input focus	Named editable button skins with normal, hover/focus, pressed, selected and unavailable states; bounded feedback, reduced-motion support	Medium; actual pointer/navigation/press/disabled changes and no idle animation work
Window family (13,17-20)	Screen stacks, modal/back/focus ownership	Shared header/tab/close/content/footer language in menu, pause, settings, journal, spell/party detail and dialogs	High; open/close/back/transition interruption, long labels, no hidden focus or overflow
Dense usable RPG HUD (16)	Top-center target, models, six skills, Sakura tint	Refined resource silhouettes, portrait/identity treatment, status chips, cast hierarchy, compact peripheral controls; no opaque center clutter	High; real casts, resource/status updates, hover/controller tooltips and responsive review
Tooltips/list information (18-20)	Anchors, collection entries, selection/recycling	Consistent title/body/stat/help sections; edge-safe placement; input-aware activation and reason feedback	Medium; real data updates, edge positions, focus/tooltip ownership
Quests and ability browsing (18,19)	Journal, tracked objective, real six-skill adapter	Multiple selectable presentation entries backed by the actual encounter; coherent detail panes, filtering where useful	Medium; selection changes detail, tracking updates HUD, actions affect encounter
Sliders/select/tabs (13)	Transactional four-tab settings, engine-backed remapping/audio/video	Same visual standard as the HUD, readable values/help/errors; complete Apply/Cancel/Defaults behavior	Medium; retain actual old/new-key, reload and failed-validation tests
Announcement/loading presentation (15,16)	Notification queue/loading/screen transitions	Original compact announcement and objective-complete treatments, honest loading/progress and cancellable transition behavior	Medium; actual encounter/queue events and reduced motion
Mobile variants (6,21)	Device presentation contracts	Preserve extension points and meaningful touch sizing; never infer hardware readiness from a seller screenshot	Deferred hardware; software presentation only until devices available
Inventory/vendor/talent drawings (17,18,20)	Generic data/entry/action interfaces	Reusable icon/list/detail/window controls suitable for an external backend	Inventory, trading, equipment and progression rules excluded
RTS application	Existing armies, selection/orders/groups/production	Same quality of skins, hierarchy, controls, tooltips and transitions in a distinct strategy layout	High; real army selection/move/commands/queue and multiple selected units
Buyer presentation	Playable hub, actual maps, authoring guide	Representative playable flows and engine screenshots for every shipped full window; no explanatory cards masquerading as gameplay	Medium; public Blueprint customization proof, current packaged interactions

Original AshenForge direction

Keep the existing authored Emberglass identity but finish it: dark blue-black enamel, warm pale text, restrained silver/bronze structural edges, ember or cool mineral action accents. Major windows receive crafted headers; small combat widgets receive quieter treatments. Use an 8-unit spacing rhythm, consistent internal gutters and aligned baselines. Ornament belongs at corners/header/identity points rather than around every label. Large headings use Cinzel sparingly; ordinary controls remain legible body text. Avoid fixed-height wrappers that clip enlarged text. Themeable states must not depend on color alone.

Do not imitate the competitor's swirls, red/blue orbs, logo, portraits or content. Independently authored vector paths and small rasterized nine-slice assets are editable sources. Art provenance and font notices stay explicit. No new gameplay systems merely to fill a window; use existing real encounter/army data and small example-only presentation adapters.

Execution milestones and acceptance

1. **Public appearance foundation.** Named button skins, full states and finite feedback through CommonUI; theme/local overrides, custom Blueprint callbacks, reduced motion, cleanup and no idle ticker. Build and inspect on real menus before propagating.
2. **One completed RPG flow.** Main → settings → confirmation → playable HUD → journal/ability detail → pause → main. Finish window family, controls, action/resource/target/party/map hierarchy and transitions together. All essential behavior uses public features and editable Widget Blueprints. No temporary placeholder screens in the hub.
3. **RTS consistency.** Apply the finished grammar to selection/army management/resources/commands/production without changing strategy behavior or borrowing Game Master's design. Keep an open central playfield.
4. **Customization and feedback.** Two complete theme variants, replaceable entries/brushes/animations/data sources, actual health/status/announcement feedback. Every addition solves a visible developer problem and appears in a working example.
5. **Release review.** UE5.8.3 build, independent core load, clean Blueprint compiles, remap/session regression, software input transitions and popup/focus/back checks; native packaged interactions; 720p/1080p/ultrawide/4K and enlarged text; clipping/tooltips/layering/animation interruption review. Record actual measurements and meaningful failures. Controller/touch/unplug remain hardware-deferred; performance is measured rather than promised.

Only finished, tested screens enter the current gallery/default hub. Historical images stay labeled historical. Public documentation must show exact nodes, asset paths, expected results and screenshots from the matching build. No claim of competitor parity from feature counts or static screenshots alone.

Work record

- Authoritative project remains C:/UEProjects/ashenforge-plugins/UIForge/Project. Bounded evidence lane: Workspace/Rebuild583/CompetitorPass. Reuse CoreLoadHost and Playable; no new hosts/clones/engine caches.
- Original control/theme backup: before-controls.zip (1,341,503 bytes of bounded original affected source/assets). Native current menu capture: Docs/Images/CompetitorPass/before-main.png.
- No Git metadata was found at the active project/ancestors; do not initialize or reset Git to conceal that. Preserve files and keep bounded explicit backups.
- Earlier foundation checkpoint (superseded by the current receipts below):9 button runtime checks,9 resource/setup/capture checks plus five actual visual/pixel cases,92 clean Blueprint compiles, current core-only16-package/four-redirect load and actual native standalone menu/confirm/cast/damage/pause/settings. Evidence: CompetitorPass/{button-runtime,resource-runtime,resource-visual,blueprints-current,core-load-current,native-foundation}.json. This does not accept the remaining visual/product milestones.

Current implementation and runtime proof - 2026-10-27

Milestones1-4 are implemented locally through public controls and editable Designer assets. Current gates:95 saved Blueprints;26 independent core packages/four redirects;9 button-state checks;13 actual ability-browser checks;13 public composition/Blueprint-source checks;24 settings-session checks;12 MMO/17 army scenarios;14 software input cases; actual native key capture→Apply→reload→gameplay→restore. Roster sizing is verified at0/1/24 selected units. Its actual TileView test uses2,000 stable presentation objects with30-40 realized entries, then restores the real army models. These are scoped tests, not a simulated2,000-unit army or UI GPU certification.

Actual source clients measured1280×722,1920×1082 and2554×1082. A3840×2160 request is desktop-clamped to2554×1354 and does **not** certify4K.150% UI and text review found a bottom RPG shortcut overflow; Auto Height now grows those controls upward. Native remapping preserved the original Three binding after proving Y. After clean editor shutdown, both original preference files were restored byte-for-byte; both gameplay saves were asserted unchanged.

Current screenshots have been recaptured after the Load/Controls/ability/roster corrections. The current installed-source package builds/cooks/stages in88.42s and passes27 scoped native observations, including process-restart old/new key checks, actual RPG/RTS interaction and clean exits. Both source candidates match installed/built runtime source/config/content; the follow-up delta is documentation only. The documentation site builds103pages with no broken local links/images. This record does not assert owner approval or that UIForge has surpassed the competitor.

VERSION AND VERIFICATION

Competitive presentation verification - 2026-10-27

Current source evidence is in Workspace/Rebuild583/CompetitorPass. The current gallery shows this revision; dated receipts below describe earlier passes.

Gate	Actual current result
Source Editor	UE5.8.3 CL58210709 build passes;95 saved Blueprints compile clean
Independent core	Fresh existing plugin-only host loads26 packages/four redirects; no ForgeExamples, MassForge or inventory module/dependency
Public appearance	9 real CommonUI state/finite-feedback/reduced-motion checks; custom Linear brush clipping/reverse/local fallback inspected; original center tint retained
Game actions	13 ability-browser checks;12 MMO/17 army scenarios; delayed closing-book actions are rejected
Custom Blueprint	13 region/theme/stable-binding and saved Blueprint-only source/entry/action checks
Settings	24 session checks; native Aegis→Y capture/Apply, new PIE reload, Y gameplay and restoration to Three verified
Input	14 software Slate cases for focus/cursor, drift/jitter, held-stick handoff and HUD/pause/modal switching; physical device testing deferred
Large collection	2,000 stable presentation objects;30-40 realized TileView entries; recycled commands retain current contexts; real source restored
Roster sizing	Actual0/1/24 selected-unit cases produce grid heights0/50/150 through the saved Blueprint graph
Responsive	Measured1280×722,1920×1082,2554×1082;150% UI/text inspected;4K request clamped to2554×1354, so4K remains unverified
Data preservation	Owned source editor closed cleanly; two original preference files restored exactly; both game saves asserted unchanged by SHA256

A bounded current editor profile recorded1,162 post-warmup samples: ProgressPaint mean0.0060ms/p950.0068ms, RootAdvance mean0.0068ms/p950.0075ms, MapPaint mean0.0042ms/p950.0051ms. These are individual inclusive counters in one small static MMO scene with other editors running. They must not be summed or advertised as the complete UI/GPU budget. The2,000-model test concerns UI virtualization, not a2,000-unit simulation.

Current Win64Development build/cook/stage/archive passes in88.42s from the exact installed source example ZIP. The native packaged receipt records27 scoped observations: hub/menu/settings, key capture/Apply, actual skill use, repeating attack, save/load/tracking, keyboard modal navigation, RTS selection/groups/movement/production/combat and two clean exits. A complete process restart preserves Y;3 no longer activates Aegis, whileY consumes12Resolve and begins12-second recovery. Source/config/content archive bytes match the built host; only documentation changed afterward. Native source and synthetic input are separate evidence. No owner approval, competitor-parity, hardware-controller, touch, multiplayer, other-engine or other-platform claim is made.

Current verification - UE 5.8.3 / Win64

RPG combat/input correction - 2026-10-26

Current local revision:92 saved Blueprints compile; UE5.8.3 Editor and Win64 Development BuildCookRun pass. The fresh core-only host loads16 packages and four legacy redirects with ForgeExamples, MassForge and InventoryForge absent. New public options are IFSurface/theme brush keys, Dialog Surface Style and Action Prompt Wrap Label. Existing core cooldown material is preserved.

- 17 rebinding checks cover settings-session ownership, draft/cancel, conflict rejection, Apply, live applied prompt, old-key rejection, new-key activation and camera-relative remapped movement. A new PIE local player reloads the saved R/O profile.
- Native mouse/key interaction changes Aegis to Y through InputKeySelector and Apply; native Y starts its actual cooldown. Five defaults checks pass. User's original Enhanced Input/UI save files are restored exactly afterward.
- 14 software Slate checks cover HUD, pause and modal device switching, focus/cursor behavior, drift/jitter rejection, held-stick handoff and simulated disconnection.11 camera checks cover actual view target, lens/zoom/pitch limits, software right-stick orbit and pause ownership.
- RPG12 and RTS17 gameplay scenarios pass on the reworked controls/presentation. The later RPG map change preserves the RTS transform; it adds explicit RPG artwork bounds and removes the RTS frustum from the RPG map.
- Three actual viewport cases are1280×722,1920×1082 and2554×1082. These are measured client extents; nominal requests were1280×720,1920×1080 and2560×1080.150% UI plus150% text preview was visually inspected, including the content-sized pause card.
- Native packaged main/modal navigation, Escape cancellation/pause/back, Cinderbolt cast/Resolve/cooldown/target damage, continuing melee, objective progression and eventual death UI are observed. A wrapped pause heading exposed footer/control overflow; the card/buttons now size to content. Content-sized main/pause cards replace the overflowing fixed-size version. Six PIE Back checks cover native pause/resume, selector cancellation, settings Back and modal cancellation. The configured action routes through CommonUI before the editor Stop shortcut, scoped to player-owned UI. Pointer/keyboard HUD focus now returns to the viewport after transitions while preserving focused text/key editing.

Receipts: Workspace/Rebuild583/RPGRework. Builds use the existing bounded CoreLoadHost/Playable outputs. No new project clone, engine copy, publishing, Git reset or inventory development. Controller hardware, actual unplug, touch, native right-drag/left-click target-picking, new multiplayer, other platforms/engines and isolated performance certification remain unverified. These limits are not converted to passes by software events or screenshots.

This page covers the **2026-10-25 army/HUD rework**, after the owner rejected the previous encounter and presentation. It distinguishes native input, runtime API scenarios, software device events and builds. Visual quality remains subject to owner review; passing a test does not establish that the product surpasses another game's art direction.

Animation follow-up - 2026-10-26

HumanMotion_v022 is integrated into the three saved maps through the separate ForgeExamples adapter. Editor build, seven-clip/441-pose source parity (maximum transform-component error1.07e-6),16 live animation checks,12 RPG checks and17 army checks pass. Targeted checks cover actual displacement/facing, active Animation Blueprint, feet, stopping, melee/cast slots, cast HUD, paused pose/damage clocks, resumed contacts and held death pose. Blueprint customization is documented in example animation.

The current Win64 Development package builds/cooks/stages successfully in32.77s. Native1600x900 mouse/keyboard checks cover both menus, RPG approach/repeated attacks/target death/cast/cancel/pause/return, and RTS24-unit selection, formation movement, attack-move and victory against18 opponents. Screenshots are in the animation guide. These are local adapter checks; hardware controller/touch, other platforms and multiplayer remain unverified. MassForge's seven source asset hashes are unchanged; no VAT bank or source FBX copied. Core runtime code/content is unchanged.

Authoring failures were corrected before these checks: source editor Control Rig references prevented a safe straight duplicate, Blend Space samples lost their missing asset references during repair, and the death auto-blend flag was originally set after montage-instance initialization. Imported skeletal keys now match the source and final runtime checks pass. A UE Animation Blueprint editor destructor crash occurred while closing an earlier authoring editor; the local report was dismissed without sending. Packaged testing did not reproduce it. This is recorded separately from successful gameplay tests, not counted as a pass.

Environment follow-up - 2026-10-26

The saved hub/RTS/RPG worlds received an environment-only cleanup. Upright/grounded geometry, connected paving/gates, foundations and a shared original terrain replace malformed scenery. The minimaps are rebaked. Navigation and17 army/12 RPG runtime checks pass again, scheduled against the existing clamped encounter step without changing gameplay time. Current environment captures are in the gallery; world editing and scope covers assets and limits. Earlier UI/core receipts below retain their original revision scope. New package/runtime/source evidence is recorded separately under WorldPolish.

Builds and independence

The installed engine reports **5.8.3 CL58210709**, with compatible descriptor baseline **5.8.0**. UIForgeEditor builds. The extracted core separately compiles **Editor Development, Game Development and Game Shipping**. A plugin-only host loads all **16 core packages** and four legacy redirected paths without ForgeExamples, MassForge or inventory. **76 saved Blueprints** compile up to date with no failures. The extracted example builds, cooks, stages and archives its four maps; the final cook completes in44.30seconds. These are Win64 results, not universal engine/platform certification.

Operated interfaces

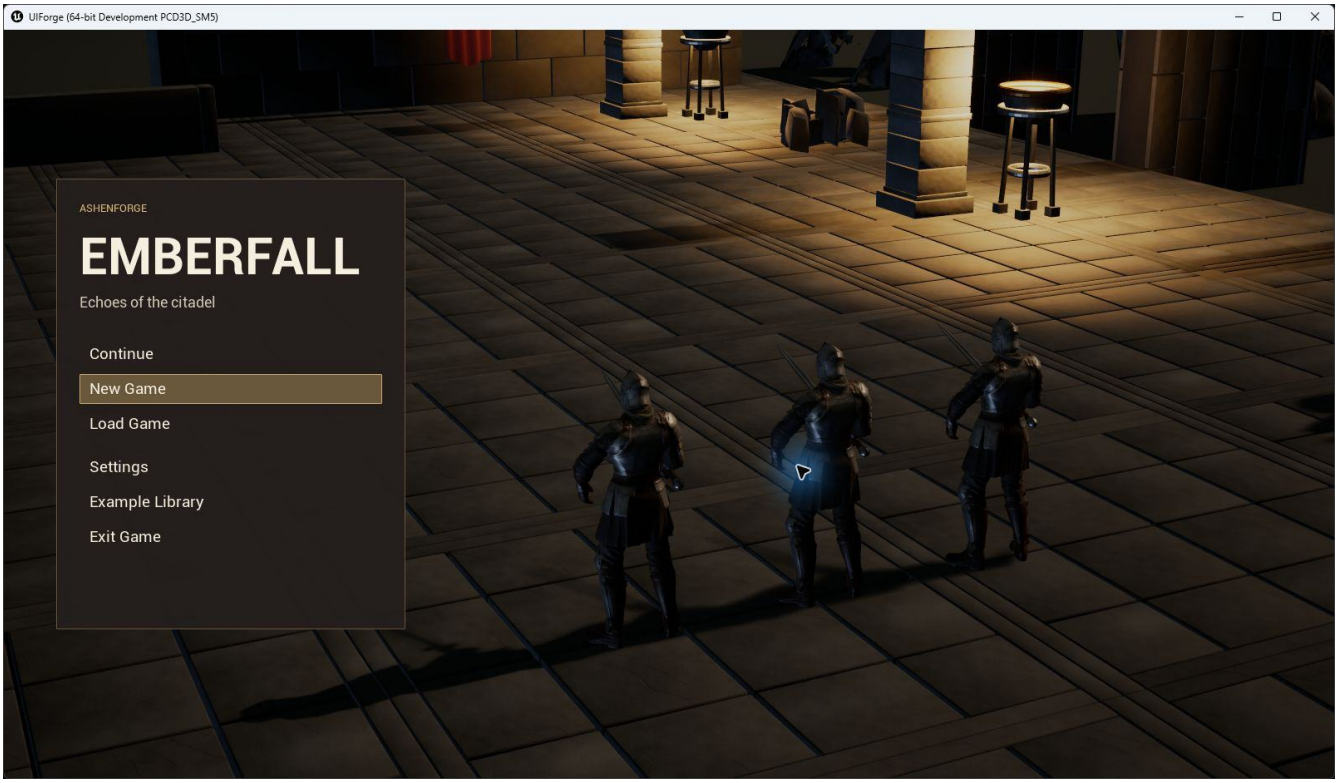
Area	Observed result and scope
RTS native editor	Eight checks: empty click deselects; marquee selects16; Ctrl+1 stores and1 recalls the same16; right-click moves all16 more than100cm while the unselected eight stay within50cm; same-type double-click selects18; a portrait selects its own unit; T produces a reinforcement. Native Shift-click is not claimed.
RTS runtime API	17 checks cover24 friendly/18 hostile units, additive removal, role selection, groups, selected-only orders, queued moves, actual navmesh movement, Stop/Hold, recruitment/cancellation, reinforcement and delayed combat contact.
RPG combat	12 runtime checks cover persistent attack, approach, repeated contacts, victim death, another target, cast priority, windup, movement interruption, legitimate recovery and pause/resume. A read-only observer records target death after one native attack key; allies also contribute damage.
Packaged RTS	Native hub travel, selection, contextual movement, Escape/pause, save/load and exit confirmation. UI scale124% previews then cancels to100%. Recruit remapped T→Y updates its prompt: T no longer recruits; Y spends60gold and creates a reinforcement. Binding restored to T through Apply.
Packaged RPG and menus	Native1 approaches and attacks to Defeated. Mend raises health32→64, Resolve100→88 and starts recovery. Escape opens Pause; keyboard opens Settings. Back restores Settings and Enter reopens it. First Escape dismisses the Video popup; second returns to Pause. Main/Cancel/Confirm and Exit operate through their modal flow.
Menu staging	Both source examples move characters, return to Main and restore idle actors with nameplates hidden. Seven checks per genre prove every logical unit field stays unchanged. Native package checks restore Emberfall after combat/death and Ironfront after a movement order. Final RTS staging clears the watchstone prop.



Native packaged army selection after movement



Native packaged RPG health, Resolve and recovery



Packaged main menu restored after combat and death

The RTS picture shows a fought encounter with22 living soldiers and14 selected; it is not the fresh24-unit start. Images show observed states and do not substitute for interaction records.

Blueprint, settings and input

- **11 entry/model checks:** saved Blueprint ability entry initialization, fraction signals, release/reuse, recycled command context, finite/clamped values and collection membership/order/duplicate handling.
- **2,000-model TileView:** the actual saved roster uses **30 displayed widgets** initially, after end scrolling and after reorder. Displayed commands keep the current context. This tests UI virtualization, not2,000 simulated soldiers.
- **20 Blueprint-only tutorial checks:** saved controller/game mode/menu/HUD/custom entry/pause/settings/dialog flow in the development project, stable model updates, actual world pause, cross-tab drafts, cancellation and unpaused return. Earlier independent-host tutorial/provider proofs remain historical; a new independent tutorial host was not repeated for this revision.
- **14 software input steps:** controller presentation/focus/cursor, drift thresholds, held-axis release after keyboard takeover, one-pixel mouse jitter, meaningful motion, keyboard return, pause, modal and simulated disconnect. Synthetic Slate events are not physical-controller or unplugging certification.
- **Submenu focus:** measured keyboard/software-gamepad runs restore Choice3 after Settings. Native packaged Enter reopens Settings after Back.

Settings retain prior scoped24 session,13 Blueprint-provider and9 real SaveGame-failure/rollback checks. The previous native forced-GC popup run proved its lifetime repair. This revision also operates remapping, scale preview/cancel and popup Back in the army/RPG package. The earlier provider/GC suites were not all repeated.

Visual review and performance

Current engine captures cover both menus/HUDs, settings, controls, confirmation, RPG journal and hub at requested1280×720,1920×1080 and2560×1080. Measured client dimensions are recorded; borders can slightly change them. At720p,150% UI and150% text previews remain operable. Review corrected caption clipping, party backgrounds, command wrapping, dock overlap and empty menu backdrops. The gallery shows current windows; HUD authoring describes their public features.

An editor CSV contains **7,687 frames** from the army scenario. Instrumented game-thread p95 values: root advance0.0082ms; progress painting0.2369ms; map painting0.0075ms; example encounter tick0.2333ms; model refresh0.1682ms. Active readout refresh p95:0.036ms. Another Unreal game/editor was running. These are instrumented scope timings, not isolated FPS/GPU benchmarks or final-package performance guarantees. Capture preceded the final focus/menu-staging corrections.

Failures found and corrected

The former MMO attack was one-shot. Persistent intention now approaches, recovers and repeats until interruption or death; explicit casting takes priority. RTS primary presses were suppressed by NoCapture in this engine: mixed HUDs now use temporary capture. Control-group modifiers are captured at keydown rather than delayed action delivery. Held controller axes neutralize after keyboard takeover.

Map-footprint painting appended an element from its own array and triggered an assertion; copying the point before append fixes it. A measured Settings return selected Choice0 instead of Choice3. Removing premature initial-focus selection for an unchanged input presentation lets CommonUI restore focus. Returning after combat exposed an empty backdrop; visual-only actor staging fixes it without mutating saves. A final visual check also found a staged soldier intersecting the watchstone; the formation was moved into a clear lane.

Early harness attempts inspected generated templates, attempted private-field reads or sent input before settled transitions. They are not counted as passes. Corrected checks use saved Designer trees, observable actors and bounded screen-state waits.

Delivery boundaries

Raw logs, receipts, captures, archive hashes and installed-source checks remain in the existing **Workspace/Rebuild583/ArmyHUD** lane. Earlier Todo receipts retain their historical scope. Final source candidates are member/CRC/hash audited and compared to the exact installed build host. No source engine, SDK stash or duplicate project was created for this rework.

Physical controllers/disconnection, touchscreens, consoles, non-Windows platforms, other engines, split-screen and multiplayer remain unverified. The playable package is **Development**; core Shipping compilation is not a Shipping example playthrough. RTS navigation/separation and RPG combat are small local adapters, not production army simulation or MMO networking. Saves preserve units/resources/progress; groups, orders and recruitment queues reset on load. Video rows cover VSync, frame limit and shadow quality; display-mode Keep/Revert remains an extension point. Cross-store Apply is not an atomic disk transaction.

Game Master's design stays exclusive. Sakura's inspected center-origin tint remains unchanged. Inventory mechanics are absent. No Fab approval, publication, license change or marketplace AI-classification decision is implied.

VERSION AND VERIFICATION

Artwork and dependencies

The runtime core depends on Unreal **CommonUI/CommonInput, Enhanced Input, UMG and Slate**. It does not depend on ForgeExamples, MassForge, inventory, the documentation website or editor authoring tools. Engine fonts, meshes and utilities remain Unreal dependencies; they are not copied out as a separate art library.

The paths below identify editable sources in the development checkout. The separate source example ZIP mirrors the SVGs, atlas and asset register under **ArtSource**. Select that local source when reimporting or replacing an example texture; the core does not require the example action atlas or knight art.

Shipped artwork	Source / replacement workflow
Shared button, header, window, combat panel and resource art	Nine small textures under /InterfaceForge/UIForge/Themes/Art, with editable SVG/raster sources and provenance hashes in Plugins/InterfaceForge/SourceAssets/UI. Independently code-authored geometry; no competitor pixels, image-generation output or game art. DA_ForgeEmber and DA_ForgeMoonsteel work without the example project.
UIForge plugin/documentation mark	Original composed-panel SVG with solid geometry; editable source and deterministic exports under ContentSource/Brand. User-selected design.
RPG frame, slot and courtyard map	Original editable geometry and deterministic rasterization in ContentSource/UI/Emberglass. Cartography uses the example world placements and explicit bounds; no reference-game pixels or image-generation model. Cinzel is unmodified SIL OFL1.1, with its full notice in Content/ForgeExamples/Licenses and source font folder.
Six action icons	Original code-authored SVG lines, polygons and circles under ContentSource/UI/AbilityVectors. The 3×2 AbilityAtlas.vector.png is imported as T_IFAbilities. Replace the texture or individual Blueprint entry brushes; preserve the cell mapping if reusing the atlas.
Six army command icons	Original SVG geometry under ContentSource/UI/CommandVectors, deterministically rasterized into CommandAtlas.vector.png and imported as T_IFCommands. The source example includes the vectors and provenance record under ArtSource.
CommonInput face glyphs	Original neutral vector UI material and four instances under /InterfaceForge/UIForge/Input. Replace controller data through CommonInput platform settings. No platform-brand icon license is supplied.
Progress / Sakura-style interior tint	Independently authored UI material/procedural drawing. Only inspected behavior is reproduced; no Sakura material package, graph or icon art is distributed.
Themes and feedback audio	Original theme data and short synthesized audio. Source WAVs are under the core plugin's SourceAssets/Audio; synthesis source is development tooling, not a runtime dependency.
Terrain backdrop and soil material	Original code-authored static mesh and deterministic baked numeric textures under ContentSource/World. Editable OBJ,512px albedo/normal and provenance records ship with the separate example. One shared12,800-triangle backdrop, two texture samplers, no image-generation model or new third-party art.
Castle, knight, blade and portrait	Bounded AshenForge-owned MassForge art imported into the separate example project, with remapped dependencies. The core contains none of these art packages. The portrait and minimaps are Unreal scene captures/bakes.

Shipped artwork	Source / replacement workflow
HumanMotion_v022 performances	Seven infantry skeletal clips selected from MassForge's edited/retargeted Quaternius CC0 Standard Universal Animation Library performances. The

Shipped artwork	Source / replacement workflow
	separate example retains the creator/source/license acknowledgement and provenance in ArtSource/Animation. No claim of exclusive ownership of the source motions. See example animation.

The replaced **AbilityAtlas.png** was produced with an image-generation tool. Its source and original provenance remain intact in private Maintenance/ArtReview/GeneratedAbilityAtlas. Earlier engine captures containing that atlas are retained privately in Maintenance/ArtReview/PreviousGallery; they are excluded from the intended current distribution and product media. The new icon assets use the geometric sources above; they are not described as human hand-drawn or as proof about every asset in the product.

Game Master's UI design remains exclusive. Its widget trees, style assets, materials, source and reference screenshots are not included in distributable examples or the core. Inventory mechanics and external project code are outside this product.

For customer-provided icons, fonts, audio and textures, check that their terms allow **source redistribution inside a plugin/template**. Permission to use an asset in a compiled game alone is insufficient. No third-party icon pack or new custom product license was introduced by this replacement. The final package manifest records actual included files; marketplace declarations must describe that whole deliverable and its provenance.

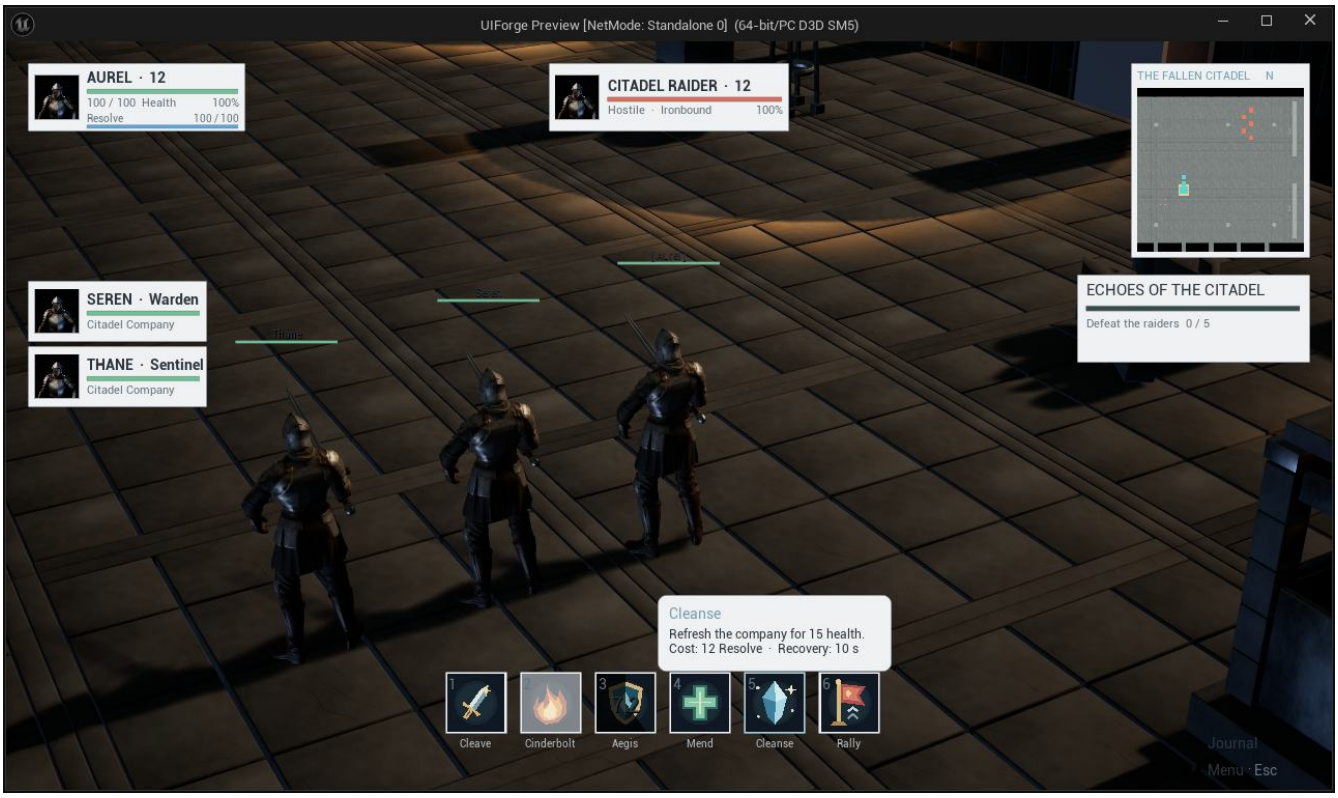
Replace an icon and review its states

The example's saved **WBP_IFAbilityEntry** is an IFReadoutEntry with an ordinary Designer Image named **IconImage**. Set the presentation model's **Icon**, then call **Notify Data Changed**. For an individual texture, set the entry's **Icon UV Min** to(0,0) and **Icon UV Max** to(1,1); atlas users choose the cell rectangle instead. The widget changes its brush without source changes or synchronous texture loading: keep the chosen texture resident through the game's asset references/loader.

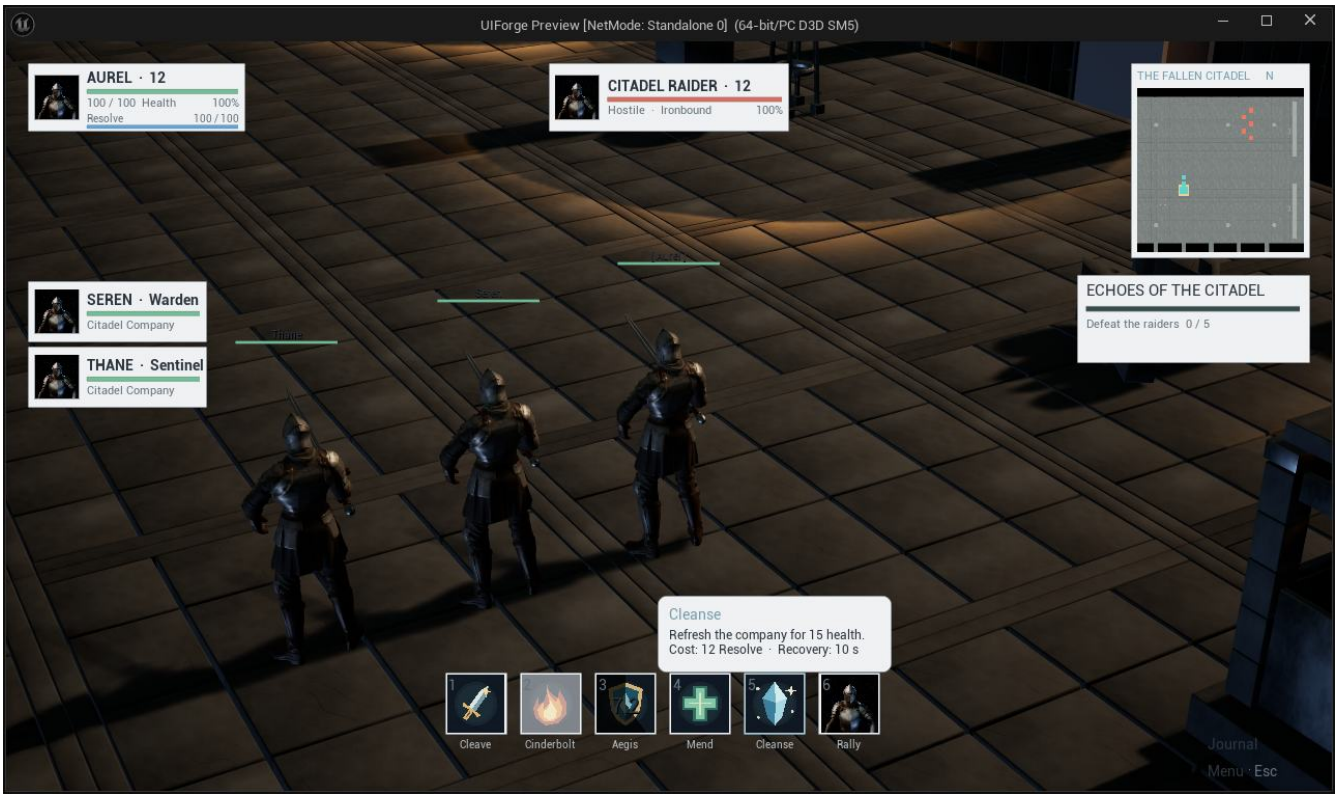
These actual UMG renders use temporary preview data to review available/unavailable actions, entry selection, navigation focus and the center-origin cooldown overlay. They are authoring checks of the existing widgets, not an additional gameplay example. The light palette is a temporary custom theme; floating action captions retain a contrasting local **Theme Override** because their background is the game world. A global light palette alone is not a finished light HUD design.



Saved Blueprint action entries with original vector icons and dark presentation



The same Blueprint entries with a temporary public light theme and local caption overrides



Model Icon and full UV properties replace the last action image with the owned knight portrait

VERSION AND VERIFICATION

UIForge 5.8.3 product rebuild

Active objective renewed by the user on 24 October 2026. Historical acceptance records describe the rejected baseline, not acceptance of this rebuild.

Architecture and boundaries

- Retain CommonUI activation, modal stacks, focus restoration and shared CommonInput intent. Passive HUD regions remain ordinary UMG; world-space widgets do not become activatable screens.
- Add public Blueprint layout profiles with named HUD regions, compact semantic typography and reusable presentation controls. Examples must use these public contracts and saved editable Widget Blueprints.
- Separate a small local gameplay/save adapter from InterfaceForge. MMO is a single-player MMO-style encounter, not an online service. RTS is a small command/defense scenario. No inventory rules, crafting, trading or replication.
- Use the finished AshenForge-owned MassForge castle/character art through a minimal independently loadable sample-art dependency closure, recording source and destination hashes. Never modify the reference plugin or include its gameplay SDK in core.
- Preserve Sakura's inspected center-origin opening; improve surrounding icon/frame/feedback presentation without changing the mask into perimeter progress.

Focused feature matrix

Foundation	Decision / developer problem	Demonstration	Cost
CommonUI stacks and intent	Retain; avoid duplicate routing	Hub, menu, pause, settings, modal and custom screen	Existing + integration
HUD layout profiles	Add; fix giant proportional cards and make genre composition replaceable	MMO action/party/target/objective/minimap; RTS resources/selection/commands/minimap	Medium
Semantic presentation	Extend existing theme, compact typography, state/icon controls	Two coherent original treatments; Blueprint replacements	Medium
Menu/save flow	Example adapter owns saves, core presents actions and dialogs	Continue/New/Load/Settings/Exit; pause Save/Load/Return with real results	Medium
MMO local encounter	Small example-only backend; UI receives observable data	Move, target, cast/cooldown, party/status, quest, feedback, victory	Medium
RTS local encounter	Small example-only backend; preserve UI independence	Select/multi-select, move/attack, train, resources, queue, victory	Medium
HUD customization	Public named regions, sizes, class/data replacement	Saved Blueprint variant and layout/theme switching	Medium
Performance	Event-driven data, shared clocks and bounded rendering	Measured source/cooked dense HUD cost; no off-thread UObject promises	Medium

Milestones and acceptance

1. **Foundation and art:** compile/load actual 5.8.3, import minimal owned art closure, add public layout/presentation contracts. Inspect a composed HUD in Unreal.

2. **Complete vertical slice:** saved hub → game menu → New/Continue/Load → playable encounter → pause/settings/save → menu. Test real saved controls, modal Back, settings rollback/persistence and software device switching. Review actual captures before expanding.
3. **Two genre examples:** recognizable MMO and RTS arrangements with appropriate density, working contextual controls, selection/targeting, minimaps and observable gameplay. Saved scenery should be inspectable in the editor. No explanatory feature panels in player interfaces.
4. **Authoring and finish:** prove theme/widget/entry/data/layout replacement without core edits; correct clipping/focus/motion, representative 1080p/720p/ultrawide/enlarged text, current Blueprint compile and package checks. Present the integrated result for user feedback.

Reference decisions

- Blizzard's HUD revamp: preserve central play space, separate action/status/objective regions and expose layout customization.
  https://worldofwarcraft.blizzard.com/en-us/news/23841481/world-of-warcraft-dragonflight-hud-and-ui-revamp
- Age IV UI updates: readable unit details, grouped teams and controllable minimap size.
  https://www.ageofempires.com/news/ageiv_seasonfour_update_60878/
- Riot interface examples: compact ability group, persistent minimap and clear cooldown/availability priority. Reference conventions only; no artwork, names, fonts or proprietary layout assets are redistributed.
- Epic 5.8 CommonUI guidelines: activation for input-routing surfaces, ordinary widgets for passive content; toggle navigation modalities rather than showing competing focus.
  https://dev.epicgames.com/documentation/unreal-engine/design-guidelines-for-using-commonui-in-unreal-engine

Current risks and workspace

Execution checkpoint

The four milestones now have composed, runnable implementations. Native Editor builds, 55 Blueprint compiles, complete saved-control flows for both genres, Blueprint customization, software input edges, three viewport shapes and enlarged settings have passed. Terrain backgrounds and the portrait were baked once in the editor, with no runtime SceneCapture. A current Development package builds; actual packaged testing found and corrected the example input context's missing cook reference, numeric wrapping after scale preview, and notification placement. The backend APIs and actual public authoring workflow are in BLUEPRINT_PRODUCT_LAYOUTS.md. Current verification and remaining limits are in PRODUCT_VERIFICATION_583.md and CURRENT_PRODUCT_STATUS.md. Historical baseline acceptance is not substituted for these checks.

Authoritative project: C:/UEProjects/ashenforge-plugins/UIForge/Project. Installed UE5.8.3 CL58210709 confirmed; descriptor now selects 5.8. UIForge has no discovered Git metadata; MassForge/Project does. No history was invented/reset. The authored-file baseline, ownership receipts and compact test records remain local. Existing core and historical example packages are retained for review.

New generated outputs belong in ../Workspace or normal ignored project build folders. Use one active project/build lane and one packaging destination, no full-host/archive chains. Physical controller testing remains user-deferred; software switching is required and cannot be called hardware certification. Visual quality is reviewed separately from functional test passes. No external publication or Git push.

VERSION AND VERIFICATION

Tested engines and platforms

The active redesign targets **UE5.8.3 CL58210709** in the single authoritative UIForge project. The descriptor selects 5.8. New `/Game/ForgeExamples` assets are authored on that engine and are not claimed to load in older versions. Rebuild the plugin for your installed engine. Avoid duplicating this development project for engine matrices.

The table and receipts below belong to earlier revisions. They do not verify the newly added APIs or redesigned examples. See current status.

Installed engine	Independent core	Latest accepted scoped sample evidence
5.6.1 CL44394996	Strict Win64 Editor Development/Game Development/Shipping and isolated load; current EP Blueprint-only install	22current source/Development cases including nullable entries/fullflow/customization/cards;32cleanBPs
5.7.4 CL51494982	Same independent/install checks	Same22current cases in separate engine copy
5.8.3 CL58210709	Same independent/install checks	Same22current cases in separate engine copy; optional Mass coverage separate

Core has one runtime module and requires CommonUI/CommonInput and EnhancedInput. It loads without examples, MassForge or inventory. The current64core files include the nullable-entry cleanup correction and match fresh strict E56/E57/E58 Editor Development/Game Development/Shipping hosts:23required native types/16distinct packages and native progress regressions. UE5.7 additionally registers a generated class in the same package. Nullable entry evidence distinguishes its actual Blueprint tests from strict builds. Current compiled installation verifies fresh Blueprint-only projects on allthree exact builds.

The full current example matrix is now accepted separately: 66source/exact Development cases,32cleanBPs per engine,177authored files/55packages and exact staging/native diagnostics. It includes source/cooked nullable cleanup and existing1,000model pooling checks; IF5 has current UE5.6.1 Shipping candidate evidence; final delivery acceptance remains separate. Optional M4 current-core verification is recorded independently.

Prior Progress records60progress/24complete menu/12Ward cases, and FieldCards records24card/24full-flow cases with32cleanBPs/175sourcefiles/55packages. Their core hashes predate the nullable correction and remain historical; do not present them as direct proof of that fix. The fix changes two core source files and no public function names/materials/assets.

The SafeArea evidence records exact versions, commands, archives, clipping repair, config-preservation limits, asymmetric insets, full RTS/RPG menu flows, and relevant aspect ratios. Compatibility polish records the warning fixes. Earlier full example proof, software input edges, settings/audio, large collections, and list themes retain their exact revision and fixture scopes.

The Localization L2 record adds36locale reports,24English menu regressions, actual staged en/tr resources and packaged culture restarts across these three versions, plus one supplemental native UE5.6 source focus check. UE5.6.1 engine startup StructuredLogFormat reports five errors with actual Turkish localization, reproduced with InterfaceForge disabled. Strict source-restart retains FAIL; packaged restart is independently clean. See language authoring for save-I/O/font/global-culture limits.

What the checks establish

The latest RPG context-layout correction is verified in twelve source/exact Development quest cases acrossUE5.6.1,5.7.4 and5.8.3, including actual saved controls, notification clearance and readable heading; all32BPs compile cleanly in the newer copies. The64-file core is unchanged. Final launch/layout evidence separately proves eight default-selection Shipping flows and two packaged bootstrap cases onUE5.6.1. It does not identify a Shipping RHI or certify other hardware.

NewLabels evidence verifies24source/exact Development-package cases across these engines on the177-file/55-package revision, with271translated catalog entries and32cleanBlueprints per engine. Runtime Turkish activation follows English startup; it does not close the UE5.6 Turkish-editor-startup limitation. The sample README was subsequently updated without changing runtime code/assets; frozen evidence remains authoritative for its recorded revision.

Own hidden Win64 DX11 engine/game processes render real UMG and execute synthetic Slate interactions. Saved Blueprint screens, custom entries, public themes/data sources, dialogs, settings, remapped prompts, input transitions, model lifetime, world entries, and menu/HUD return behavior have scoped runtime evidence. Normal/enlarged, ultrawide, 4:3, and portrait-sized Win64 viewports have corresponding records. Audio evidence measures engine mixer signals; it does not certify physical audibility.

Physical controller tests are deferred by the user. Software gamepad/touch events and mapper notifications do not certify controller hardware, native disconnection, touch panels, or real stick drift. Portrait Win64 is not mobile verification. Console, Android, iOS, macOS, Linux, CJK/RTL, and device safe-area behavior remain unverified. The local IF1 candidate additionally has a full UE5.6.1 Win64 Shipping build/cook/stage/archive (65.31 seconds, 5,459 UFS staging targets, 30 clean Blueprint classes). The later G revision additionally has24actual Shipping primary/reload reports acrossallthreeengines with public menu world travel, glyph/cursor/modal/focus, action/tint and UIsettings persistence. These records are scoped to frozen177-source/55packages before the localization refresh; physical input, ShippingPCM/remapping/defaults and final release acceptance remain separate. Split-screen and multiplayer claims require their own runtime acceptance; local-player ownership alone is insufficient.

No engine compatibility wider than these installed versions is claimed. Do not overwrite or upgrade Game Master, Sakura Serenade, or MassForge to validate the plugin. Game Master design remains exclusive; Sakura's requested cooldown has independent original implementation/provenance.

The corrected Review2 archives preserve EnabledByDefault=false. Blueprint-only packaging proves UE5.6.1 Win64 Development packaging/module load with no authored game C++; Unreal generates a temporary native target and requires Windows C++ tools. Editor loads pass allthree exact builds; newer/Shipping Blueprint-only packaging is not claimed.

VERSION AND VERIFICATION

Shipping example verification

The actual Win64 Shipping example has an explicitly opt-in review route. Normal play does not spawn the review actor. The review operates the same saved Blueprint screens and public provider actions used by players; it does not substitute a test-only menu or HUD.

Shipping logging keeps its normal engine configuration. The fixture writes a UTF-8 sidecar under Unreal's example user save directory. It records the compiled configuration, engine version, executable path, PID, destination map, actual viewport, individual checks and terminal result. The external verifier requires the actual Shipping executable and checks the declared genre against the loaded map. A process exit or static frame alone is insufficient.

What the current records cover

The SI6F revision has four primary flows and four clean-process reloads on UE5.6.1 CL44394996: RTS/RPG at 1280×720 and 800×600 with enlarged UI/text. Checks cover saved settings Cancel/Apply, isolated SaveGame persistence, independent UI/game volume values, controller glyphs and cursor/input presentation, actual Patrol/Ward responses, center-tint material, backend pause/resume, custom Blueprint cards, dialogs/focus and return to main.

The actual slider graph is invoked through its value-change event. This verifies the saved Blueprint binding and resulting setting, not physical slider manipulation. Audio-value checks do not replace the separate mixer PCM tests. Remapping, physical controllers/touch, OS input, other platforms and broader Shipping feature coverage retain their own verification gates.

SI6G adds a settled-frame primary-label geometry check and player-menu travel in both directions. Its eightactualUE5.6 Shipping primary/reload cases nowpass, with12menu-travel trails andpositive settledlabelgeometry. Reviewednormaland enlargedframesare readable; earlierclipping-lookingcaptureis retainedwithoutclaimingitscause. SI7G/SI8G now have the same eight scoped actual Shipping primary/reload passes on UE5.7.4 CL51494982 and UE5.8.3 CL58210709. Shipping-G-Matrix pins24cases/36menu-travel trails across the three frozen177-source/55-package revisions. Each engine compiles32Blueprints cleanly. This evidence predates the new-label localization refresh.

Review data and reproducibility

The review uses uniquely named `IFShipQA-` profiles. A primary run refuses to overwrite an existing profile; its separate reload requires that profile to exist. Existing terminal records are preserved and can only be resumed after strict verification. Shipping uses the engine user save directory, which differs from the Development archive's Saved directory. The launcher copies owned records, frames and profile snapshots into internal Evidence for review.

Run `Tools/package_shipping_review.ps1` with a fresh prepared host tag and its engine version. It serializes build/cook/archive, immediately captures UAT receipts, compiles all saved Blueprints, verifies distribution boundaries and runs primary/reload cases through `Tools/run_shipping_reviews.ps1`. `Tools/verify_shipping_review.py` verifies sidecars against the exact executable. These tools belong to the development workspace, not the normal buyer authoring workflow.

The real main-menu **Visit Emberwatch / Visit Ashen Reach** actions select the worlds. The tested Shipping executable ignored a command-line map override; the rejected record identified its actual RTS map instead of claiming an RPG success. See world navigation for the public Blueprint/provider route and root cleanup around Open Level.

Current records are development evidence, not a declaration of full product completion or platform certification. Failed source/launcher/map cases and prior artifacts are retained separately. Final delivery excludes internal Evidence, user save data, reference captures and private SDK validation archives.

VERSION AND VERIFICATION

InterfaceForge feature matrix

This focused release supplies reusable UI foundations and real Outpost RTS/Emberwatch RPG interfaces through public Blueprint features. Costs S/M/L describe relative implementation effort. The acceptance record links exact revision-scoped proof; builds alone do not establish usability. Historical milestone records remain preserved separately.

Game Master's visual design stays exclusive. Sakura's requested center-origin cooldown is independently reproduced from recorded behavior. Inventory mechanics are excluded. CommonUI/CommonInput/EnhancedInput remain the engine foundations; InterfaceForge adds per-player presentation, lifecycle and convenient Blueprint contracts.

Feature	Decision / developer problem solved	Dependencies / cost	Included example and verification
Screens, layers and dialogs	Extend CommonUI activatable stacks; remove game-specific controller casts and the single global layout	CommonUI / L	Main → settings → HUD → pause → return; modal replies, Back, loading and focus restoration. Per-player ownership is implemented; split-screen runtime is unverified.
Input presentation	Centralize meaningful device intent instead of repeating handlers in each screen	CommonInput + Slate / L	Saved menus, HUD, popup and custom screen share pointer/keyboard/gamepad/touch presentation. Software drift/jitter/repeats/warp/disconnect/resume tests and actual Slate touch activation pass. Physical controller is user-deferred.
Prompts and remapping	Query applied EnhancedInput mappings and use CommonUI action rows, with configurable glyphs and readable fallback	EnhancedInput + CommonInput / M	Actual keyboard/controller stage, Cancel, Apply and live prompt updates; custom Blueprint prompt body, modal and focus checks. No third-party glyph pack.
Blueprint screens and entries	Hooks/interfaces handle initialization, observable refresh, selection, recycling and release without one C++ subclass per variation	UMG / L	Saved Watchpost screen, entry and entirely Blueprint source; theme/widget/source replacement. None-data cleanup, repeated release and stale-model isolation are tested. Recipes.
Lists and tiles	Keep UMG virtualization and pooled entries; separate data objects from realized views	ListView/TileView / M	Playable journal and field cards, search/selection/model changes;1,000/10,000-model tests retain a bounded realized widget pool. Whole-scene timings are not entity-simulation benchmarks.
HUD composition	Bind game-owned models/actions through a project-neutral interface	IFDataSource/IFPresentationData / L	Outpost selection/multi-selection, commands, resources, production and cleanup; Emberwatch party, status, interaction, quest and action recovery. Small example gameplay providers stay outside the plugin.
Themes and local overrides	Share semantic typography/color/spacing/state/sound tokens; retain per-widget overrides	UMG/Slate + theme assets / M	Live theme changes reach screens, entries, pooled lists and world views. Scroll components and the list-theme node use the same tokens. Custom root and widget classes remain replaceable.

Feature	Decision / developer problem solved	Dependencies / cost	Included example and verification
Feedback and transitions	Combine engine stack transitions with ordinary editable UMG timelines; honor Reduced Motion	CommonUI/UMG + audio routing / M	Two saved concurrent animations work while actual world time is paused; activation, Back, reopen, live Reduced Motion and cleanup are tested. Arbitrary custom exit choreography is an extension point.
Contextual descriptions	Share IFEntry bodies between pointer MenuAnchor popups and a reserved navigation-hint lane	UMG + local-player tooltip service / M	Actual pointer → keyboard return, live source refresh, delay during pause, modal/class/release/rebind checks. Hints yield to action results without covering focused cooldowns.
Result notifications	Bounded passive queue with weak scopes, coalescing and explicit lifecycle	Local-player subsystem / M	Patrol/Ward/quest results; FIFO, overflow, GUID, custom host/body, modal reading-time suspension and cleanup. Full quest route caught and corrected enlarged ribbon/context overlap.
Loading states	Retain StreamableManager soft-class loading; expose replaceable body, failure/retry and cancellation	Engine async loading / M	Settings/Begin requests, competing requests, fallback and scope/root cleanup; omitted-cook failure was reproduced and repaired. Map travel uses Remove Root → Open Level; MoviePlayer startup loading is outside this release.
UI settings	Draft/preview, Apply/Cancel/defaults, validation and persistence reuse engine systems	SaveGame + EnhancedInputUserSettings / L	Saved scale/audio/remap/language controls; real save failure/retry and process reload. Independent mixer waveform analysis verifies gain/mute/rollback. Graphics remain GameUserSettings-owned; not a separate graphics backend.
Responsive/accessibility	Keep engine DPI separate from player scale; support safe areas, text scale, scrolling and reduced motion	SafeZone + UMG / M	Normal/enlarged, ultrawide,4:3 and portrait-sized Win64 flows; long labels and essential-control bounds checked. Physical device safe areas/mobile are unverified.
Localization	Keep FText and engine culture nodes; provide original en/tr resources and transactional controls	Unreal localization / M	271-entry resources, culture preview/Cancel/Apply/defaults and packaged restart. UE5.6 Turkish editor startup has an engine-reproduced limitation. CJK/RTL/font coverage beyond the supplied languages is unverified.
Sakura cooldown	Preserve actual center-origin angular tint, layering and timing over original icons	Independently authored material/widget / M	Original widget/material runtime tracing and comparison; real Ward/Kindle recovery, countdown, pause and refresh. Parameters and controls; no reference asset redistribution.
Generic progress	One configurable visual contract instead of ability-specific subclasses	Original Slate presentation / M	Five public styles, direction/origin/size and charge normalization; saved Blueprint customization, clock/pause/release/theme tests. Progress recipe.

Feature	Decision / developer problem solved	Dependencies / cost	Included example and verification
World-space UI	Keep WidgetComponent and caller-owned connection geometry; bind weak actors through public entry contracts	Engine WidgetComponent + IFWorldEntryLibrary / M	Actual rendered endpoints, model/theme refresh, invalid/tree/ range/pause/clear/anchor lifetime tests. Planar example connections; occlusion and large actor populations are not certified.
Optional MassForge	Isolate actual entity APIs in a separate adapter plugin	InterfaceForgeMass + separately installed MassForge0.2 / M	Current M4 real entity lifetime/serial/coalescing and eight source/cooked Blueprint cases on UE5.8.3. Equivalent standalone provider; core/default sample load without SDK. No SDK distribution or multiplayer claim.
Local documentation	Maintain the user-requested AshenForge style with Docs as the authored source	VitePress/Mermaid / M	40guides,29rendered workflows, search, dark/light and responsive navigation. Strict generated link/image checks; stale generated Markdown removed. Local only.

The latest one-asset RPG layout correction passes twelve source/exact Development quest cases acrossUE5.6.1,5.7.4 and5.8.3. Final launch/layout checks add default-selection Shipping and packaged bootstrap proof. IF7 pins source/Shipping delivery and reviewed frames; documentation-only snapshots inherit its identical runtime. The unchanged64-file core is independently built/loaded on all three engines. Compatibility distinguishes core, sample, package and device evidence; acceptance states the completed included workflows and unverified boundaries.

Deliberate boundaries: inventory, equipment rules, crafting/trading and inventory replication are separate products. Hierarchical Tree authoring remains an engine extension, not an included complete example. Split-screen, multiplayer, non-Windows SDK/device certification and arbitrary localization are unverified. These limits do not hide pending implementation in the included menu/HUD workflows; the acceptance record tracks their actual completion.

VERSION AND VERIFICATION

Local acceptance and distribution boundaries

The current delivery is a **UE5.8.3 / Win64 local review candidate**, with the tested scope in verification. It is not a Fab approval, published listing, all-platform certificate or networking promise.

The bounded local Release folder contains two separate, CRC/member/hash-audited source ZIPs:

- **UIForge-Core-583-Source.zip**: one InterfaceForge plugin root, one UIForge content pack, C++ source, Config, Resources, curated guides/real images and original source WAVs. No game example, SDK, capture helper, website tooling, generated build/cache or private evidence.
- **UIForge-Examples-583-Source.zip**: one UIForge project, current editable UMG/settings/tutorial assets, separate local example runtime, core plugin and editable vector artwork. The editor authoring/capture module and scripts are omitted; an installation of this exact source builds and cooks the examples. No engine installation is copied.

Current archive path limits are **70** characters for the core and **168** for the example, measured from their overarching folder. Every distributed source/build file carries the publisher/year notice. The core declares Win64 module/platform support and compatible-engine metadata. FilterPlugin explicitly includes the intended guides and source audio. No custom product license, external submission, listing edit or invented Fab URL was added.

The separate example retains its established development content roots. **Do not submit it as a single-pack Fab content product without migration/review**. The core's consolidated content-pack audit passes. The current Fab requirements and Epic file-structure guidance were rechecked on2026-10-25 during preparation (the technical-requirements page reports an update on2026-09-21); the marketplace makes its own review decision. No DirectX renderer override was introduced to imitate a recommendation in MassForge's report.

The image-generated six-icon atlas and its provenance remain intact in private review, alongside earlier affected product captures. Current icons are independently constructed editable geometry, with honest artwork provenance. Replacing those icons does not, by itself, determine the whole product's marketplace AI declaration. Publisher disclosure must describe the actual deliverable; no classification or licensing change is made here.

Game Master's visual design remains exclusive. Sakura's inspected center-origin tint is independently reproduced, with its source assets excluded. Bounded owned MassForge art appears only in the separate examples; the core requires no MassForge or inventory plugin. Optional adapter source remains in the development checkout and is outside these audited candidates.

Deferred physical-controller tests remain an explicit limitation, as requested by the user. Unverified platforms/devices, display-mode confirmation, networking and production pathfinding are not advertised as completed features.

API REFERENCE

API and result handling

Use the public Blueprint contracts below to compose your interface. Their detailed recipes include node names, setup and expected behavior. The code and Blueprint scheme lists the implementation and reflected declarations.

Find an API by task

Task	Public entry point	Guide
Create the UI and open screens	IFLocalPlayerSubsystem → Create Root; IFRootLayout → Push Screen	Game setup
Compose settings pages and controls	IFSettingsMenu, IFSettingDefinition, IFSettingRow	Tabbed settings
Replace presentation and entries	IFTheme, IFScreen, reusable entry contracts	Blueprint customization
Present input-aware prompts	IFActionPrompt; IFRemappingSubsystem	Controls, glyphs
Show dialogs and loading	IFDialogData; Push Screen Async	Dialogs, loading
Read gameplay data and request actions	Get UI Model (Message); Request UI Action (Message)	Presentation sources
Post feedback	IFNotificationSubsystem → Post Notification	Notifications

Handle results explicitly

UIForge does not use one universal error enum. Check the actual return value or completion pin of each operation. Keep validation and gameplay authority in your game or adapter.

Operation	Result	Expected handling
IFSettingsSubsystem → Apply	Boolean and Failure Reason	On false, retain the editing session and show the reason. Do not show a saved message.
IFRemappingSubsystem → Begin Edit, Stage Key Mapping, Validate Pending, Apply	Boolean and Failure Reason	Branch on success. Present rejected keys/conflicts; only report applied after Apply succeeds.
IFRemappingSubsystem → Get Registered Mapping	Boolean and Mapping	Use the Mapping only when found. Register the Enhanced Input player-mappable context first.
IFDialogData → On Completed	EIFDialogResult: Confirmed / Cancelled	Perform the caller's action only on Confirmed. Back resolves as Cancelled.
Push Screen Async	Ready / Failed / Cancelled, with Screen and Reason	Use Screen on Ready; show failure feedback on Failed; treat Cancelled as a normal end.
Get Load State	EIFScreenLoadState: Loading / Ready / Failed / Cancelled	Use for state presentation. A cancelled request prevents its late screen push.
Request UI Action (Message)	Accepted Boolean	Present refusal as normal feedback. Acceptance is not a promise that asynchronous gameplay has finished.

For a settings menu, let **IFSettingsMenu** own the combined editing workflow. Do not start competing transactions from every row. Custom providers must implement validation, preview, persistence and restoration through their public contract; settings composition explains rollback limits.

Verification and compatibility

See current product status and current verification for checks actually performed and remaining limits. An API declaration does not certify a platform, physical controller, multiplayer mode or engine version.