Input Fundamentals

Learn how standard UI input systems relate to CommonUI.

CommonUI is an extension of the Slate/UMG framework. CommonUI implements a method for routing input, but it still relies on the underlying logic of Slate's existing input systems.

Each section of this guide contains a few tips or methods that you can use to modify how various parts of CommonUI interact with the base Slate/UMG input system.

Change Your Application's UI Input Handling with Input Configs

Sometimes you may want to change the way your application handles input based on which widget is currently active. For instance, you may want to prevent the player from moving in your game world when a social sidebar or pause menu is open. To handle this, **CommonUI** supports optional **Input Configs** for **Activatable Widgets**.

You are not required to use Input Configs in your application, and you can take advantage of CommonUI's other features regardless of whether you use them.

Using Input Configs in Your Widgets

Input Configs are represented with the `FUIInputConfig` struct found in `UIActionBindingHandle.h`. Each Input Config tracks the state of multiple input methods, including mouse capture options, handling for move and look axes, and the overall input mode CommonUI uses.

When you activate an Activatable Widget, it getsan Input Config using `UCommonActivatableWidget::GetDesiredInputConfig`. This function returns a null Input Config by default, but you can override it with any logic you want to use. Whenever the function returns a null Input Config, CommonUI falls back to the last valid Input Config it used.

By default, CommonUI will apply a default Input Config as a fallback if there isn't one specified by any Activatable Widgets. However, you can disable this behavior using the `bEnableDefaultInputConfig` variable located in the `UCommonInputSettings` class.

When a widget deactivates, CommonUI restores the previous Input Config it used so as to avoid getting stuck without suitable Input Config options to support the current widget. You can find this implementation logic in `FActivatableTreeRoot::ApplyLeafmostNodeConfig` function.

If you deactivate all the widgets in your UI, CommonUI will default to the Input Config for the last widget that was deactivated. If you have a use-case where you need to deactivate every widget in your UI, make sure the last deactivated widget re-applies a reasonable input handling state to avoid a soft-lock.

Recommended Use

If you are using Input Configs, **you should avoid using standard input configuration methods in your UI.** The default implementation for the virtual function

`UCommonUIActionRouterBase::ApplyInputConfig` calls the following standard UE configuration methods as part of the setup process:

- `APlayerController::SetIgnoreMoveInput`
- `UGameViewportClient::SetMouseCaptureMode`
- `UGameViewportClient::SetHideCursorDuringCapture`

Because of this, mixing CommonUI's Input Configs with other calls to these functions may result in them overriding each other, creating confusion when managing your input states.

To simplify managing your Input Configs , you can create a default implementation that assigns commonly-used Input Configs based on an enum value in your widget. For an example of this, refer to the Lyra sample project. This provides a useful implementation for applications that only need a few fixed, non-dynamic Input Configs per widget.

Input Handling State Reference

`FUIInputConfig` tracks a bundle of multiple input states. Once you set an Input Config in `UCommonActivatableWidget::GetDesiredInputConfig`, you should have a complete configuration for how you want input to work when the widget is focused. These states are tracked using the following variables:

Parameter	Type	Description
`InputMode`	Enum / `ECommonInputMode`	Set CommonUI's internal input mode.
`MouseCaptureMode`	Enum / `EMouseCaptureMode`	Sets CommonUI's mouse capture mode.
`bHideCursorDuringViewportCapture`	Bool	If true, the viewport will hide the mouse cursor during mouse capture.

Parameter	Type	Description
`bIgnoreMoveInput`	Bool	If true, the player controller will ignore movement inputs.
`bIgnoreLookInput`	Bool	If true, the player controller will ignore look inputs.

The following table summarizes the modes available for configuring your InputMode (`ECommonInputMode`):

Input Mode	Description
Menu	Input is received by the UI only.
Game	Input is received by the game only.
All	Input is received by both the UI and the game.

The following table summarizes the modes available for configuring your MouseCaptureMode (`EMouseCaptureMode`):

Mouse Capture Mode	Description
No Capture	Do not capture the mouse at all.
CapturePermanently	Capture the mouse permanently when the viewport is clicked, and consume the initial mouse down that caused the capture so that it isn't processed by player input.
CapturePermanently_IncludingInitialMouseDown	As CapturePermanently, except that player input will process the mouse down that caused the capture.
CaptureDuringMouseDown	Capture the mouse when a mouse button is down, then release on mouse button up.

Mouse Capture Mode	Description
CaptureDuringRightMouseDown	Capture only when the right mouse button is down, not any of the other mouse buttons.

Use FReply to Change How Widgets Respond to Input

`FReply` tracks the handled/unhandled status for input events. Most input handlers in Slate will either return a result of `FReply::Handled` or `FReply::Unhandled.

- `FReply::Handled` indicates that an input generally **should not be forwarded** to other widgets or input systems.
- `FReply::Unhandled` indicates that even if an input was used in some way, it **should still be forwarded** to other widgets or input systems for additional processing.

The following are some commonly-used `SWidget` input events:

- `FReply OnKeyUp(const FGeometry& MyGeometry, const FKeyEvent& InKeyEvent);`
- `FReply OnAnalogValueChanged(const FGeometry& MyGeometry, const FAnalogInputEvent& InAnalogInputEvent);`
- `FReply OnMouseMove(const FGeometry& MyGeometry, const FPointerEvent& MouseEvent);`
- `void OnMouseEnter(const FGeometry& MyGeometry, const FPointerEvent& MouseEvent);`

Many of these functions (but not all of them) return a `FReply`. These replies can be set or overridden in Blueprints, so if you need to stop or allow the processing of a certain type of input, you can try returning a certain `FReply` to get the result you want. However, most of the time the default `FReply` results should be sufficient for a well-designed widget or set of widgets. Dealing with custom `FReply`s is more of an issue when working with widgets in Slate.

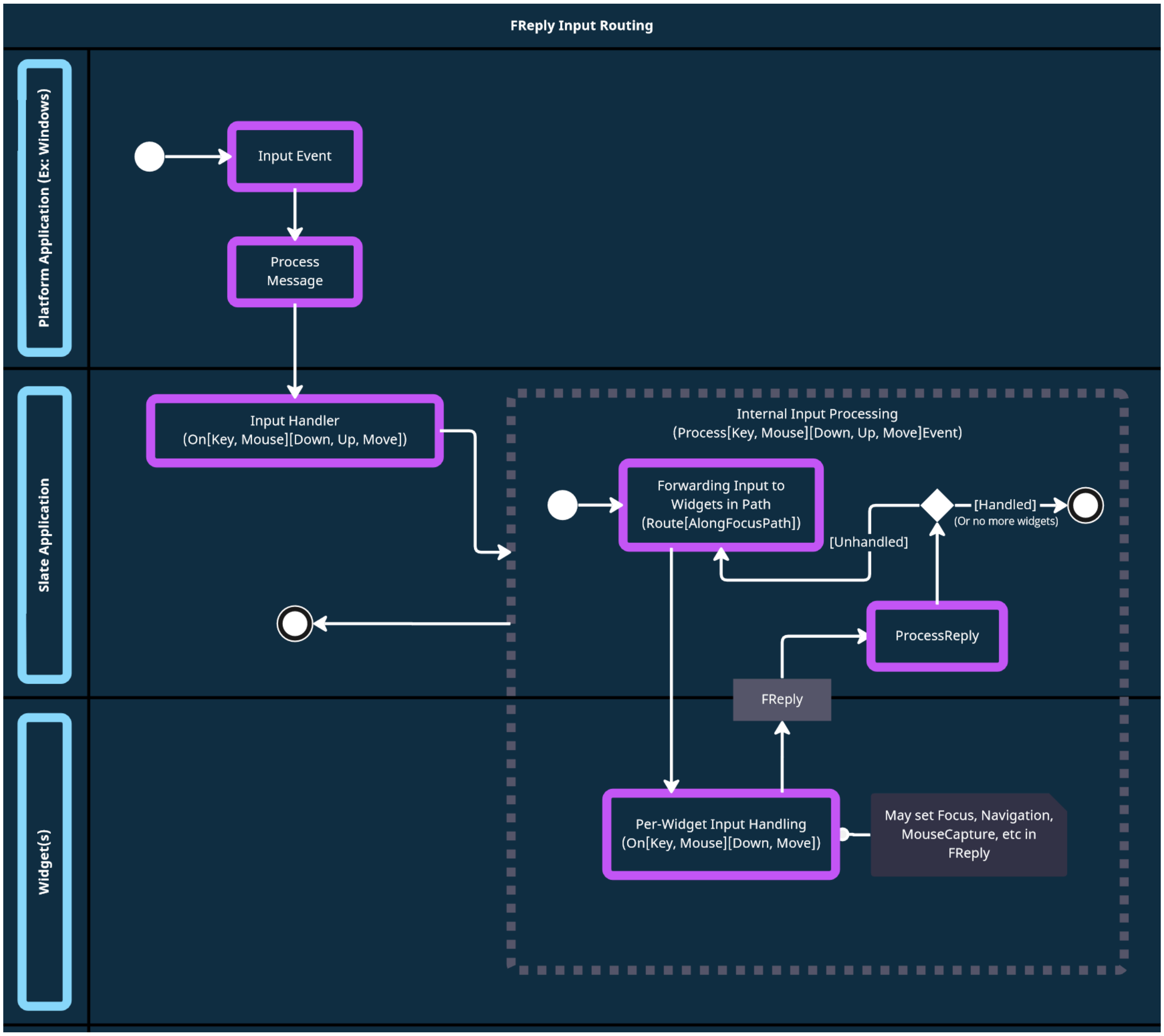

FReply Input Routing
Platform Application (Ex: Windows)
Slate Application
Widget(s)
Input Event
Process Message
Input Handler (On[Key, Mouse][Down, Up, Move])
Internal Input Processing (Process[Key, Mouse][Down, Up, Move]Event)
Forwarding Input to Widgets in Path (Route[AlongFocusPath])
[Handled]
(Or no more widgets)
[Unhandled]
ProcessReply
FReply
Per-Widget Input Handling (On[Key, Mouse][Down, Move])
May set Focus, Navigation, MouseCapture, etc in FReply

Chart illustrating the flow of FReply Input Routing. Input starts with the platform's own input event, which then forwards to the Slate Application. The Slate Application then sends it to to widgets, which use FReply to determine whether the input is Handled or Unhandled. This will repeat until either the input is Handled or until all widgets are checked.

FReply Settings

`FReply` tracks the handled/unhandled status for input events, but you can also track additional data within an `FReply`, such as with the following members:

Parameter	Description
CaptureMouse	Ask the system to forward all mouse events to a specific widget.
ClearUserFocus	Ask the system to clear user focus.
ReleaseMouseCapture	Ask the system to release mouse capture.
SetUserFocus	Ask the system to set users' focus to the provided widget.
SetNavigation	Ask the system to attempt a navigation to the specified destination

The list above is not an exhaustive list, as it is only intended to demonstrate what kinds of methods you should expect to see. See the Official C++ API for `FReply` for a complete listing.

Some of these events, such as `FReply::CaptureMouse` and `FReply::SetUserFocus`, take additional arguments, including target widgets.

To those familiar with UMG or Slate, these methods may look familiar. However, they are in the `FReply` namespace, which means that you can modify the behavior that occurs when Slate processes your `FReply`. Calling these methods in a `FReply` can yield a slightly different behavior that may not be easily replicated by calling equivalent methods outside of a `FReply`.

When Would You Set an FReply?

As an example of when to use `FReply`s, imagine that you need to set or clear widget focus on a key press. Normally, you maytry to alter widget focus by directly calling the relevant functions on the `FSlateApplication` in the keypress handler.

This approach may not work in all scenarios, especially when using Input Routing, as you are attempting to change or clear focus **while input is still being processed on the current widget**. This input flow can lead to undesired behaviors.

Instead, we reccomend you use Slate to process the input completely, then handle changes like these using the input event reply or `FReply`.

> Originally, states controlled or exposed in the `FReply` API were meant to only be set using `FReply`. This was changed when it proved too restrictive, therefore this workflow is more of a recommended guideline. However, we strongly recommend it, as this is the preferred workflow.

Customize Navigation in Your UI

This section provides guidelines and options for customizing navigation in CommonUI.

Navigation Configs

> Navigation configs are not directly related to CommonUI, but understanding them helps to understand input handling.

Slate supports cardinal navigation regardless of whether the CommonUI is enabled. Using **Navigation Configs**, or `FNavigationConfig`, determines which keys map to the cardinal directions:

- Left
- Right
- Up
- Down
- Next
- Previous

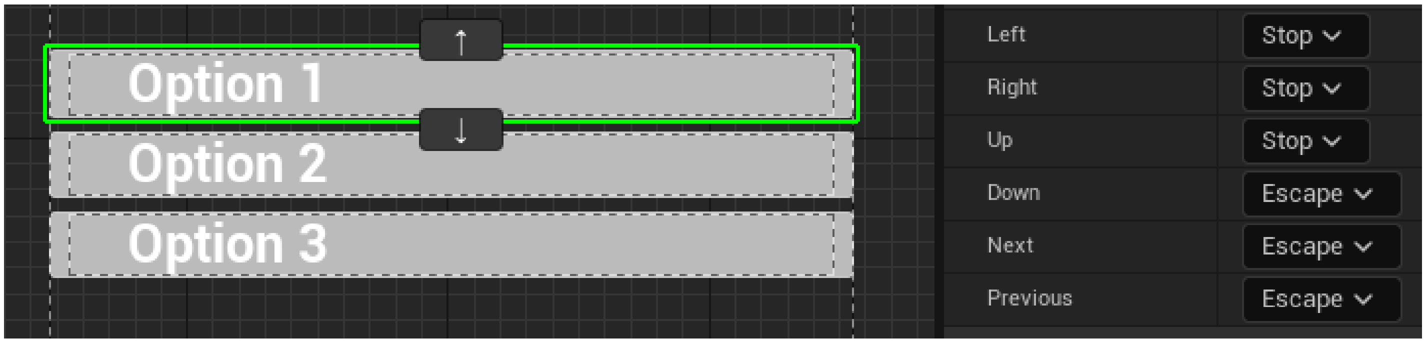
> Manual navigation configuration isn't required for Slate to use cardinal navigation.

To set a Navigation Config, call `FSlateApplication::SetNavigationConfig`. Usually, you would call this function using a custom navigation config that derives from `FNavigationConfig`. As an example, if you wanted users to interact with your UI with WASD keys, this would be the ideal place to start.

You can also set navigation configs on a per-user basis by calling `FSlateUser::SetUserNavigationConfig`.

Manually Control Navigation

To manually set what will happen when a navigation event occurs, select a widget in UMG, then locate the **Navigation** section of the **Details panel**. This section contains options for each of the cardinal directions.



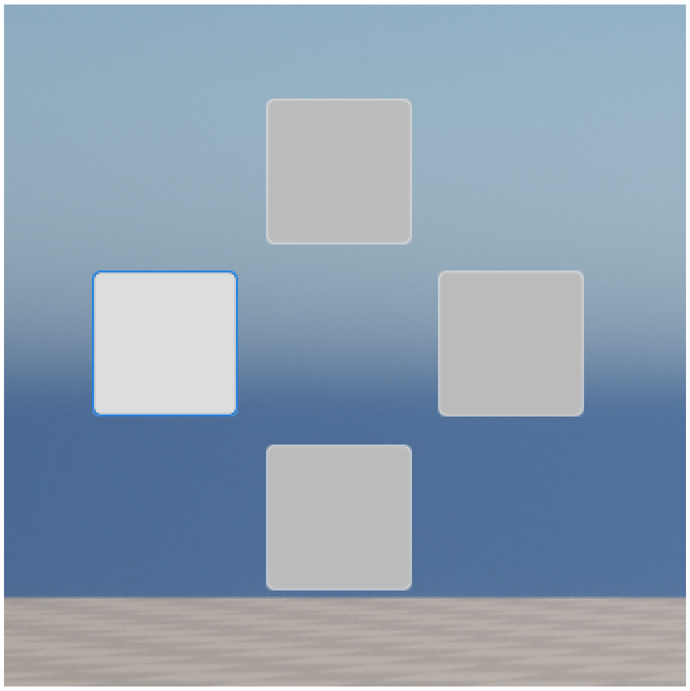
Left: Stop
Right: Stop
Up: Stop
Down: Escape
Next: Escape
Previous: Escape

The options are detailed in the table below:

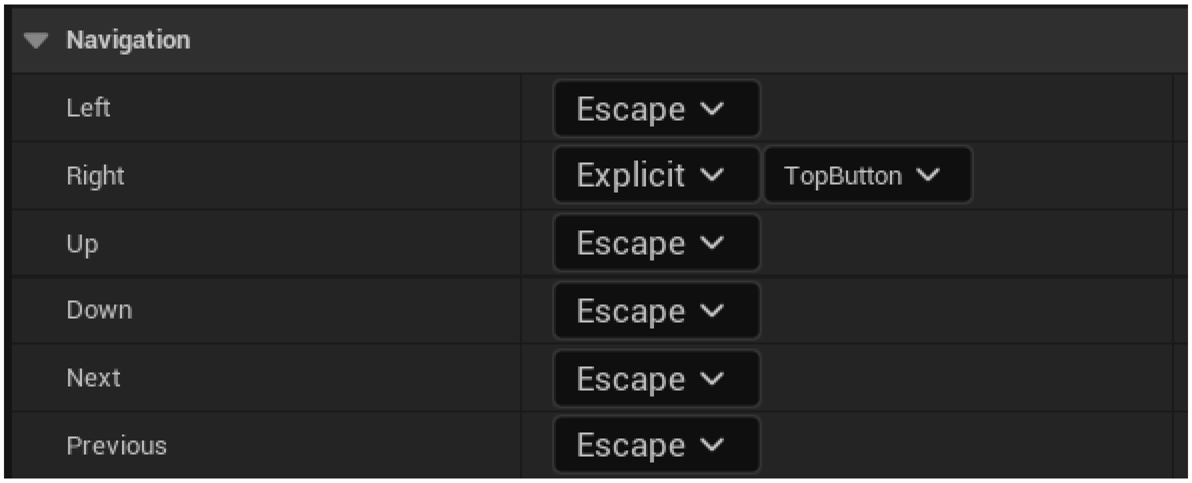
Navigation Control Option	Description
Escape	Allow the movement to continue in that direction, seeking the next navigable widget automatically.
Explicit	Move to a specific widget.
Wrap	Wrap movement inside this container, causing the movement to cycle around from the opposite side, if the navigation attempt would have escaped.
Stop	Stops movement in this direction.

Navigation Control Option	Description
Custom	Custom navigation handled by user code.
CustomBoundary	Custom navigation handled by user code if the boundary is hit.

For example, the following is a use-case where **Explicit** might be a useful option:



In this example, the top and bottom buttons do not overlap vertically with the focused button on the left. Because there is no overlap, if we navigate Right, Unreal will focus on the button to the far-right furthest away. If we want Right to navigate to the top button, we can configure the Navigation settings to do so.



Navigation	
Left	Escape
Right	Explicit · TopButton
Up	Escape
Down	Escape
Next	Escape
Previous	Escape

By setting an Explicit navigation to the **TopButton** widget, whenever the user presses Right, that widget will be focused instead.

To set a widget as an Explicit navigation target it must be manually named. This ensures long term maintainability of navigation behaviors.

Activatable Widgets and Action Bindings

This section provides information on how to customize how Activatable Widgets and bound Input Actions behave for your UI.

Set the Focus for Activatable Widgets on Activation

Whenever you activate an Activatable Widget, it calls the `UCommonActivatableWidget::GetDesiredFocusTarget` function to choose which widget CommonUI should focus the user's input on.

If you do not implement a custom version of `GetDesiredFocusTarget`, CommonUI may have difficulty knowing where to focus as widgets activate and deactivate. For this reason, **we strongly recommend that you always implement this function in your Activatable Widgets.**

In the Lyra sample project, each Activatable Widget class has a custom Enum type that determines what method to use for getting the desired focus target. We recommend a similar implementation for most menus that use fixed, non-dynamic methods of determining default focus.

Change When Your Triggering Input Action Fires

When creating a `FBindUIActionArgs` for an action binding, set `FBindUIActionArgs::KeyEvent` to the type of action that should trigger the event, for example, `IE_Released`.

CommonUI Console Variables Reference

You can use the following table of console variables to configure how CommonUI functions and obtain debugging information:

CVar	Description
CommonUI.bDumpInputTypeChangeCallstack	If true, CommonUI will dump the call stack when the input type changes. This is useful for debugging when the input type appears to change rapidly.
CommonInput.ShowKeys	Toggles whether or not to show the keys for the current input device.
CommonInput.EnableGamepadPlatformCursor	Toggles whether the cursor should be enabled during gamepad input
UseTransparentButtonStyleAsDefault	If true, the default **Button Style** for the `SButton` in **CommonButtonBase** will be set to **NoBorder**, which has a transparent background and no padding.
Mobile.EnableUITextScaling	Enables Mobile UI Text Scaling.
ActionBar.IgnoreOptOut	If true, the **Bound Action Bar** will display bindings whether or not they are configured.
CommonUI.AlwaysShowCursor	If true, CommonUI will always show the mouse cursor regardless of the current Input Config.
CommonUI.VideoPlayer.PreviewStepSize	Time step amount for CommonVideoPlayer.