Article

Handling input events

Receive controller input using either polling or callbacks.

Overview

The Game Controller framework provides your game with low-level and fast access to input from connected gamepads, arcade sticks, and racing wheels. Not all games receive and process input the same way. The Game Controller framework input APIs support several ways for your game to receive controller input:

- Use polling when input handling occurs at a fixed interval as part of your game's loop. You request the latest input state from the Game Controller framework each time you handle input. This can occur once per frame, or at a lower frequency than the frame rate.
- Use callbacks when the input handling in your game is event-driven. You provide one or more blocks that the Game Controller framework invokes when any or certain elements' input values change.

Get the input profile

You get game controller input from the device's input profile. The `GCDevicePhysicalInput` protocol declares the common properties for the input profile. The `GCControllerLiveInput` and `GCRacingWheelInput` classes provide the concrete implementations for gamepads and arcade sticks, and racing wheels, respectively.

For gamepads and arcade sticks, first get the `GCControllerLiveInput` object from the `GCController` object using the `input` property.

```
let controller: GCController
let input: GCControllerLiveInput
```

```
// Get the input profile object.
input = controller.input
```

For racing wheels, first get the `GCRacingWheelInput` object from the `GCRacingWheel` object using the `wheelInput` property.

```
let racingWheel: GCRacingWheel
let input: GCRacingWheelInput

// Get the racing wheel input profile object.
input = racingWheel.wheelInput
```

Configure the input dispatch queue

The Game Controller framework processes input events, and invokes any configured callbacks, on the `GCController` or `GCRacingWheel` `handlerQueue` property of the device. The `handlerQueue` property is the main queue by default. To configure a dedicated queue for input processing, set the `queue` property of the `GCControllerLiveInput` or `GCRacingWheelInput` object.

Access input elements

Game controllers organize their input states by elements. These elements correspond to the individual buttons, directional pads, thumbsticks, and other kinds of controls on the gamepad, arcade stick, or racing wheel. Game controllers identify elements by their semantic name, and group them into one or more collections, such as the `elements`, `buttons`, `dpads`, `axes`, and `switches` properties.

To get objects representing specific elements of a controller, use the subscript notation with the collection properties from an object conforming to the `GCDevicePhysicalInputState` protocol. This includes the `GCControllerLiveInput` and `GCRacingWheelInput` objects.

For example, to get the A button of a controller, use `.a` with the `buttons` property, as in the following example:

```
// Get the A button element.
let aButton = input.buttons[.a]
```

To get the left thumbstick of a directional pad, use `.leftThumbstick` with the `dpads` property.

```
// Get the left thumbstick element.
let leftThumbstick = input.dpads[.leftThumbstick]
```

Get the current input state

Both the `GCControllerLiveInput` and `GCRacingWheelInput` objects track the last received input state from the game controller. If you poll for input, but just need the current input values, get the elements from the `GCControllerLiveInput` and `GCRacingWheelInput` objects and read their input values.

```
let input: GCControllerLiveInput

// Read the current A button element pressed state.
let aButtonPressed = input.buttons[.a].pressedInput.isPressed
```

Reading a single input value is an atomic operation. However, the current input state can change between reading distinct input values. To avoid these inconsistencies, use the `GCDevicePhysicalInput.capture()` method to obtain a snapshot of the input state at the current moment. Then read the input values from the snapshot. Snapshots carry very little performance overhead as long as your game doesn't hold onto the snapshot for long periods of time.

```
let input: GCControllerLiveInput
let snapshot: GCControllerInputState

// Capture the current controller input state.
snapshot = input.capture()

// Read the current A and B button pressed states from the snapshot.
let aButtonPressed = snapshot.buttons[.a].pressedInput.isPressed
let bButtonPressed = snapshot.buttons[.b].pressedInput.isPressed
```

Poll for input changes

Sometimes an input device generates input more frequently than your game can process it. For example, a racing wheel with a high refresh rate generates hundreds of inputs per second. If you read the current input values every frame (at 60 frames per second), you miss input events. In this case, configure the Game Controller framework to maintain a queue of input states that you retrieve and process later.

First, change the input queue depth to a value that's appropriate for the frequency at which your game polls for input. The input queue depth specifies the maximum number of input states that the Game Controller framework buffers before it purges older input states from the queue. The default value of 1 indicates no buffering.

```
// Set the queue depth.
input.inputStateQueueDepth = 20
```

In your game loop, handle all the inputs that occur after the last iteration of your game loop using the `nextInputState()` method. Call the `nextInputState()` method until the queue is empty.

Optionally, assign a callback to the `inputStateAvailableHandler` property of the `GCControllerLiveInput` or `GCRacingWheelInput` objects. The Game Controller framework calls this block when a new input state becomes available, and not again until your game processes all input states in the queue.

```
// Set the input state available handler.
input.inputStateAvailableHandler = { (input) in
    // Either handle the queued input states,
    // or wake up your game loop to handle the input
    // states later.
}
```

Each call to the `nextInputState()` method returns a snapshot of the oldest input state in the queue. Use the snapshot to access individual elements and read their input values.

```
// Set the input handler.
input.inputStateAvailableHandler = { (input) in
    while let nextInputState = input.nextInputState() {
        // Handle the input for specific elements.
        if let buttonA = nextInputState.buttons[.a] {
            let buttonAPressed = buttonA.pressedInput.isPressed;
            if (buttonAPressed) {
                // Handle when the user presses the button.
            }
        }
    }
}
```

To determine whether an element's input changed compared to the previous input state, use the `change(for:)` method. To get a collection of all the elements with changed inputs compared to

the previous input state, use the `changedElements()` method.

```
// Handle all the input since the last game loop iteration.
while let nextInputState = input.nextInputState() {
    // Get all the elements with changed inputs.
    if let changedElements = nextInputState.changedElements() {
        for changedElement in changedElements {
            // Handle the change to the element.
        }
    }
}
```

Handle when the queue overflows. If the queue is empty, the `nextInputState()` method returns `nil`. If the queue overflows, the `changedElements()` method returns `nil` and the `change(for:)` method returns `GCDevicePhysicalInputElementChange.unknownChange`.

Receive callbacks for input changes

To receive callbacks from specific elements when their inputs change, assign a callback to one or more of the element's inputs. Then implement the handlers to take the appropriate action for your game.

For example, a button element has a pressed input representing its pressed state. Assign a block to the `pressedDidChangeHandler` property of the button's `pressedInput` property to take an action when the user presses it.

```
// Register a callback for the press/release of the A button.
if let aButton = input.buttons[.a] {
    aButton.pressedInput.pressedDidChangeHandler = { (_, _, pressed) in
        if pressed {
            // Handle when the user presses the button.
        } else {
            // Handle when the user releases the button.
        }
    }
}
```

To receive a callback when the inputs of any element change, assign a callback to the `elementValueDidChangeHandler` property of the `GCControllerLiveInput` or `GCRacingWheelInput` objects. When input occurs, the Game Controller framework calls the block once for each

element that changes. Your code reads the latest values from the element that Game Controller passes to the handler.

```
// Register a callback for any element change.
input.elementValueDidChangeHandler = { (input, element) in
    if element === input.buttons[.a] {
        // Handle A button input.
    }
}
```

See Also

Essentials

Game Controller updates

Learn about important changes to Game Controller.

`GCSupportsControllerUserInteraction`

A Boolean value indicating whether the app supports a game controller.

`GCSupportedGameControllers`

The types of game controller profiles that the app supports or requires.

`GCSupportsMultipleMicroGamepads`

A Boolean value indicating whether the physical Apple TV Remote and the Apple TV Remote app operate as separate game controllers.