Class

GCController

A representation of a real game controller, a virtual controller, or a snapshot of a controller.

iOS 7.0+ | iPadOS 7.0+ | Mac Catalyst 13.1+ | macOS 10.9+ | tvOS 9.0+ | visionOS 1.0+

```
class GCController
```

Mentioned in

- Adding touch controls to games that support game controllers in iOS
- Handling input events
- Discovering and tracking spatial game controllers and styli

Overview

This class represents a real or virtual controller that a user interacts with during a game. A *real controller* is a physical controller that connects directly or wirelessly to the device. A real controller can be formfitting or can attach closely to a device so players can use controls on both simultaneously. A *virtual controller* is a software emulation of a real controller.

You discover controllers, and then you process the input from those controllers during gameplay. Use the `controllers()` method to get the currently connected controllers. If necessary, use the `startWirelessControllerDiscovery(completionHandler:)` method to connect with wireless controllers.

This framework supports multiple connected game controllers. To identify which player is using a controller in a multiplayer game, check the `playerIndex` property and set it, if necessary. For

single-player games, use the `current` property to get the controller that the player is actively using.

A controller's profile encapsulates the details about a controller's buttons, pads, axis, and other input elements. Get the controller's profile using one of the profile properties, such as `extendedGamepad`, and then process the input from its elements.

You can either get the values of input elements on each iteration of your game loop, or set handlers to receive callbacks when those values change. For example, use the `leftThumbstick` property of the `GCExtendedGamepad` profile to get the thumbstick state. Use the `valueChangedHandler` property to set a handler that you implement to process any input values that change in the profile.

Alternatively, you can create a snapshot of a real or virtual controller using the `capture()` method. A *snapshot* is a copy of a controller at a moment in time with its current element values. Creating a snapshot may impact performance, and over time a snapshot doesn't stay current. Unlike other types of controllers, you can set the values of elements in a snapshot.

Topics

Discovering controllers

```
class func controllers() -> [GCController]
```

Returns the connected controllers for the device.

```
class func startWirelessControllerDiscovery(completionHandler: (() -> Void)?)
```

Starts searching for nearby wireless controllers.

```
class func stopWirelessControllerDiscovery()
```

Stops searching for nearby wireless controllers.

```
static let GCControllerDidConnect: NSNotification.Name
```

A notification that posts after a controller connects to the device.

```
static let GCControllerDidDisconnect: NSNotification.Name
```

A notification that posts after a controller disconnects from the device.

Handling multiple controllers

```
class var current: GCController?
```

The most recently used game controller.

`static let GCControllerDidBecomeCurrent: NSNotification.Name`

A notification that posts when a controller becomes the current controller.

`static let GCControllerDidStopBeingCurrent: NSNotification.Name`

A notification that posts when a controller stops being the current controller.

Inspecting a controller

`var isAttachedToDevice: Bool`

A Boolean value that indicates whether the controller closely integrates with the device.

`class func supportsHIDDevice(IOHIDDevice) -> Bool`

Returns a Boolean value that indicates whether the framework supports the specified human interface device.

`class var shouldMonitorBackgroundEvents: Bool`

A Boolean value that indicates whether the app needs to respond to controller events when it isn't the frontmost app.

Accessing controller input

`var input: GCControllerLiveInput`

The input profile for the controller.

`class GCControllerLiveInput`

The input profile for a controller.

`class GCControllerInputState`

A class that represents an input state for gamepads and arcade sticks.

Accessing controller profiles

`var extendedGamepad: GCExtendedGamepad?`

The extended gamepad profile.

`class GCPhysicalInputProfile`

The base class for controller profiles that support physical buttons, thumbsticks, and directional pads.

`class GCKeyboardInput`

A controller profile that uses the keyboard as the input device.

`class` **GCMouseInput**

A controller profile that tracks input from a mouse.

`class` **GCExtendedGamepad**

A controller profile that supports the extended set of gamepad controls.

`class` **GCDualShockGamepad**

A controller profile that supports the DualShock 4 controller.

`class` **GCXboxGamepad**

A controller profile that supports the Xbox controller.

`class` **GCDualSenseGamepad**

A controller profile that supported the DualSense controller.

`var` **microGamepad**`: GCMicroGamepad?`

The micro gamepad profile.

`class` **GCMicroGamepad**

A controller profile that supports the Siri Remote.

`class` **GCDirectionalGamepad**

A profile that supports only the directional pad, without motion or rotation.

`var` **motion**`: GCMotion?`

The motion input profile.

`var` **physicalInputProfile**`: GCPhysicalInputProfile`

The physical input profile for the controller.

~~`var` **gamepad**`: GCGamepad?`~~

The gamepad profile.

Deprecated

Accessing controller elements

`class` **GCControllerElement**

An input for a physical control, such as a button or thumbstick.

`class` **GCControllerAxisInput**

A control element that tracks movement along an axis.

`class` GCControllerButtonInput

A control element that represents a button touch or press.

`class` GCControllerTouchpad

A control element that represents a touch event on a touchpad.

`class` GCControllerDirectionPad

A control element associated with a directional pad or a thumbstick.

`class` GCDeviceCursor

A control element for the cursor used as a directional pad.

`class` GCDualSenseAdaptiveTrigger

A class that encapsulates the features of a DualSense adaptive trigger.

Identifying controllers and displaying a player index

`var playerIndex: GCControllerPlayerIndex`

The player index for the controller.

`enum` GCControllerPlayerIndex

The possible values for controller player indices.

Accessing battery, haptics, and light objects

`var battery: GCDeviceBattery?`

The controller's battery information.

`var haptics: GCDeviceHaptics?`

The controller's haptics information.

`var light: GCDeviceLight?`

The controller's light settings.

Creating snapshots

`class func withExtendedGamepad() -> GCController`

Returns a snapshot of a newly created controller with an extended gamepad profile.

`class func withMicroGamepad() -> GCController`

Returns a snapshot of a newly created controller with a micro gamepad profile.

`func capture() -> GCController`

Returns a snapshot of the controller with its current element values.

`var isSnapshot: Bool`

A Boolean value that indicates whether the controller is a snapshot of a controller.

Responding to a paused controller or controller event

~~`var controllerPausedHandler: ((GCController) -> Void)?`~~

The block that the framework calls when the user presses the pause button on the controller.

Deprecated

`protocol GCGameControllerSceneDelegate`

`class GCEventInteraction`

An interaction that indicates the view's intent to receive game controller events through the Game Controller framework.

Identifying the activation context

`class GCGameControllerActivationContext`

Structures

`struct DidBecomeCurrentMessage`

A message that posts after a game controller becomes the most recently used controller.

`struct DidConnectMessage`

A message that posts after a game controller accessory connects to the device.

`struct DidDisconnectMessage`

A message that posts after a game controller accessory disconnects from the device.

`struct DidStopBeingCurrentMessage`

A message that posts after a game controller stops being the most recently used controller.

Relationships

Inherits From

`NSObject`

Conforms To

`CVarArg`
`CustomDebugStringConvertible`
`CustomStringConvertible`
`Equatable`
`GCDevice`
`Hashable`
`NSObjectProtocol`

See Also

Game controllers

{} Supporting Game Controllers

Support a physical controller or add a virtual controller to enhance how people interact with your game through haptics, lighting, and motion sensing.

Letting players use their second-generation Siri Remote as a game controller

Support the second-generation Siri Remote as a game controller in your Apple TV game.

Discovering and tracking spatial game controllers and styli

Receive controller and stylus input to interact with content in your augmented reality app.

`protocol GCDevice`

A protocol that defines a common interface for game input devices.

`class GCRacingWheel`

An object that represents a physical racing wheel controller connected to a device.

`class GCKeyboard`

An object that represents a physical keyboard connected to a device.

`class` GCMouse

An object that represents a physical mouse connected to a device.

`class` GCStylus

An object that represents a physical stylus connected to the device.