Article

Responding to control-based events using target-action

Handle user input by connecting buttons, sliders, and other controls to your app's code using the target-action design pattern.

Overview

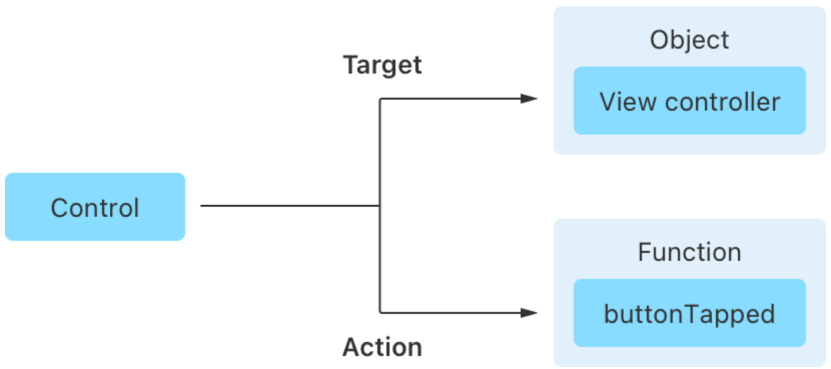
User interactions generate thousands of events for your app. Every swipe of the finger or button click, for instance, results in numerous events your app has to process.

`Target–action` is a design pattern to help you handle these user interactions efficiently. By only registering actions against specific events on your controls, you simplify your event processing, while making your code more maintainable and easier to read.

Use `target–action` to connect events directly from controls in your UI to methods in your code.

Designate an object to handle control-related events

When a person interacts with your control, the control needs to know where to send the event. The destination for the event is the *target*. View controllers make good targets because they're adept at handling user interactions, as well as hosting UI controls.



Define the action methods you use to respond to events

With a control assigned to the target, provide a function to call on that target when the event occurs. This is the *action* method.

Action methods have a distinct signature:

Swift Objective-C

```
// UIKit.
@IBAction func doSomething()
@IBAction func doSomething(sender: Any)
@IBAction func doSomething(sender: Any, forEvent event: UIEvent)

// AppKit.
@IBAction func doSomething()
@IBAction func doSomething(sender: Any)
```

Xcode uses the `@IBAction` keyword to connect controls in Interface Builder to functions in your code. The keyword also bridges the Swift and Objective-C runtimes, enabling Swift code to call into Objective-C, which is where the target-action functionality lives.

The `sender` parameter defines where the event comes from; for example, a button. Use the generic type `Any` to allow any control to call the action method, or specify the `sender` type when you need more information about the control for event processing. For example, set the `sender` type to `UISlider` to read the slider thumb value when someone moves the slider left or right.

Swift Objective-C

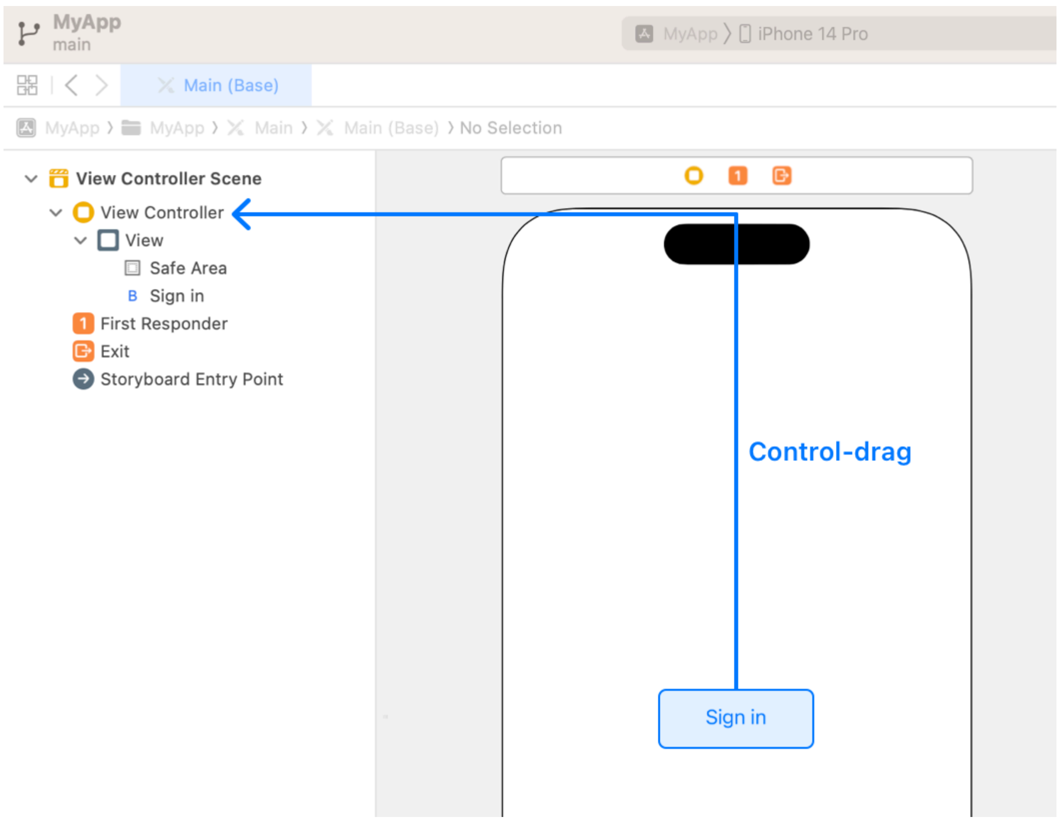
```
// UIKit.
@IBAction func adjustVolume(sender: UISlider) {
    print(sender.value)
}

// AppKit.
@IBAction func adjustVolume(sender: NSSlider) {
    NSLog("%@", sender.stringValue)
}
```

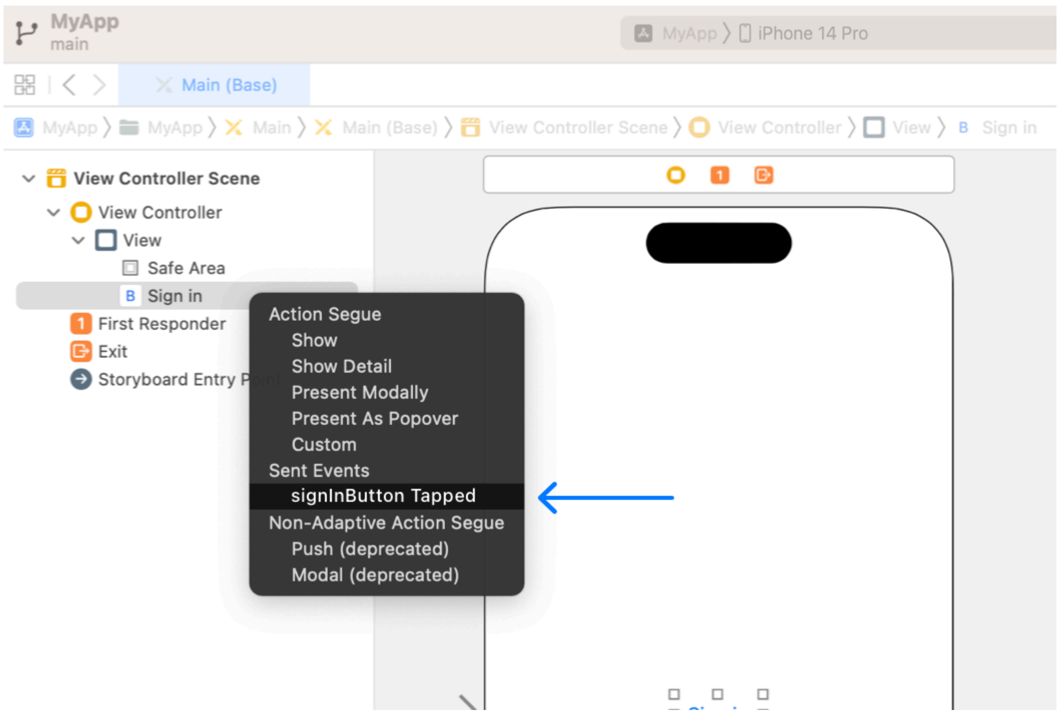
UIKit also supports coupling action methods to specific event types (see `UIControl.Event`). For example, to set up target-action for a user dragging their finger inside the bounds of a control, register for the event type `touchDragInside`.

Connect controls to the action methods in Interface Builder

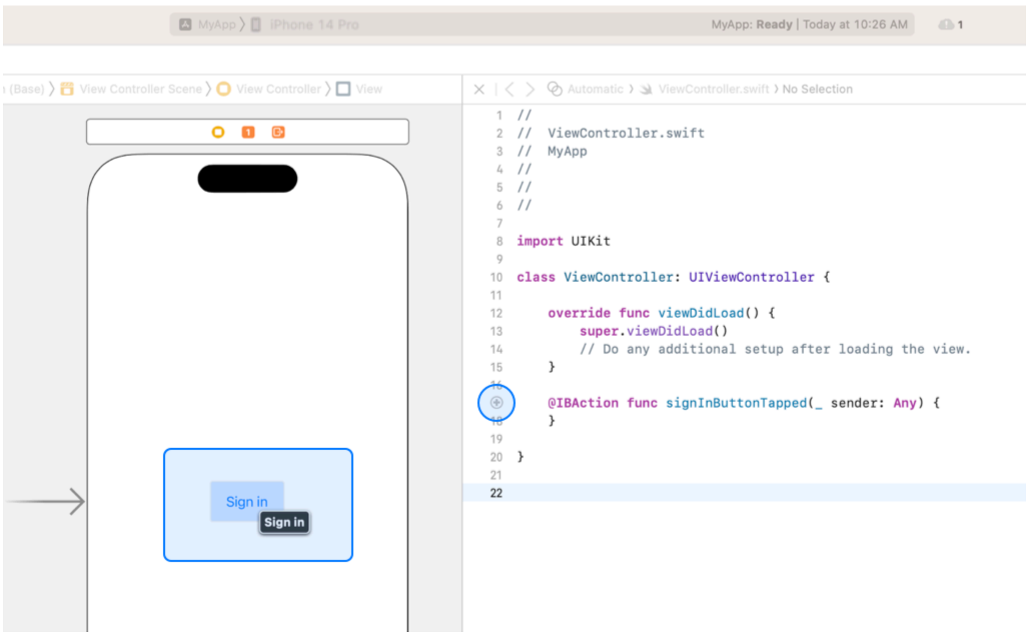
With the `target` and `action` methods defined, you're ready to connect them visually to the controls in Interface Builder.

1. Select the control (the `target`) you want to connect to the code (the `action`).
2. Control-drag the control to the view controller in the document outline.
3. Select the action method that the control connects to.



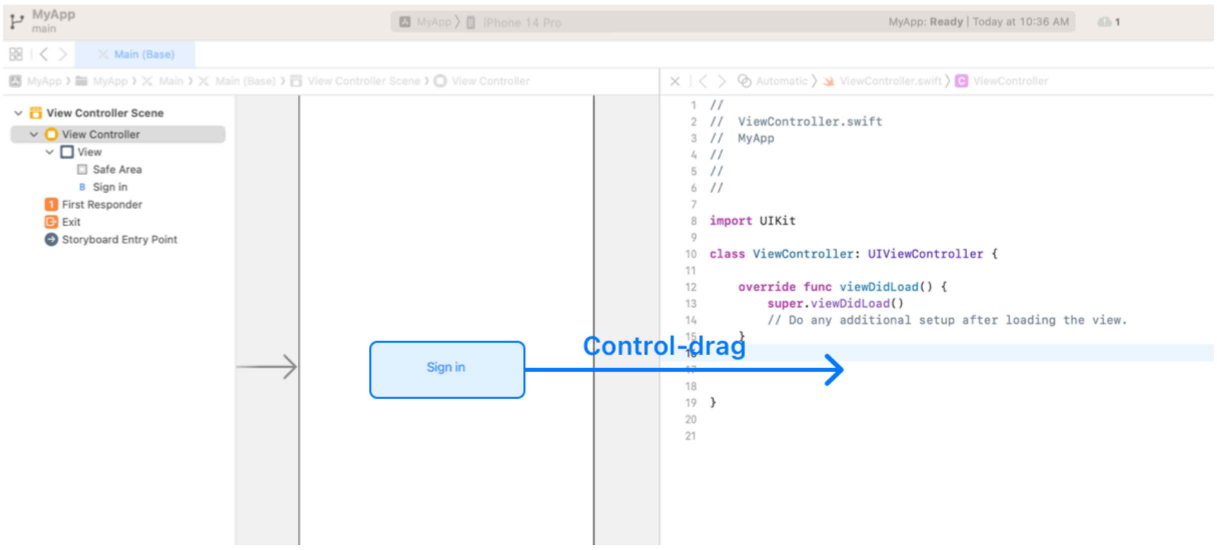


You can verify that the control and action are connected by moving the pointer over the circular dot to the left of the action method. When you do, Xcode highlights the control.


MyApp 〉 iPhone 14 Pro
MyApp: Ready | Today at 10:26 AM
(Base) 〉 View Controller Scene 〉 View Controller 〉 View
Automatic 〉 ViewController.swift 〉 No Selection
Sign in
Sign in

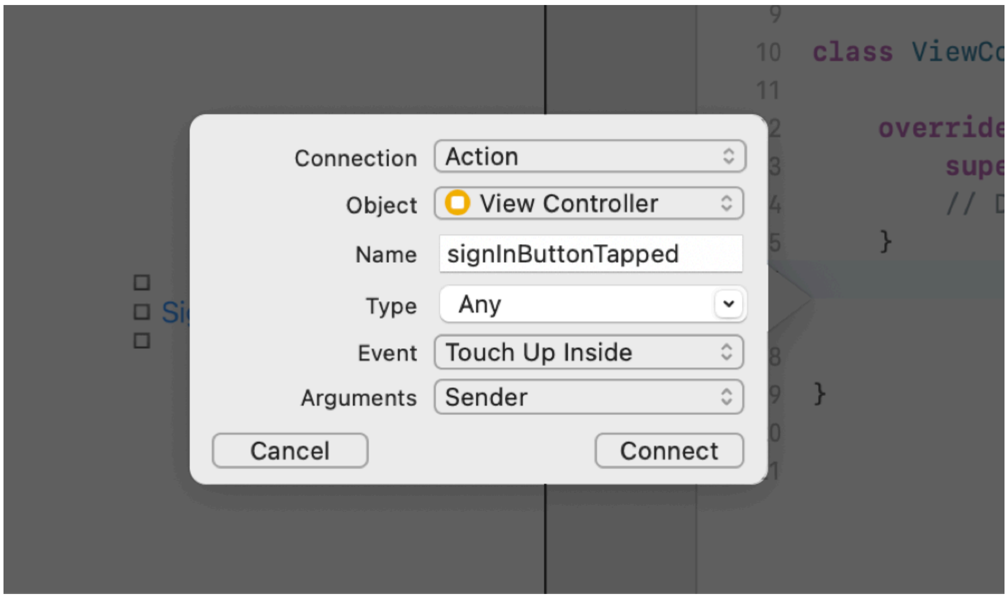
```
//
//  ViewController.swift
//  MyApp
//
//
//

import UIKit

class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view.
    }

    @IBAction func signInButtonTapped(_ sender: Any) {
    }

}
```

Another way to connect is to Control-drag the control from Interface Builder into the view controller.


MyApp
main
MyApp 〉 iPhone 14 Pro
MyApp: Ready | Today at 10:36 AM
Main (Base)
MyApp 〉 MyApp 〉 Main 〉 Main (Base) 〉 View Controller Scene 〉 View Controller
Automatic 〉 ViewController.swift 〉 ViewController
View Controller Scene
View Controller
View
Safe Area
Sign in
First Responder
Exit
Storyboard Entry Point
Sign in
Control-drag

```
//
//  ViewController.swift
//  MyApp
//
//
//

import UIKit

class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view.
    }


}
```

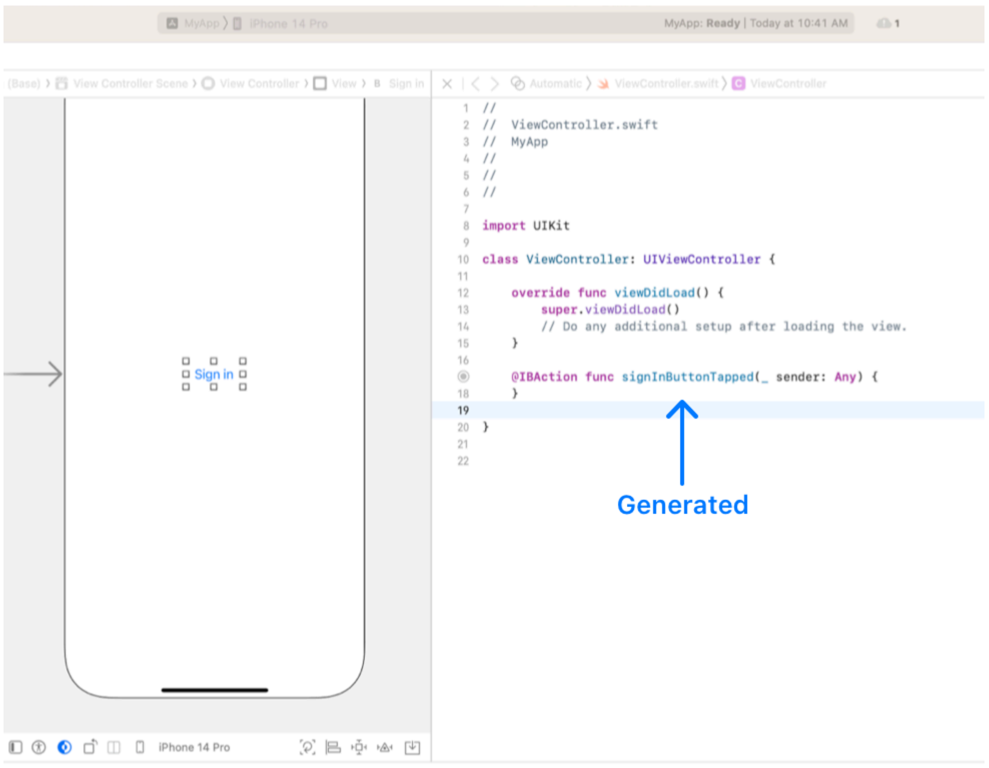
Enter the name of the `action` method you'd like the control to call and then click Connect.


Connection: Action
Object: View Controller
Name: signInButtonTapped
Type: Any
Event: Touch Up Inside
Arguments: Sender

The following table lists the parameters available for configuration.

Parameter	Description
Connection	The type of connection to make. Select Action to establish a target-action relationship between the control and the target.
Object	The `target` or object your control is connecting to.
Name	The name of the function, or `action`, your control calls when the event occurs.
Type	The type of the control sending the event. Choose `Any` if you don't require any further information regarding the control's state. Specify the control type if you require more information from the control for event processing.
Event	The `UIEvent` associated with the action. This feature is only available in UIKit.
Arguments	The arguments to include in the signature or your target-action method. Choose None to include no arguments. Choose Sender to pass in the control type as the sender. Choose Sender and Event to pass in both the control type and the event that caused the action. This feature is available only in UIKit.

Connecting the target-action this way ensures the method signature and parameters are correct as Xcode generates the action method for you.



```
//
//  ViewController.swift
//  MyApp
//
//
//

import UIKit

class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view.
    }

    @IBAction func signInButtonTapped(_ sender: Any) {
    }

}
```

Connect a control to your code programmatically

In UIKit, use the `addTarget(_:action:for:)` method to set up target-action programmatically on a control:

Swift Objective-C

```swift
import UIKit


class ViewController: UIViewController {
    let signInButton = UIButton(type: .system)

    override func viewDidLoad() {
        super.viewDidLoad()


        signInButton.translatesAutoresizingMaskIntoConstraints = false
        signInButton.setTitle("Sign in", for: .normal)

        // Target-action.
        signInButton.addTarget(self, action: #selector(buttonTapped), for: .touchUpI

        view.addSubview(signInButton)
        signInButton.centerXAnchor.constraint(equalTo: view.centerXAnchor).isActive
        signInButton.centerYAnchor.constraint(equalTo: view.centerYAnchor).isActive
    }

    @IBAction func buttonTapped(sender: UIButton) {
        print("Sign in successful 🎉")
    }
}
```

In AppKit, use the control's `target` and `action` properties to set target-action programmatically:

Swift   Objective-C

```
import Cocoa

class ViewController: NSViewController {
    let signInButton = NSButton()

    override func viewDidLoad() {
        super.viewDidLoad()

        signInButton.translatesAutoresizingMaskIntoConstraints = false
        signInButton.title = "Sign in"

        // Target-action.
        signInButton.target = self
        signInButton.action = #selector(buttonTapped(sender:))

        view.addSubview(signInButton)
        signInButton.centerXAnchor.constraint(equalTo: view.centerXAnchor).isActive
        signInButton.centerYAnchor.constraint(equalTo: view.centerYAnchor).isActive
    }

    @IBAction func buttonTapped(sender: NSButton) {
        print("Sign in successful 🎉")
    }
}
```

Enable other objects to respond to control-related events

When the object hosting the control isn't the one you want handling the event, invoke the responder chain by passing in `nil` as the target. This causes the control to search the responder chain for the specified action method and invoke it when found. For more information, see Using responders and the responder chain to handle events.

See Also

Controls

`class UIControl`

The base class for controls, which are visual elements that convey a specific action or intention in response to user interactions.

`class UIButton`

A control that executes your custom code in response to user interactions.

`class UIColorWell`

A control that displays a color picker.

`class UIDatePicker`

A control for inputting date and time values.

`class UIPageControl`

A control that displays a horizontal series of dots, each of which corresponds to a page in the app’s document or other data-model entity.

`class UISegmentedControl`

A horizontal control that consists of multiple segments, each segment functioning as a discrete button.

`class UISlider`

A control for selecting a single value from a continuous range of values.

`class UIStepper`

A control for incrementing or decrementing a value.

`class UISwitch`

A control that offers a binary choice, such as on/off.