

< [How to detect the location of a tap inside a view](#)

[How to use @StateObject to create and monitor external objects](#) >

What's the difference between @ObservedObject, @State, and @EnvironmentObject?

Paul Hudson @twostraws September 3rd 2021

Updated for Xcode 13.2


State is inevitable in any modern app, but with SwiftUI it's important to remember that all of our views are simply functions of their state – we don't change the views directly, but instead manipulate the state and let that dictate the result.

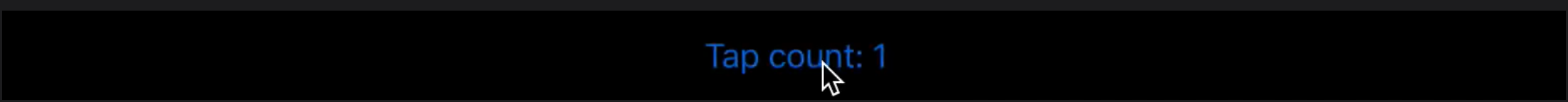
SwiftUI gives us several ways of storing state in our application, but they are subtly different and it's important to understand *how* they are different in order to use the framework properly.

The simplest way of working with state is the `@State` property wrapper, used like this:

```
struct ContentView: View {
    @State private var tapCount = 0

    var body: some View {
        Button("Tap count: \(tapCount)") {
            tapCount += 1
        }
    }
}
```

 Download this as an Xcode project



This creates a property inside a view, but it uses the `@State` property wrapper to ask SwiftUI to manage the memory. This *matters*: all our views are structs, which means they can't be changed, and if we weren't even able to modify an integer in our apps then there wouldn't be much we *could* do.

So, when we say `@State` to make a property, we hand control over it to SwiftUI so that it remains persistent in memory for as long as the view exists. When that state changes, SwiftUI knows to automatically reload the view with the latest changes so it can reflect its new information.

`@State` is great for simple properties that belong to a specific view and never get used outside that view, so as a result it's important to mark those properties as being private to re-enforce the idea that such state is specifically designed never to escape its view.

What is @ObservedObject?

For more complex properties – when you have a custom type you want to use that might have multiple properties and methods, or might be shared across multiple views – you will often use `@ObservedObject` instead.

This is very similar to `@State` except now we're using an external reference type rather than a simple local property like a string or an integer. You're still saying that your view depends on data that will change, except now it's data you're responsible for managing yourself – you need to create an instance of the class, create its own properties, and so on.

Whatever type you use with `@ObservedObject` should conform to the `ObservableObject` protocol. When you add properties to observable objects you get to decide whether changes to each property should force views that are watching your object to refresh or not. You usually will, but it's not required.

There are several ways for an observed object to notify views that important data has changed, but the easiest is using the `@Published` property wrapper. You can also use custom publishers from the Combine framework if you need more control, but realistically this will be very rare. If the observable object happens to have several views using its data, either option will automatically notify them all.

Warning: When you use a custom publisher to announce that your object has changed, this *must* happen on the main thread.

What is @StateObject?

Somewhere between `@State` and `@ObservedObject` lies `@StateObject`. This is a specialized version of `@ObservedObject`, and it works in almost exactly the same way: you must conform to the `ObservableObject` protocol, you can use `@Published` to mark properties as causing change notifications, and any views that watch an `@StateObject` will refresh their body when the object changes.

There is one important difference between `@StateObject` and `@ObservedObject`, which is *ownership* – which view *created* the object, and which view is just *watching* it.

The rule is this: whichever view is the first to create your object must use `@StateObject`, to tell SwiftUI it is the owner of the data and is responsible for keeping it alive. All other views must use `@ObservedObject`, to tell SwiftUI they want to watch the object for changes but don't own it directly.

What is @EnvironmentObject?

You've seen how `@State` declares simple properties for a type that automatically cause a refresh of the view when it changes, and how `@ObservedObject` declares a property for an external type that may or may not cause a refresh of the view when it changes. Both of these two must be set by your view, but `@ObservedObject` might be shared with other views.


There's another type of property wrapper available to use, which is `@EnvironmentObject`. This is a value that is made available to your views through the application itself – it's shared data that every view can read if they want to. So, if your app had some important model data that all views needed to read, you could either hand it from view to view or just put it into the environment where every view has instant access to it.

Think of `@EnvironmentObject` as a massive convenience for times when you need to pass lots of data around your app. Because all views point to the same model, if one view changes the model all views immediately update – there's no risk of getting different parts of your app out of sync.

Summing up the differences

- Use `@State` for simple properties that belong to a single view. They should usually be marked `private`.
- Use `@ObservedObject` for complex properties that might belong to several views. Most times you're using a reference type you should be using `@ObservedObject` for it.
- Use `@StateObject` once for each observable object you use, in whichever part of your code is responsible for creating it.
- Use `@EnvironmentObject` for properties that were created elsewhere in the app, such as shared data.

Of the four you will find that `@ObservedObject` is both the most useful and the most commonly used, so if you're not sure which to use start there.

 **Raycast**

SPONSORED

Get work done faster with [Raycast](#). The Mac app brings the macOS Spotlight experience to the next level: Search files, create GitHub pull requests, close Jira issues and so much more. Use the [Xcode extension](#) to open recent projects, search iOS documentation and clear derived data. Or, build your own extension with an easy-to-use API.

[Find out more](#)

Sponsor Hacking with Swift and reach the world's largest Swift community!

Similar solutions...

- [@State Property Wrapper explained and compared](#)
- [SwiftUI tips and tricks](#)
- [How to use Instruments to profile your SwiftUI code and identify slow layouts](#)
- [Adding items to an order with @EnvironmentObject](#)
- [Building a menu using List](#)

< [How to detect the location of a tap inside a view](#)

[How to use @StateObject to create and monitor external objects](#) >



Was this page useful? Let us know!

★★★★★

Average rating: 4.8/5

[Click here to visit the Hacking with Swift store >>](#)



@twostraws



paul@hackingwithswift.com



Sponsor the site

[About](#) [Glossary](#) [Privacy Policy](#) [Refund Policy](#) [Update Policy](#) [Code of Conduct](#)

Thanks for your support, Philip Bacchi!

Swift, SwiftUI, the Swift logo, Swift Playgrounds, Xcode, Instruments, Cocoa Touch, Touch ID, AirPods, iBeacon, iPhone, iPad, Safari, App Store, watchOS, tvOS, Mac and macOS are trademarks of Apple Inc., registered in the U.S. and other countries. Pulp Fiction is copyright © 1994 Miramax Films. Hacking with Swift is ©2021 Hudson Heavy Industries.



You are not logged in

[Log in or create account](#)