< What is the @Published property wrapper?

What is the @EnvironmentObject property wrapper? >

What is the @ObservedObject property wrapper?

Paul Hudson @twostraws February 9th 2021

Updated for Xcode 13.2

SwiftUI gives us the `@ObservedObject` property wrapper so that views can watch the state of an external object, and be notified when something important has changed. It is similar in behavior to `@StateObject`, except it must *not* be used to create objects – use `@ObservableObject` only with objects that have been created elsewhere, otherwise SwiftUI might accidentally destroy the object.

For example, we might use something like this:

```
class Order: ObservableObject {
    @Published var items = [String]()
}

struct ContentView: View {
    @ObservedObject var order: Order

    var body: some View {
        // your code here
    }
}
```

That `Order` class uses `@Published` so it will automatically send change announcements when `items` changes, and `ContentView` uses `@ObservedObject` to watch for those announcements. Without `@ObservedObject` the change announcements would be sent but ignored.

Although that looks straightforward enough, it's worth digging into a few specifics.

First, any type you mark with `@ObservedObject` must conform to the `ObservableObject` protocol, which in turn means it must be a class rather than a struct. This isn't optional – SwiftUI requires us to use a class here.

Second, observed objects are specifically designed for data that is external to your view, which means it might be shared across more than one view. The `@ObservedObject` property wrapper will automatically make sure the property is watched closely so that important changes will reload any views using it. This also means the data must be created elsewhere, then sent in to your view.

Third, not all properties in an observed object cause views to refresh – you need to decide which properties should send change notifications, either using `@Published` or custom announcements. Types that conform to `ObservableObject` are given a default `objectWillChange` publisher to make custom announcements as needed.

Similar solutions...

- What's the difference between @ObservedObject, @State, and @EnvironmentObject?
- How to use @ObservedObject to manage state from external objects
- How to fix "Referencing initializer 'init(wrappedValue:)' on 'ObservedObject' requires that 'SomeType' conform to 'ObservableObject'"
- All SwiftUI property wrappers explained and compared
- What is the @GestureState property wrapper?

< What is the @Published property wrapper?

What is the @EnvironmentObject property wrapper? >