Understanding Responders and the Responder Chain

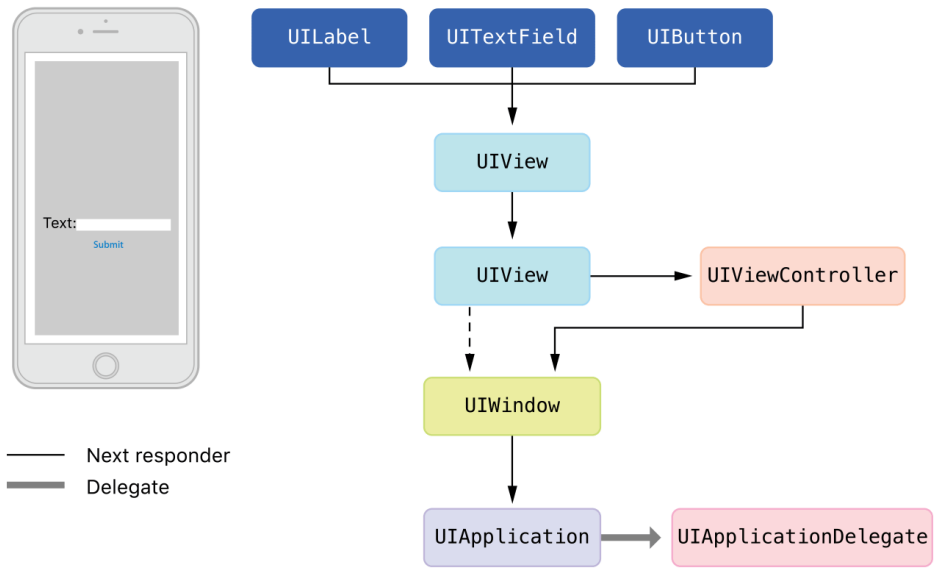
On This Page

Apps receive and handle events using responder objects. A responder object is any instance of the `UIResponder` class, and common subclasses include `UIView`, `UIViewController`, and `UIApplication`. UIKit manages most responder-related behavior automatically, including how events are delivered from one responder to the next. However, you can modify the default behavior to change how events are delivered within your app.

UIKit directs most events to the most appropriate responder object in your app. If that object does not handle the event, UIKit forwards it to the next responder in the active *responder chain*, which is a dynamic configuration of your app's responder objects. Because it is dynamic, there is no single responder chain within your app. However, it is easy to determine the next responder in the chain because events always flow from specific responder objects to more general responder objects. For example, the next responder for a view is either its superview or the view controller that manages it. Events continue to flow up the responder chain until they are handled.

Figure 7-1 shows the responder chains that form in an app whose interface contains a label, a text field, a button, and two background views. If the text field does not handle an event, UIKit sends the event to the text field's parent `UIView` object, followed by the root view of the window. From the root view, the responder chain diverts to the owning view controller before returning to the view's window. If the window does not handle the event, UIKit delivers the event to the `UIApplication` object, and possibly to the app delegate if that object is an instance of `UIResponder` and not already part of the responder chain.

Figure 7-1 An example of a responder chain



For every event, UIKit designates a *first responder* and sends the event to that object first. The first responder varies based on the type of event.

- **Touch events.** The first responder is the view in which the touch occurred. For information about handling these events, see Handling Touches in Your View.
- **Press events.** The first responder is the responder that has focus. For information about handling these events, see *App Programming Guide for tvOS*.
- **Motion events.** The first responder is the object that you designated to handle the events. Core Motion handles events related to the accelerometers, gyroscopes, and magnetometer. Motion events do not follow the responder chain.
- **Shake-motion events.** The first responder is the object that you (or UIKit) designate as the first responder. For information about handling these events, see Handling UIKit Shake Gestures.
- **Remote-control events.** The first responder is the object that you (or UIKit) designate as the first responder. For information about handling these events, see Handling Remote-Control Events.
- **Editing-menu messages.** The first responder is the object that you (or UIKit) designate as the first responder. For information about the UIKit editing commands, see `UIResponderStandardEditActions`.

Action messages sent by controls to their associated object are not events, but they may still take advantage of the responder chain. When the target object of a control is `nil`, UIKit walks the responder chain from the target object and looks for an object that implements the appropriate action method. For example, the UIKit

editing menu uses this behavior to search for responder objects that implement methods with names like `cut:`, `copy:`, or `paste:`.

If a view has an attached gesture recognizer, the gesture recognizer may delay the delivery of touch and press events to the vie

On This Page

`UIGestureRecognizer` determine when and how touches are delayed. For more information about using gesture recognizers to handle events, see Gesture Recognizer Basics.

Determining Which Responder Contained a Touch Event

UIKit uses view-based hit testing to determine where touch events occur. Specifically, UIKit compares the touch location to the bounds of view objects in the view hierarchy. The `hitTest:withEvent:` method of `UIView` walks the view hierarchy, looking for the deepest subview that contains the specified touch. That view becomes the first responder for the touch event.

> NOTE
>
> If a touch location is outside of a view's bounds, the `hitTest:withEvent:` method ignores that view and all of its subviews. As a result, when a view's `clipsToBounds` property is `NO`, subviews outside of that view's bounds are not returned even if they happen to contain the touch. For more information about the hit testing behavior, see the discussion of the `hitTest:withEvent:` method in `UIView`.

UIKit permanently assigns each touch to the view that contains it. UIKit creates each `UITouch` object when the touch first occurs, and it releases that touch object only after the touch ends. As the touch location or other parameters change, UIKit updates the `UITouch` object with the new information. Several properties of the touch do not change, including the assigned view. Even when the touch location moves outside the original view, the value in the touch's `view` property remains the same.

Altering the Responder Chain

You can alter the responder chain by overriding the `nextResponder` property of your responder objects. Many UIKit classes already override this property and return specific objects.

- If you override the `nextResponder` property for any class, the next responder is the object you return.
- `UIView`
  - If the view is the root view of a view controller, the next responder is the view controller.
  - If the view is not the root view of a view controller, the next responder is the view's superview.
- `UIViewController`
  - If the view controller's view is the root view of a window, the next responder is the window object.
  - If the view controller was presented by another view controller, the next responder is the presenting view controller.
- `UIWindow`. The window's next responder is the application object.
- `UIApplication`. The app object's next responder is the app delegate, but only if the app delegate is an instance of `UIResponder` and is not a view, view controller, or the app object itself.

For more information about responders, see `UIResponder`.