Directing Input to a Specific Responder

On This Page

Touch events are automatically dispatched to the view in which they occurred, but most other events are directed to the object designated as the first responder. You can also set the first responder programmatically to any of your app's responder objects. User interactions may also cause a view to become the first responder automatically, but only if it wants to be the first responder. For example, tapping a text field causes the text field to become the first responder, but tapping a button triggers the button's action and does not cause it to become the first responder.

When an object becomes the first responder, UIKit displays the *input view* associated with that responder. An input view is a custom interface that you can use to gather information. The system keyboard is an example of an input view.

Programmatically Changing the First Responder

To designate an object as the first responder, call its `becomeFirstResponder` method. UIKit makes the object the first responder only if the following conditions are met:

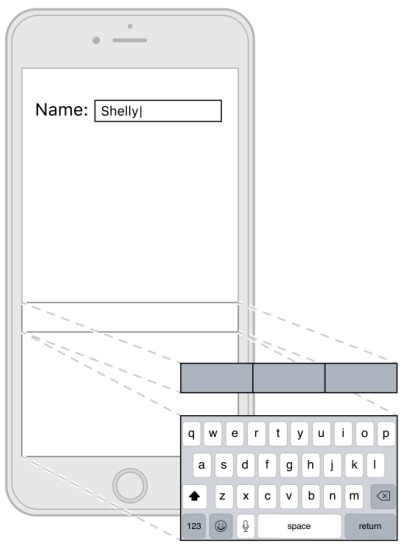
- The current first responder's `canResignFirstResponder` property returns `YES`.
- The new responder's `canBecomeFirstResponder` property returns `YES`. (The default implementation of this property returns `NO`, so you must override it to allow your object to become the first responder.)

Most responder objects resign the first responder status readily, but you can return `NO` as needed from the `canResignFirstResponder` property. For example, you might return `NO` to prevent your custom control from resigning as first responder while it contains invalid data.

Assigning an Input View to a Responder

When an object becomes the first responder, UIKit displays its associated input view. An input view handles user interactions and generates data for the underlying responder object. Input views are used mostly to manage custom data entry for views. For example, the default input view for text fields and text views is the keyboard, shown in Figure 8-1. The keyboard generates character strings and delivers them to the view's delegate.

Figure 8-1 The input view for a text field



Every responder can have a custom input view, which you assign to the `inputView` property. (If you prefer to manage your input view using a view controller, assign the view controller to the `inputViewController` property instead.) When the responder becomes the first responder, UIKit animates your input view into position. Here are some tips for setting up your input view:

- **Configure your input view to span the width of the screen.** UIKit sets the size of your input view to the screen width and the height you specify for your view.

- **Use a delegate to report data back to your responder.** It is your responsibility to manage communications between your responder and the input view. One way to do this is for your input view to deliver updates to an assigned delegate object.
- **Use the k**
  `UIKeyboar`
  `UIKeyboardDidChangeFrameNotification`, and `UIKeyboardDidShowNotification` notifications (in that order) for your custom input views as well as for the system keyboard. Use these notifications to update your view or animate other changes in your interface. For example, you might adjust the frame of the underlying view to ensure that controls are not hidden by your input view.

On This Page

In addition to an input view, your responder can also have an *accessory view*, which you provide using the `inputAccessoryView` or `inputAccessoryViewController` property. Accessory views supplement your main input view. For example, the default accessory view for the keyboard lists potential completions for partially typed words. It is your responsibility to manage any communication between your input view and your accessory view.