

Keyboard control in UIKit apps

Douglas Hill • @qdoug
iOS Conf SG 2020

iPad Pro

Overview Design Why iPad Tech Specs Buy

Why iPad

Like a computer. Unlike any computer.

With iPad, you get what you need from a computer, along with many incredible things you'd never expect from one. Here are a few reasons why your next computer just might be an iPad.



It works like a computer. And in ways most computers can't.

iPad works with a keyboard when you need one. The onscreen keyboard lets you respond to an email or write a paper, and it even acts as a trackpad. And if you want a physical keyboard, just attach the Smart Keyboard for a great typing experience that also doubles as a cover or easily pair a Bluetooth keyboard.³



USE FAMILIAR SHORTCUTS

JUST ATTACH AND START TYPING

Bring Your iPad App to Mac

Now it's incredibly simple to start building a native Mac app from your current iPad app. With Mac Catalyst, your apps share the same project and source code so you can efficiently convert your iPad app's desktop-class features, and add more just for Mac. Deliver your new Mac app to an engaged audience of over 100,000,000 active Mac users.



Get a Head Start on Your Native Mac App

“As a general rule, iOS apps don’t tend to be optimised for keyboard interactions.”

— Apple, Mac Catalyst Human Interface Guidelines



- API
- User experience
- Implementation



A Complete PDF Solution You Can Rely On

PSPDFKit is the best framework for working with PDF files. Our SDK provides first-rate PDF solutions for your application with features like annotating, signing, and form filling. We power successful projects for businesses across the globe.

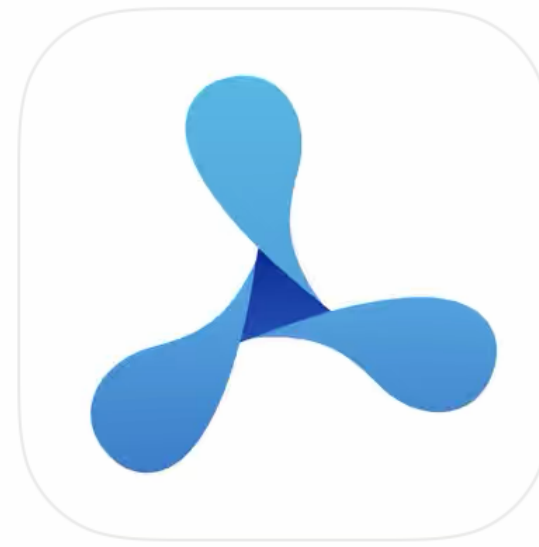


Try PSPDFKit in your app today.

DOWNLOAD TRIAL →



Today



PDF Viewer - Annotation Expert

Fill Forms, Sign, Edit, Redact



4.7 ★★★★★
409 Ratings

4+
Age

What's New

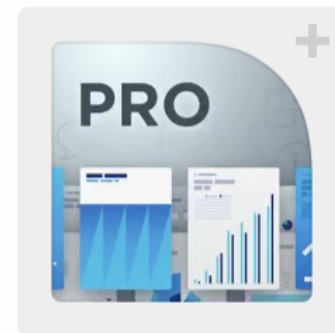
[Version History](#)

Improvements to image selection. Fixes a crash that could occur when opening PDF Viewer from its widget
Here's the full list of changes:

[more](#)

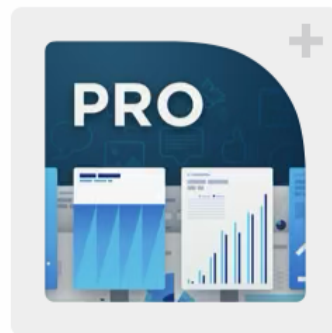
3w ago
Version 4.1.1

Subscriptions



PDF Viewer Pro 3 Months
Redaction, PDF merging & more for 3 months

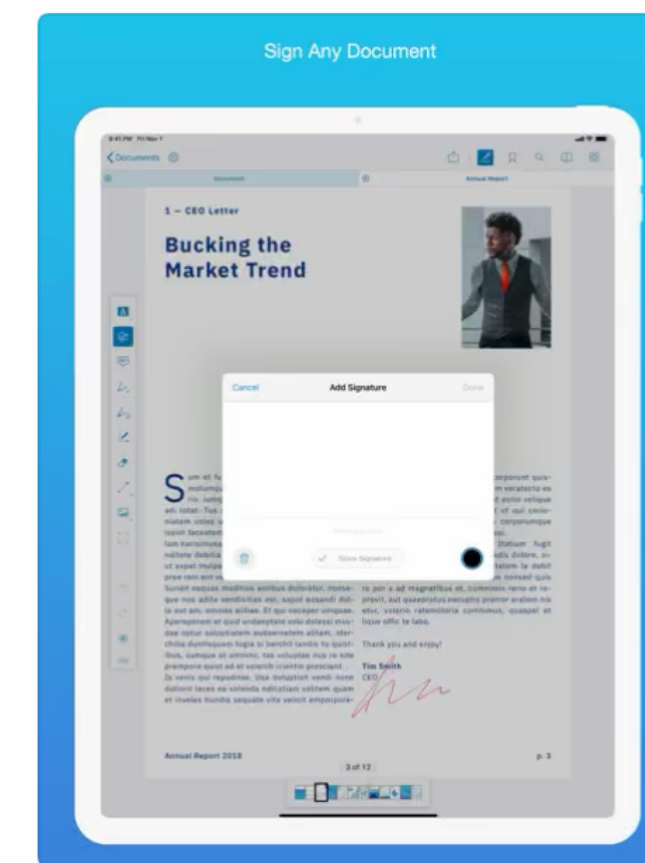
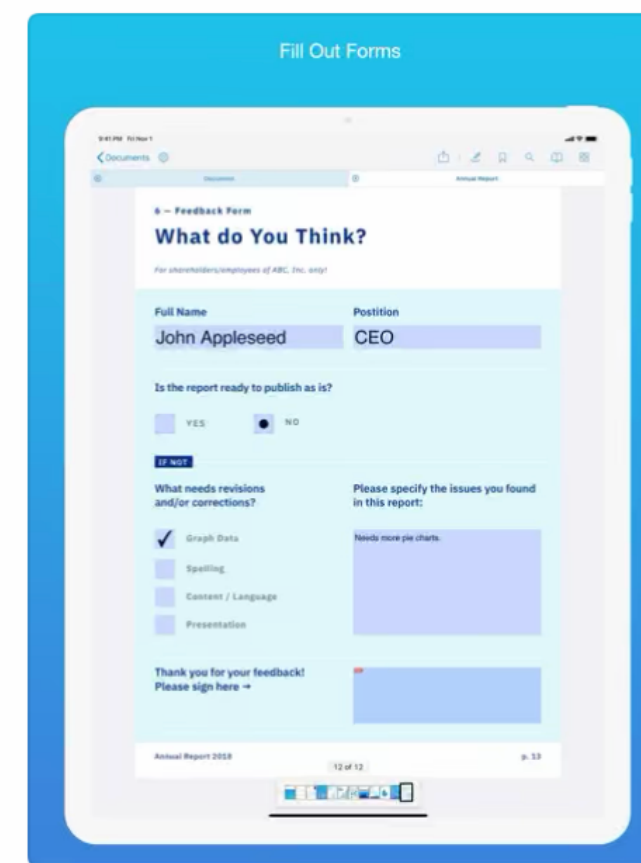
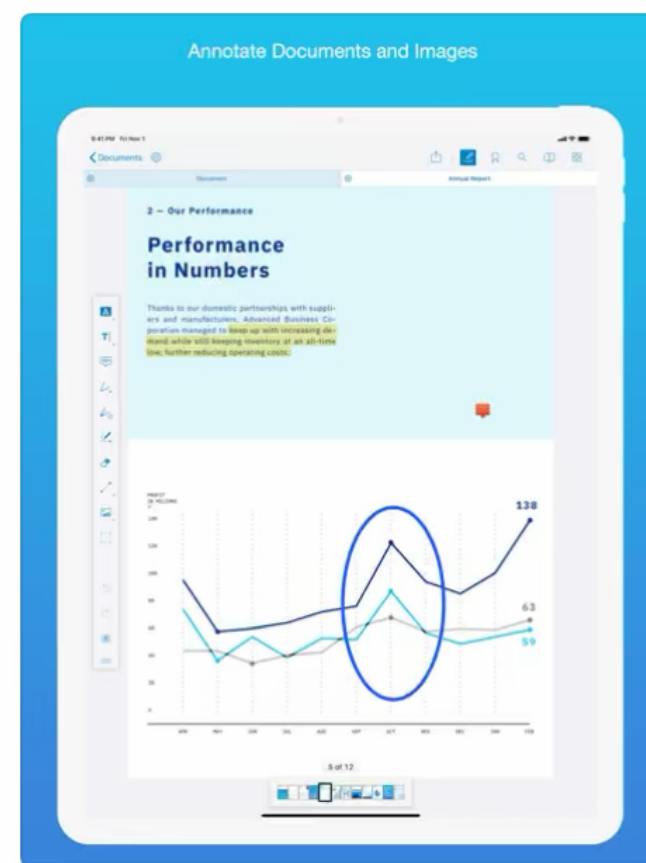
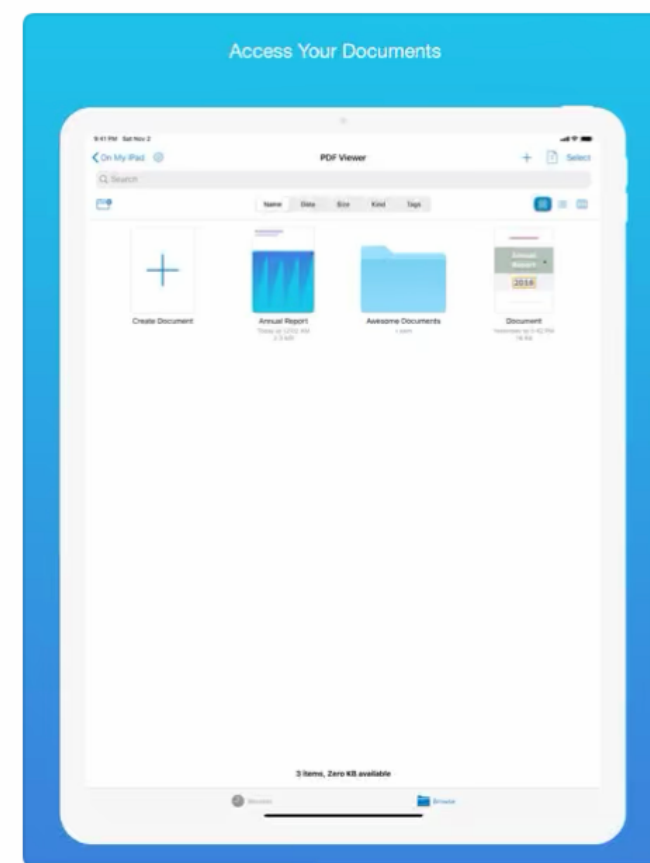
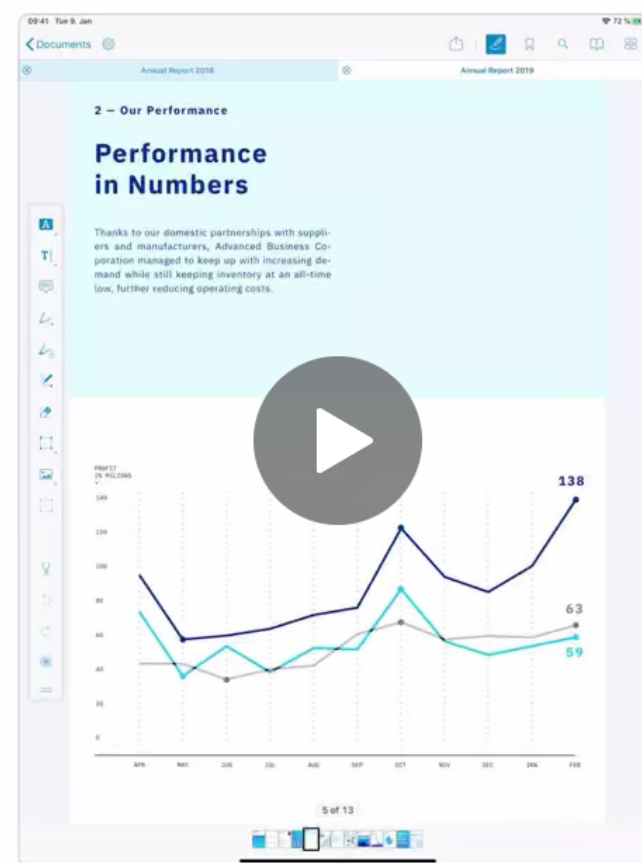
£6.49

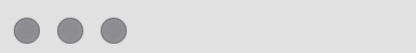


PDF Viewer Pro Yearly
Redaction, PDF merging & more for 1 year

SUBSCRIBED

Preview





Search

- ★ Discover
- 🎮 Arcade
- 🔨 Create
- 📁 Work
- 🎮 Play
- 🔨 Develop
- 📁 Categories
- 📁 Updates



PDF Viewer - Annotation Expert

Fill Forms, Sign, Edit, Redact
PSPDFKit GmbH

OPEN



4.7 ★★★★★

3 Ratings

4+

Age

What's New

[Version History](#)

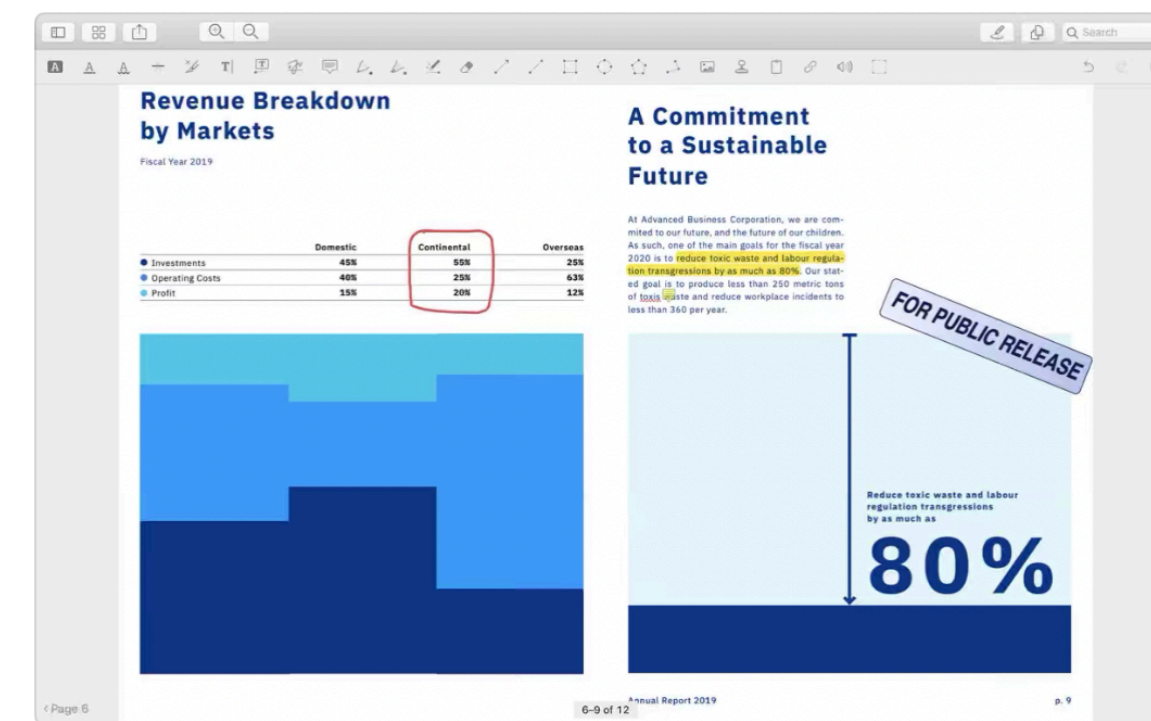
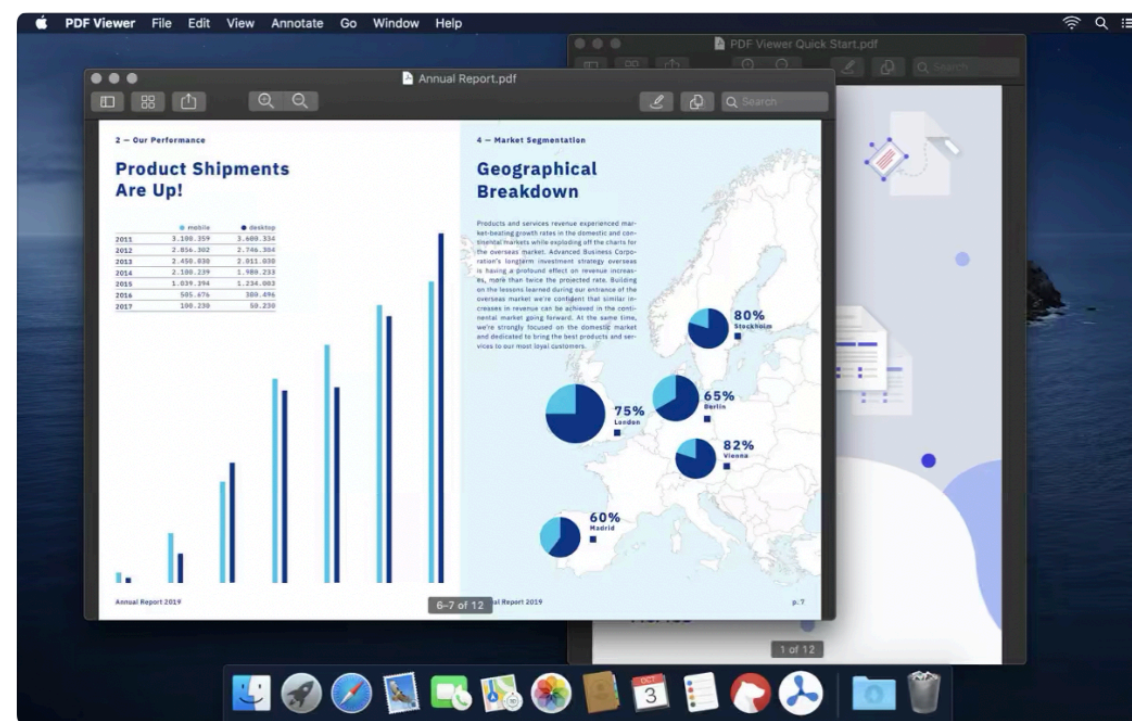
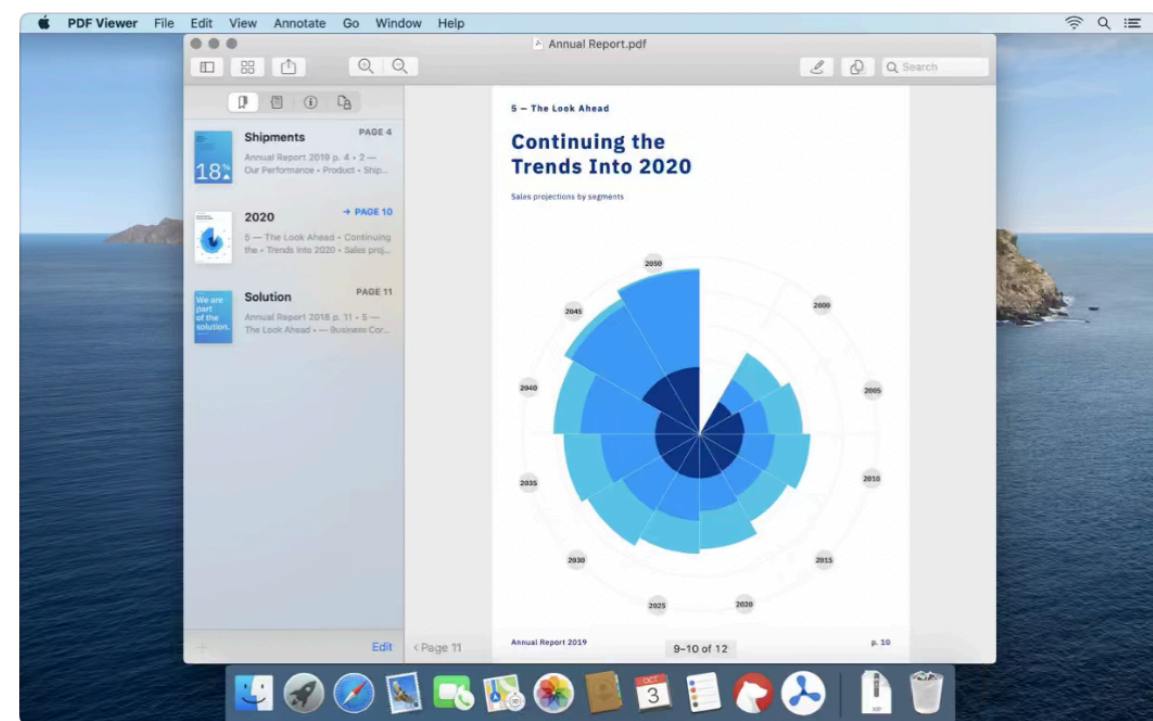
Improvements to text and image selection. Fixes saving when closing windows.
Here's the full list of changes:

[more](#)

2w ago

Version 4.1.1

Preview



PDF Viewer Pro is a fast and beautiful app, allowing you to view, search, and annotate PDF documents with ease on your Mac.

VIEW DOCUMENTS ON YOUR MAC

- Open files from the Finder. Supports Drag & Drop.
- Search for the exact text you are looking for within the document.
- Annotate images (JPEG, PNG) non-destructive, just like PDF documents. (PRO)

ANNOTATE WITH EASE

- Highlight and markup text while reviewing a document.
- Leave comments by adding a note, text, or drawing directly on the PDF.
- Add images or audio.
- Reply to notes. (PRO)

SIGN DOCUMENTS ANYWHERE

- Add your signature to any document.
- Move and resize your signature as needed.
- Flatten a document to ensure the signature doesn't change.

CREATE AND EDIT PDFs

- Create new documents from existing documents by moving, rotating, deleting, or adding new pages.
- Combine multiple documents into a single one. (PRO)
- Leave a simple bookmark or create an entire table of contents for your document.

PSPDFKit GmbH

[Website](#)

[Support](#)

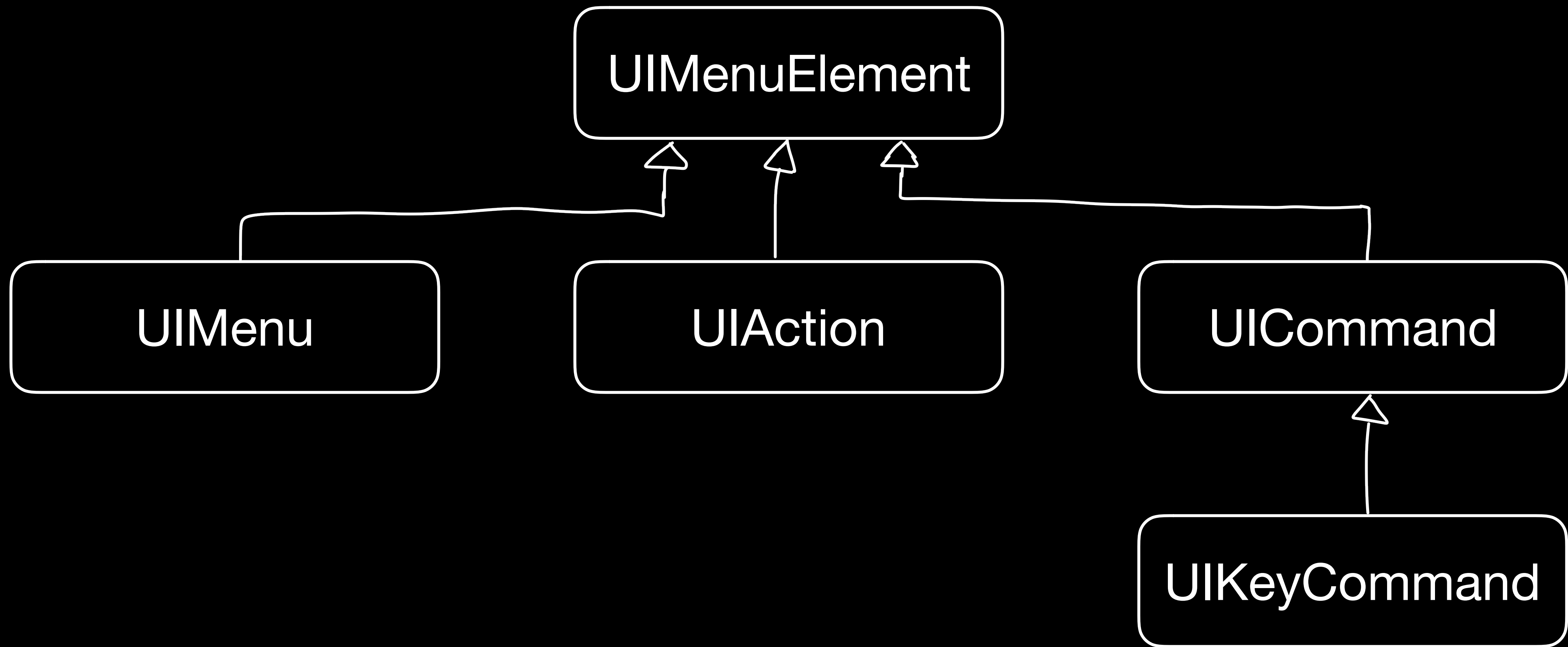


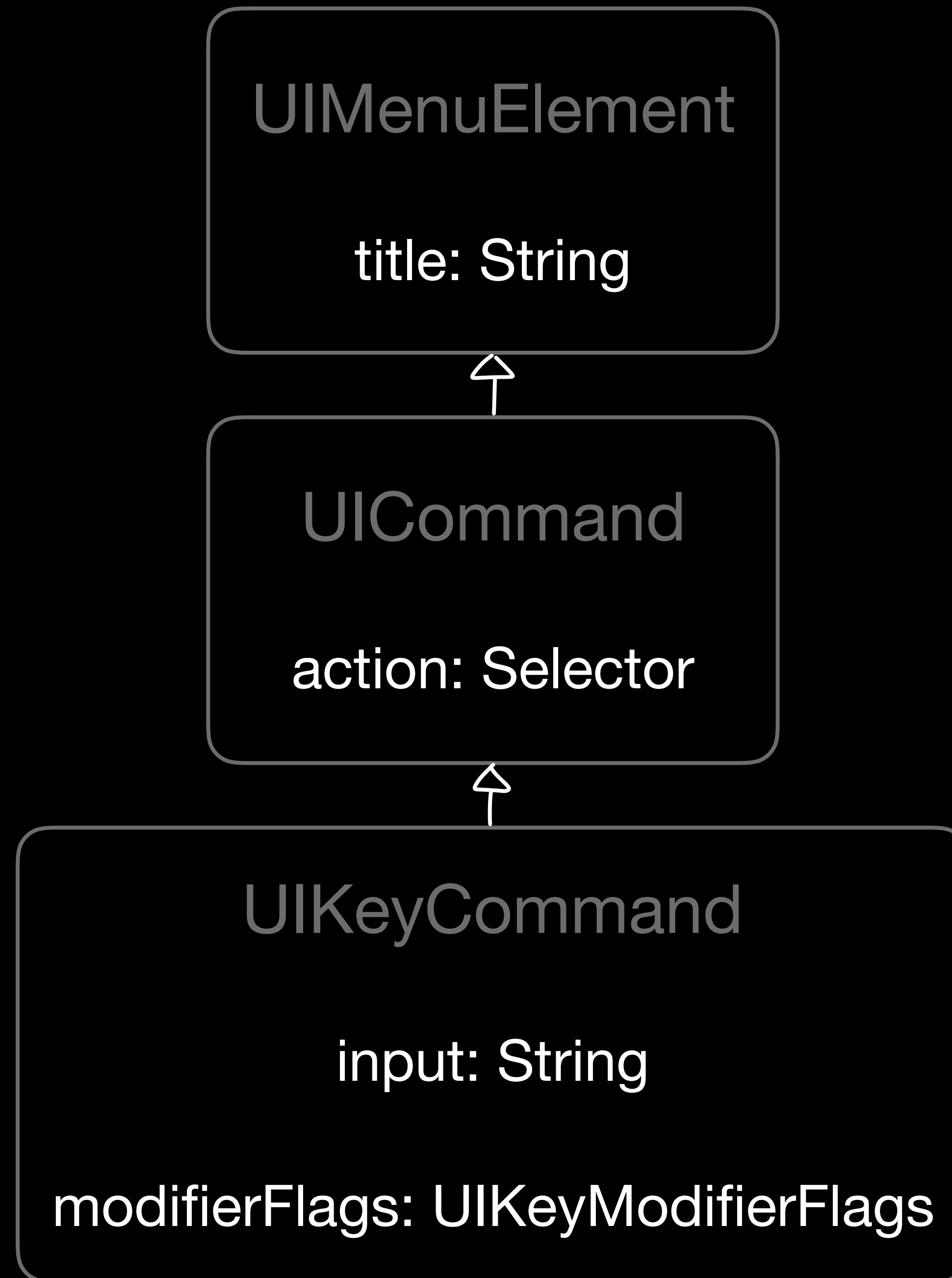
- Speed
- Ergonomics



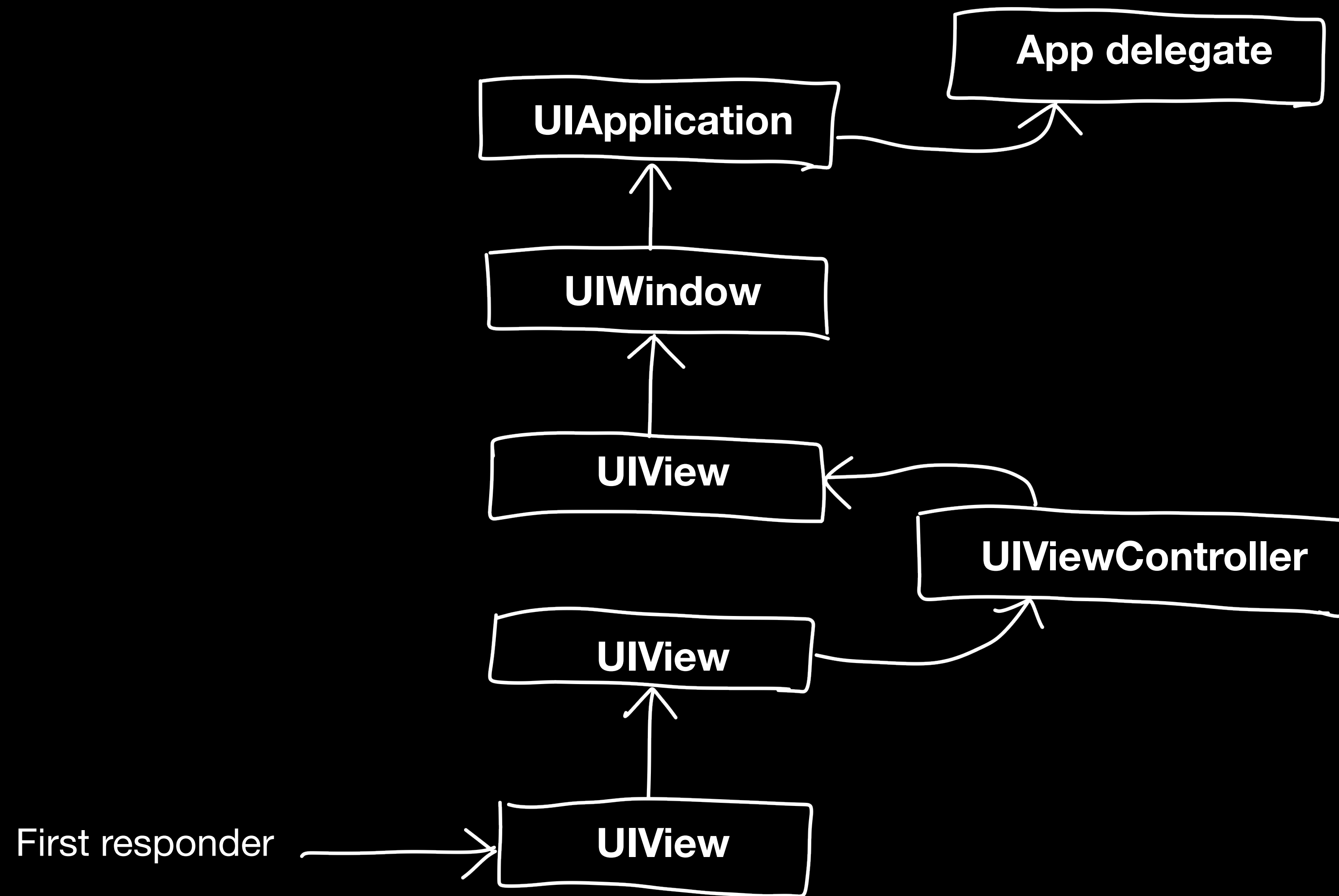
Myke Hurley, @imyke

- API
- User experience
- Implementation





Responder chain



- How do we provide key commands to the system?
- How does the system decide possible key commands?
- How does a key command run our code?

- How do we provide key commands to the system?
- How does the system decide possible key commands?
- How does a key command run our code?


```
extension UIResponder {  
    open var keyCommands: [UIKeyCommand]? { get }  
}
```

```
extension UIViewController {  
    open func addKeyCommand(_ keyCommand: UIKeyCommand)  
}
```



```
open class UIResponder: NSObject {  
    open func buildMenu(with builder: UIMenuBuilder)  
}
```

- How do we provide key commands to the system?
- How does the system decide possible key commands?
- How does a key command run our code?

Getting Started. Integration

PSPDFKit 9 for iOS is designed for Xcode 11+ (SDK 13+), and fully supports iOS 11, iOS 12 and iOS 13.

CocoaPods

- Go To Page ⌘ G
- Select All ⌘ A
- Select All Annotations ⌘ A
- Close Tab ⌘ W
- Close All Tabs ⌘ W
- Close All Other Tabs ⌘ W
- Show Next Tab ⌘ →
- Show Previous Tab ⌘ ←

Carthage

```
binary "https://customers.pspdfkit.com/carthage/.../PSPDFKit.json" >= 9.0
```

Manual Integration

For information on how to manually integrate PSPDFKit, see our [Integrating PSPDFKit](#) guide article.

Getting Started. Swift 5

Here is a simple example for using PSPDFKit, first in Swift 5:

```
import PSPDFKit
```

- Back ⌘ ←
- Settings ⌘ ,
- Share ⌘ S
- Annotations ⌘ A
- Add Bookmark ⌘ D
- Search ⌘ F
- Outline ⌘ I



Getting Started. Integration

PSPDFKit 9 for iOS is designed for Xcode 11+ (SDK 13+), and fully supports iOS 11, iOS 12 and iOS 13.

Getting Started. SWIFT 5

Here is a simple example for using PSPDFKit, first in Swift 5:

CocoaPods

Go To Page ⌘ G

Select All ⌘ A

Select All Annotations ⌘ A

Close Tab ⌘ W

Close All Tabs ⌘ W

Close All Other Tabs ⌘ W

Show Next Tab ^ →

Show Previous Tab ^ ⌘ →

Back ⌘ ←

Settings ⌘ ,

Share ⌘ S

Annotations ⌘ A

Add Bookmark ⌘ D

Search ⌘ F

Outline ⌘ I

```
import PSPDFKit
```

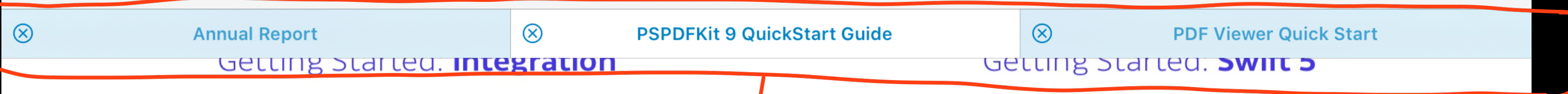
Carthage

```
binary "https://customers.pspdfkit.com/carthage/.../PSPDFKit.json" >= 9.0
```

Manual Integration

For information on how to manually integrate PSPDFKit, see our [Integrating PSPDFKit](#) guide article.





Getting Started. **Integration**

PSPDFKit 9 for iOS is designed for Xcode 11+ (SDK 13+), and fully supports iOS 11, iOS 12 and iOS 13.

Getting Started. **Swift 5**

Here is a simple example for using PSPDFKit, first in Swift 5:

CocoaPods

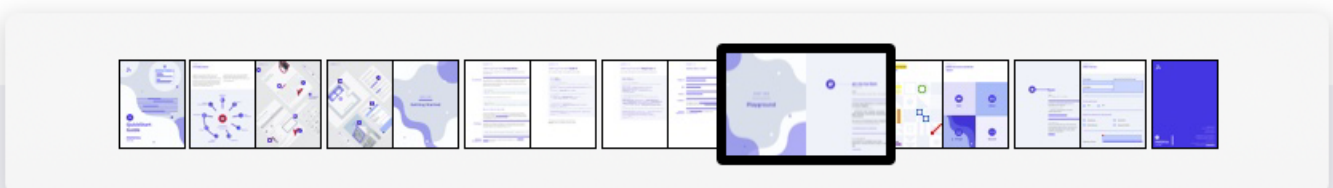
- Go To Page ⌘ G
- Select All ⌘ A
- Select All Annotations ⌘ A
- Close Tab ⌘ W**
- Close All Tabs ⌘ W**
- Close All Other Tabs ⌘ W**
- Show Next Tab ⌘ →
- Show Previous Tab ⌘ ←


Carthage







```
binary "https://customers.pspdfkit.com/carthage/.../PSPDFKit.json" >= 9.0
```

Manual Integration

For information on how to manually integrate PSPDFKit, see our [Integrating PSPDFKit](#) guide article.



Documents 

Annual Report

















PSPDFKit 9 QuickStart Guide

PDF Viewer Quick Start

Getting Started. Integration

PSPDFKit 9 for iOS is designed for Xcode 11+ (SDK 13+), and fully supports iOS 11, iOS 12 and iOS 13.

CocoaPods

- Go To Page   G
- Select All  A
- Select All Annotations   A
- Close Tab  W
- Close All Tabs   W
- Close All Other Tabs   W
- Show Next Tab   
- Show Previous Tab   

Carthage

```
binary "https://customers.pspdfkit.com/carthage/.../PSPDFKit.json" >= 9.0
```










Manual Integration

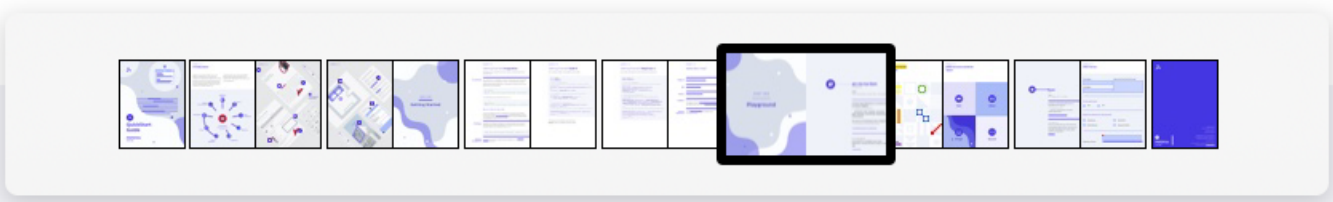
For information on how to manually integrate PSPDFKit, see our [Integrating PSPDFKit](#) guide article.

Getting Started. Swift 5

Here is a simple example for using PSPDFKit, first in Swift 5:

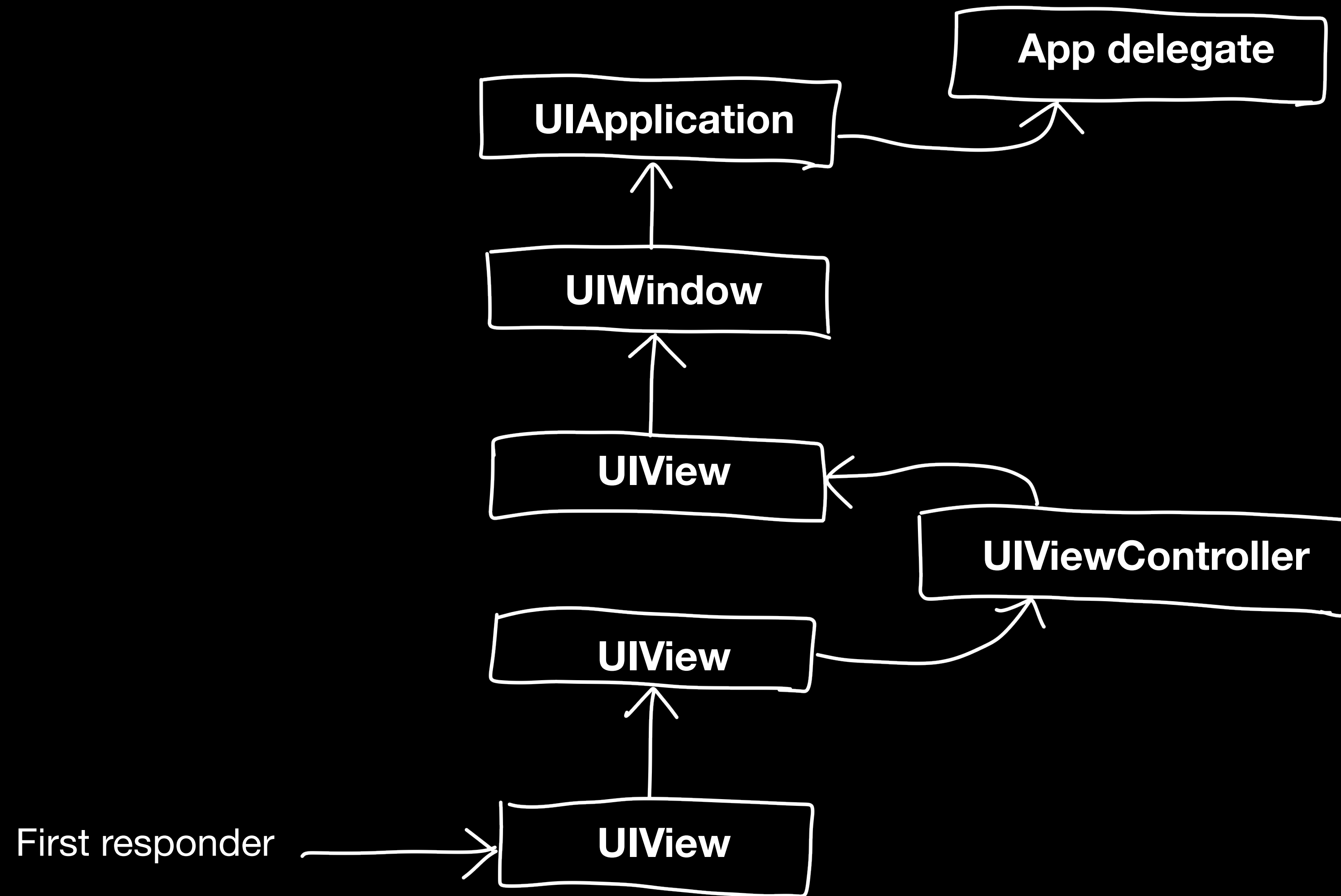
```
import PSPDFKit
```

- Back  
- Settings  ,
- Share  S
- Annotations   A
- Add Bookmark  D
- Search  F
- Outline  I



- How do we provide key commands to the system?
- How does the system decide possible key commands?
- How does a key command run our code?

Responder chain



- API
- User experience
- Implementation

User experience

- Look at AppKit
- Consider discoverability
- Be fast and responsive



Human Interface Guidelines

Overview Resources Videos What's New

- > macOS
- > App Architecture
- > **User Interaction**
 - Authentication
 - Data Entry
 - Drag and Drop
 - File Handling
 - Help
 - Keyboard**
 - Mouse and Trackpad
 - Providing User Feedback
 - Requesting Permission
- > System Capabilities
- > Visual Design
 - > Icons and Images
 - > Windows and Views
 - > Menus
 - > Buttons
 - > Fields and Labels
 - > Selectors
 - > Indicators
 - > Touch Bar
 - > Extensions
- iOS
- tvOS
- watchOS
- > Technologies

Keyboard

The keyboard is an essential input device for entering text, navigating, and initiating actions.



Keyboard-Only Interaction

Some people prefer using a keyboard over a mouse or a trackpad. Others, such as VoiceOver users, need to use the keyboard. To ensure your app can be used by all users, make sure its core features are accessible using the keyboard.

Respect standard keyboard shortcuts and create app-specific shortcuts for frequently used commands. Keyboard shortcuts let people activate menu items and actions by pressing specific key combination.

Add full keyboard access mode support to all custom interface elements. Full keyboard access mode lets users navigate and activate windows, menus, interface elements, and system features using the keyboard alone.

User experience

- Look at AppKit
- Consider discoverability
- Be fast and responsive

[5]. Therefore, a software architect may allocate only 50 microseconds to the gesture recognition task when a new stroke is collected, to ensure that the process is not disruptive to other game tasks. This poses two issues. First, with such a tight constraint, certain recognizers cannot be used because they are simply too slow. Second, this limits the number of templates a recognizer can check during the matching process. Another concern is that recognition latency plays a critical role in user perception and interface design. Gray and Boehm-Davis [4], for instance, have shown that even seemingly negligible latencies (on the order of milliseconds) can influence how users interact with software. Therefore, to address these issues we introduce *Penny Pincher*, an accurate recognizer that achieves high accuracy through speed. This \$-family extension is designed to do the absolute minimum amount of work possible to match candidate gestures with templates so that more templates can be evaluated within the same amount of time as other recognizers. As a result, our recognizer is significantly more accurate than its kindred.

Most other recognizers use Euclidean distance to compare gestures whereas we accomplish our goal by comparing between-point vectors, see Figure 1. As do other recognizers, we resample the input candidate stroke to a constant number of points; however, we avoid rotating, scaling, and translating the gesture. The only limitation of this choice is that our recognizer is not rotation invariant, but with enough templates loaded, this is not an issue. Further, after resampling the input, we use only addition and multiplication operations; that is, we also avoid any calls to computationally expensive geometric functions, such as \cos or $\sqrt{\cdot}$. In Section 3 we explain Penny Pincher in detail. Our evaluation in Section 4 shows that although our method is deceptively simple, we still achieve high accuracy for a variety of datasets with a limited number of templates and superior accuracy under an imposed time constraint. Finally we conclude the paper in Sections 5 and 6.

2 RELATED WORK

Since Rubine's [13] seminal work on gesture recognition, numerous techniques have been developed for both gesture and symbol recognition. One popular branch of research focuses on easy to understand and easy to implement recognizers. The \$-family of recognizers started with Wobbrock et al. [17] who designed the \$1 recognizer. This simple and effective unistroke recognizer works by resampling strokes to a predetermined number of points, and then these strokes are translated and scaled to the unit square centered at zero. Euclidean distance between points is then used as a similarity metric to compare candidate strokes with template gesture strokes. The template with the least distance is assumed to be the most likely gesture. To ensure the best match possible, \$1 also uses a golden section search (GSS) [11] to rotate candidate strokes into alignment with the template under consideration. Anthony and Wobbrock [2] extended the \$1 unistroke recognizer into the \$N multistroke recognizer. This version is able to handle multistroke gestures by combining all strokes into a single stroke so that \$1 techniques can be leveraged. Given an example gesture, during training when templates are constructed, all permutations of strokes and stroke directions are used to generate all possible ways in which the gesture can be written.

While the previous approaches work with arcs, Vatavu et al. [16] considers input as point clouds. The \$P recognizer still resamples, scales, and translates strokes and uses Euclidean distances as a similarity metric. However, the points are not treated as a time series; instead the authors employ a greedy algorithm that tests different permutations of points to find the overall minimum distance between candidate and template point clouds. Herold and Stahovich [6], on the other hand, continue to work with strokes as time series, but their 1^{ϵ} recognizer generates a rotation invariant representation that is a time series vector of one-dimensional points—normalized

distances measured from the stroke's centroid. That is, strokes are resampled as usual, but then each two-dimensional point is converted into a one-dimensional scalar, the distance of the point from the centroid normalized by the standard deviation of all distances. Candidate and template strokes are then compared using the one-dimensional Euclidean distance.

One major limitation of \$1, \$N, and \$P is that they are generally slow. The former two recognizers are slow due to their use of the GSS. Li [9] was able to overcome this limitation. He discovered a closed form equation to find the optimal rotation of a candidate stroke to match the template. Since Li's Protractor recognizer represents each stroke as a vector and works with the angle between the two vectors (between the candidate and template vectors), it is not necessary to scale candidate strokes when matching. However, resampling and translation to the centroid are still required. Anthony and Wobbrock [3] adapted Protractor's technique to create a fast multistroke recognizer, \$N-Protractor.

To further speed up template matching, Reaver et al. [12] proposed Quick\$ to use agglomerative hierarchical clustering with a

Select All	⌘	A
Clear Selection		esc
Close Tab	⌘	W
Close All Tabs	⇧ ⌘	W
Close All Other Tabs	⌘	W
Show Next Tab	⇧	→
Show Previous Tab	⇧ ⇧	→

$$g = \{g_i = (x_i, y_i) \in \mathbb{R}^2\}, \quad (1)$$

where i indexes the gesture's points in time relative order. Gestures are resampled by length into n points so that the distance between each point is equal. A gesture is then converted into a series of between-point vectors:

$$v = \{v_i = g_{i+1} - g_i\}, \quad (2)$$

for $i < n$. Given two gestures v and w , their similarity is taken to be the sum of the angles between corresponding between-point vectors:

$$D(v, w) = \sum_{i=1}^{n-1} \frac{v_i \cdot w_i}{\|v_i\| \|w_i\|}. \quad (3)$$

A perfect score is therefore $n - 1$ because the normalized dot product is 1 for identical vectors. Now, let m be the number of distinct

gestures and \mathcal{T} the set of templates created by training the recognizer:

$$\mathcal{T} = \{t_i = (s_i, l_i) \mid l_i \in \{1 \dots m\}\}, \quad (4)$$

where t_i is the enumerate template in set \mathcal{T} , s_i is the gesture sample's between-point vector, and l_i is the gesture's class label. A candidate gesture c takes the classification of the template T that is most similar:

$$T = \arg \max_{t_i \in \mathcal{T}} D(t_i, c), \quad (5)$$

This already simple calculation can be simplified further. During training, let the components of the between-point vectors be normalized so that each $\|s_i\| = 1$, thus eliminating one normalization factor in Equation 3. Next, as a simplifying assumption, say that the length of each between-point vector component is equal (each $\|w_i\| = \|w_{i+1}\|$) because gestures are resampled into $n - 1$ equidistance arc lengths; this assumption is evaluated in Section

Settings	⌘	,
Outline	⌘	I
Search	⌘	F
Add Bookmark	⌘	D
Annotations	⇧ ⌘	A
Share	⌘	S
Back	⌘	◀

boundary of the \$-family of recognizers. Without the use of filtering or other culling techniques we make the following assumptions:

1. Candidate gestures are always resampled.
2. The resampled candidate gesture is always compared against every template in \mathcal{T} .
3. Each point in the resampled candidate gesture is evaluated against at least one corresponding template point.

Under these assumptions, let $r = |c|$ be the length of the candidate gesture before resampling, representing the number of raw data points. As before, n is the number of resampled data points. The cost of resampling c is the cost of first calculating the stroke length and then computing the resample points. This requires touching every raw data point twice, $\Omega(2r)$, and every resample point once, $\Omega(n)$. The latter is because each resample point is part of the

path and must be considered as the resampling procedure continues. Supposing that the recognizer can match the resampled points directly (e.g., without further manipulation), then no additional processing is necessary. Template matching is subsequently carried out for each $t \in \mathcal{T}$, which is $\Omega(n|\mathcal{T}|)$. Therefore, at best, a template based recognizer as described is bounded by:

$$\Omega(2r + n + n|\mathcal{T}|). \quad (7)$$

To the best of our knowledge, Penny Pincher is the first \$-family recognizer to achieve this lower bound. Note that the complexity of several other recognizers are available in [16], however, these do not consider the cost of resampling. Of course Ω notation hides certain details that are important in practice. We have to consider that 1^{ϵ} , for example, represents each point in its template as a one-dimensional point where the other methods use two-dimensional points. This means that 1^{ϵ} can compare templates faster despite having a non optimal resampling strategy as is shown in the evaluation Section 4.3. Penny Pincher, on the other hand, relies on straightforward dot product calculations without having to utilize external or built-in math libraries.

4 EVALUATION

We evaluate Penny Pincher using three different tests. All tests were performed on a MacBook Pro with a 2.4 GHz Intel Core i7 processor, 8 GB of 1333 MHz DDR3 memory, and an 760 GB mechanical SATA disk. In our first test we check the validity of the assumption that all between-point vector lengths of a single gesture are of equal length. Next we evaluate the accuracy of our method compared to other \$-family recognizers when the number of templates is controlled (the standard method of evaluation). Finally we investigate the accuracy of the fastest recognizers to see how well they perform under varying time constraints. However, we first describe the various datasets used in our tests in the follow subsections.

4.1 Datasets

This subsection gives a brief description of each dataset used in our evaluation. For additional information beyond what is presented here, we refer the reader to the associated cited work.

\$1-GDS. The \$1 recognizer gesture dataset [17] is a unistroke gesture dataset comprising 16 gestures collected from 10 subjects at 3 speeds (slow, medium, and fast) on an HP iPAQ h4334 Pocket PC. Because each subject supplied 10 samples of each gesture at each speed, there are 4800 samples in aggregate.

SIGN. The Synchronmedia-Imadoc Gesture New On-Line Database [1] is a unistroke gesture dataset comprising 17 classes collected from 20 writers on tablet PCs and whiteboards. Each writer supplied 25 samples over four sessions so that SIGN contains 8500 samples in aggregate.

EDS 1 and EDS 2. Two unistroke gesture datasets were collected by Vatavu et al. [15] to study gesture execution difficulty, referred to as Execution Difficulty Set #1 (EDS 1) and Execution Difficulty Set #2 (EDS 2). The first dataset contains 5040 samples in aggregate comprising 18 gestures collected from 14 participants providing 20 samples each. The latter dataset contains 4400 samples in aggregate comprising 20 gestures from 11 participants providing 20 samples each. All samples were collected on a Wacom DTU-710 Interactive Display.

MMG. The Mixed Multistroke Gestures dataset [3] comprises 16 gestures having between one and three strokes. Samples were collected from 20 participants using either a stylus or their finger on a tablet PC at three different speeds (slow, medium, and fast). In aggregate there are 9600 samples.

UJI. The UJIPenchars2 Database [10] is a multistroke gesture dataset of lowercase and uppercase letters, digits, characters with diacritics, and punctuation marks containing an aggregate of 11640



PDF Viewer

File

Edit

View

Annotate

Go

Window

Help

Show Tab Bar

Show All Tabs

⌘⇧\

Single Page

⌘1

Two Pages

⌘2

Automatic based on Window Size

⌘3

Thumbnails

⌘4

Document Editor

⌘5

Page Transition



Scroll Direction



Page Fitting



Appearance



Open New Windows as Tabs

Zoom In

⌘+

Zoom Out

⌘-

Show Annotation Toolbar

⌘⇧A

Show Sidebar

⌘⇧B

Hide Toolbar

⌘⇧T

Customize Toolbar...

Enter Full Screen

⌘⇧F

User experience

- Look at AppKit
- Consider discoverability
- Be fast and responsive

What to Expect From Marzipan • The Breakroom

It's clear that this year's WWDC is going to be a doozy. We've written here previously with our thoughts about [Dark Mode](#), now it's time to talk about iOS apps coming to the Mac.

Of course I'm talking about Marzipan, a technology Apple introduced with few details during last year's Keynote. We knew that some apps in Mojave used the new technology and that was about it.



The iOS version of Twitterrific running on macOS thanks to Marzipan.

Thanks to the [hard work of Steve Troughton-Smith](#), we have a much clearer view of what's happening behind the scenes with application architecture and available APIs.

What I'm going to focus on today is how this new technology will affect product development, design, and marketing. I see many folks who think this transition will be easy: my experience tells me that it will be more difficult than it appears at first glance.

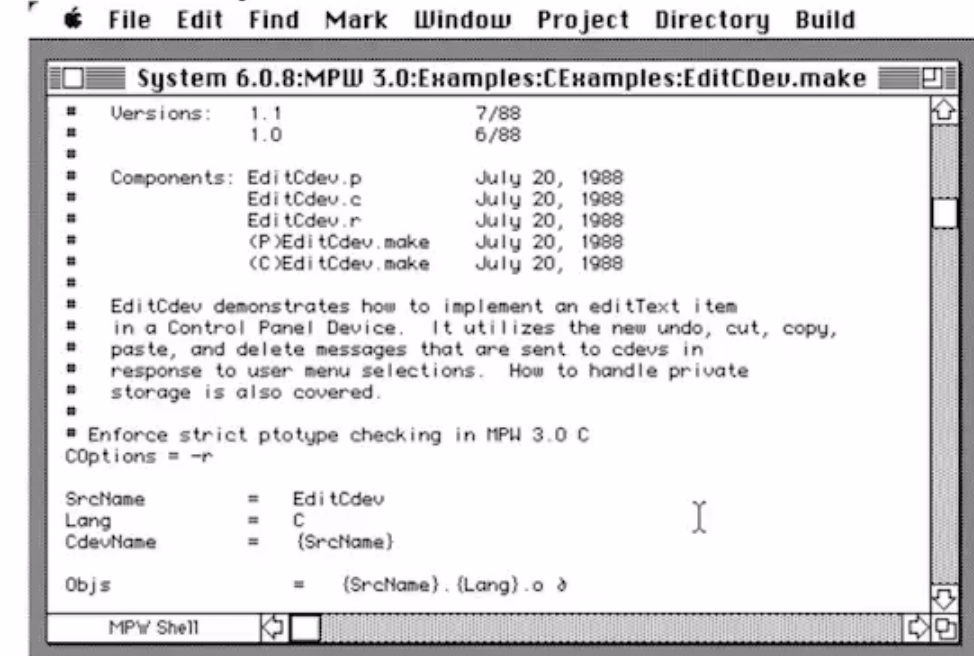
For the past year, I've been working on a new product that runs natively on three operating systems: iOS, tvOS, and macOS. As a result, I feel

like I can talk with some authority on the differences between these platforms. My biggest takeaway from this project is how different interaction models have a ripple effect throughout a product.

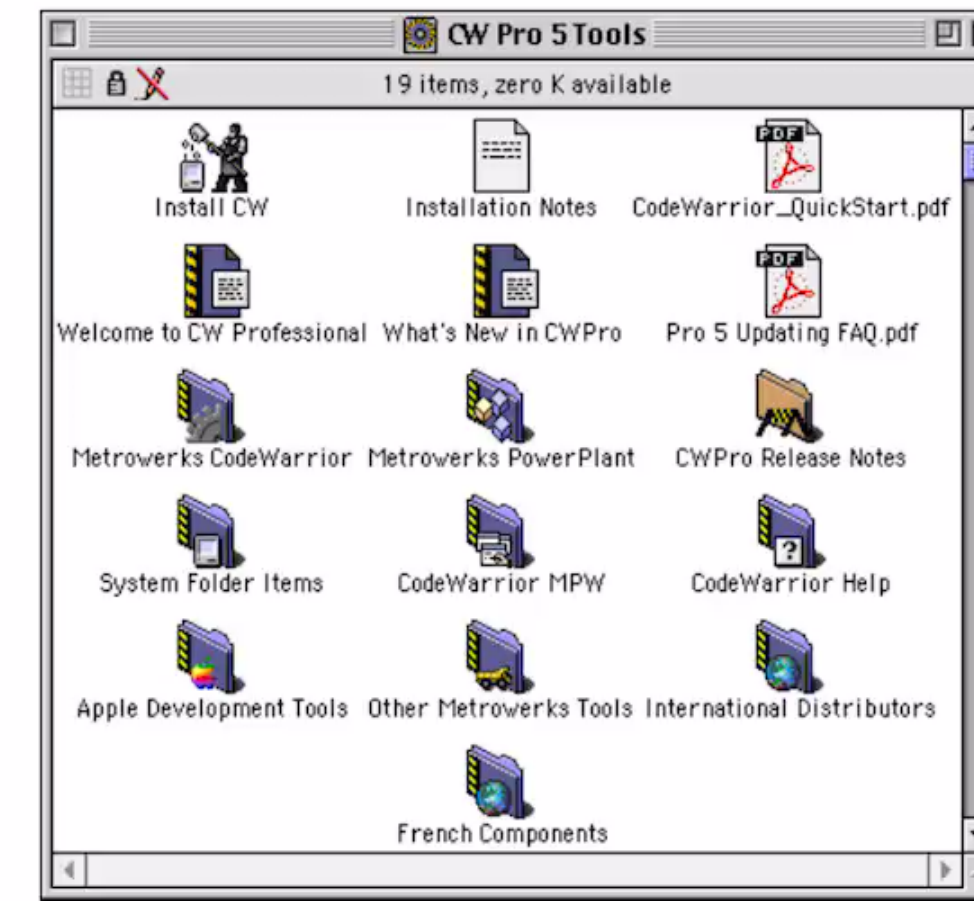
(We're not ready to talk about this project in public yet, but [Patreon supporters](#) are getting sneak peeks.)

A Little Bit of History

Before we get into the interaction details, let's take a look at tool and framework transitions on the Macintosh. Marzipan may be something new, but it's certainly not a first.



I'm old enough to have been around when the Mac first shipped: in 1984 you could buy some floppy disks to write code in Pascal or 68K assembly code. After a few years, the way you made these revolutionary computer interfaces evolved into MPW, the [Macintosh Programmer's Workshop](#).

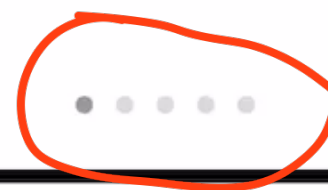


Fast forward a few years and things had moved on to a new language and tools. Many folks were using [CodeWarrior](#) and C++ to build their Mac apps. I developed the Iconfactory's [first software product](#) using this environment.

The modern era then arrived with [NeXTSTEP](#) in the year 2000. It brought an unfamiliar language called Objective-C, a new IDE called Project Builder, and awesome new frameworks named Foundation and AppKit. These tools have served the Mac well for many years and even provided a foothold for iOS with UIKit in 2007.

And now a modern macOS toolkit will include Swift, UIKit, and Marzipan. Or, as we like to call it, [Chameleon 2.0](#) :-)

As you can see, the stuff that you use to build Mac apps has changed radically over the past 35 years – but it's important to note what hasn't changed during that same period of time. You still use a bitmap display for output with a keyboard and mouse for input. The interaction model has changed, but only slightly. There are



- API
- User experience
- Implementation

Build up components

- UITabBarController
- UINavigationController
- UITableViews

Build up components

- UITabBarController
- UINavigationController
- UITableView

Today ⌘ 1
Games ⌘ 2
Apps ⌘ 3

Updates ⌘ 4
Search ⌘ 5

If the NoZoom flag is set, the annotation shall always maintain the same fixed size on the screen and shall be unaffected by the magnification level at which the page itself is displayed.

In either case, the annotation's position shall be determined by the coordinates of the upper-left corner of its annotation rectangle, as defined by the Rect entry in the annotation dictionary and interpreted in the default user space of the page.

NOTE Figure 58 shows how an annotation whose NoRotate flag is set remains upright when the page it is on is rotated 90 degrees clockwise.

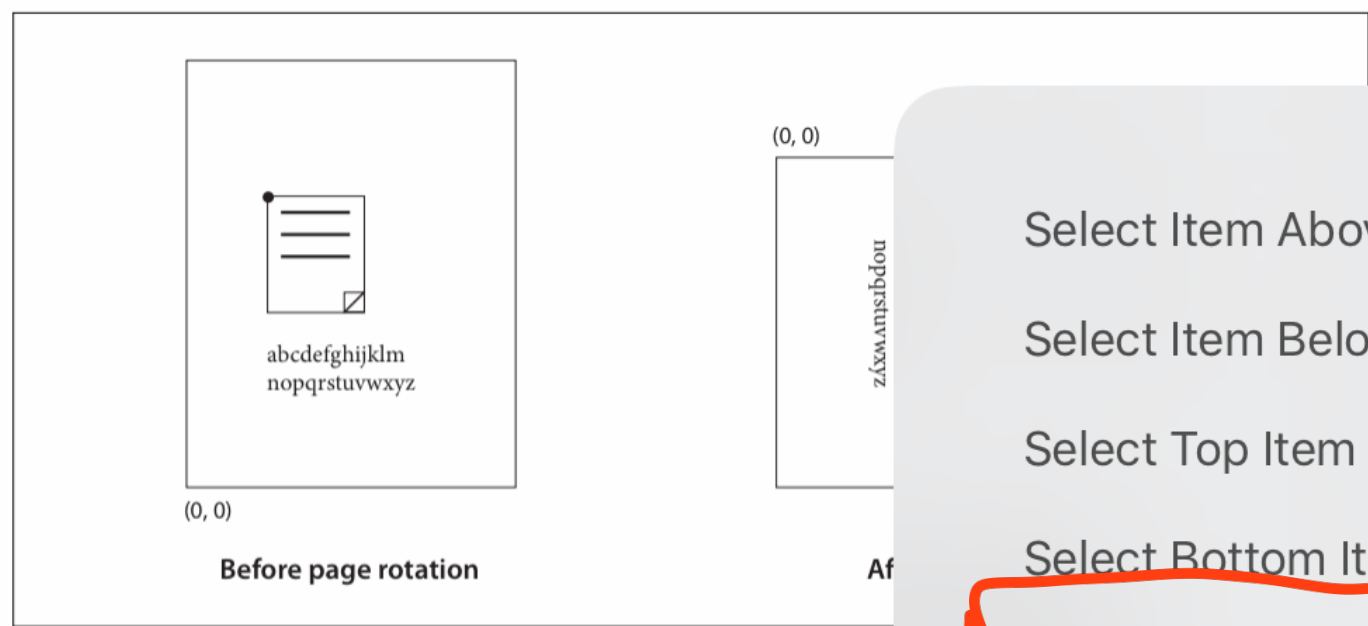


Figure 58 – Coordinate adjustment with the page

12.5.4 Border Styles

An annotation may optionally be surrounded by a border when displayed or printed. If present, the border shall be drawn completely inside the annotation rectangle.

Table 166 – Entries in a border style dictionary

Table with 3 columns: Key, Type, Value. Rows include Type (name) and W (number).

Table 166 – Entries in a graphics state parameter dictionary

Table with 3 columns: Key, Type, Value. Rows include S (name) and D (array).

Context menu with options: Select Item Above, Select Item Below, Select Top Item, Select Bottom Item, Outline, Bookmarks, Annotations, Info, Add Bookmark.

Table of contents for PDF 32000-1:2008, listing sections like Graphics State Parameter Dictionary, QuadPoints, Metadata Streams, and Document info.

12.5.5 Appearance Streams

Beginning with PDF 1.2, an annotation may specify one or more appearance streams as an alternative to the simple border and colour characteristics available in earlier versions.

The algorithm outlined in this sub-clause shall be used to map from the coordinate system of the appearance XObject to the annotation's rectangle in default user space:

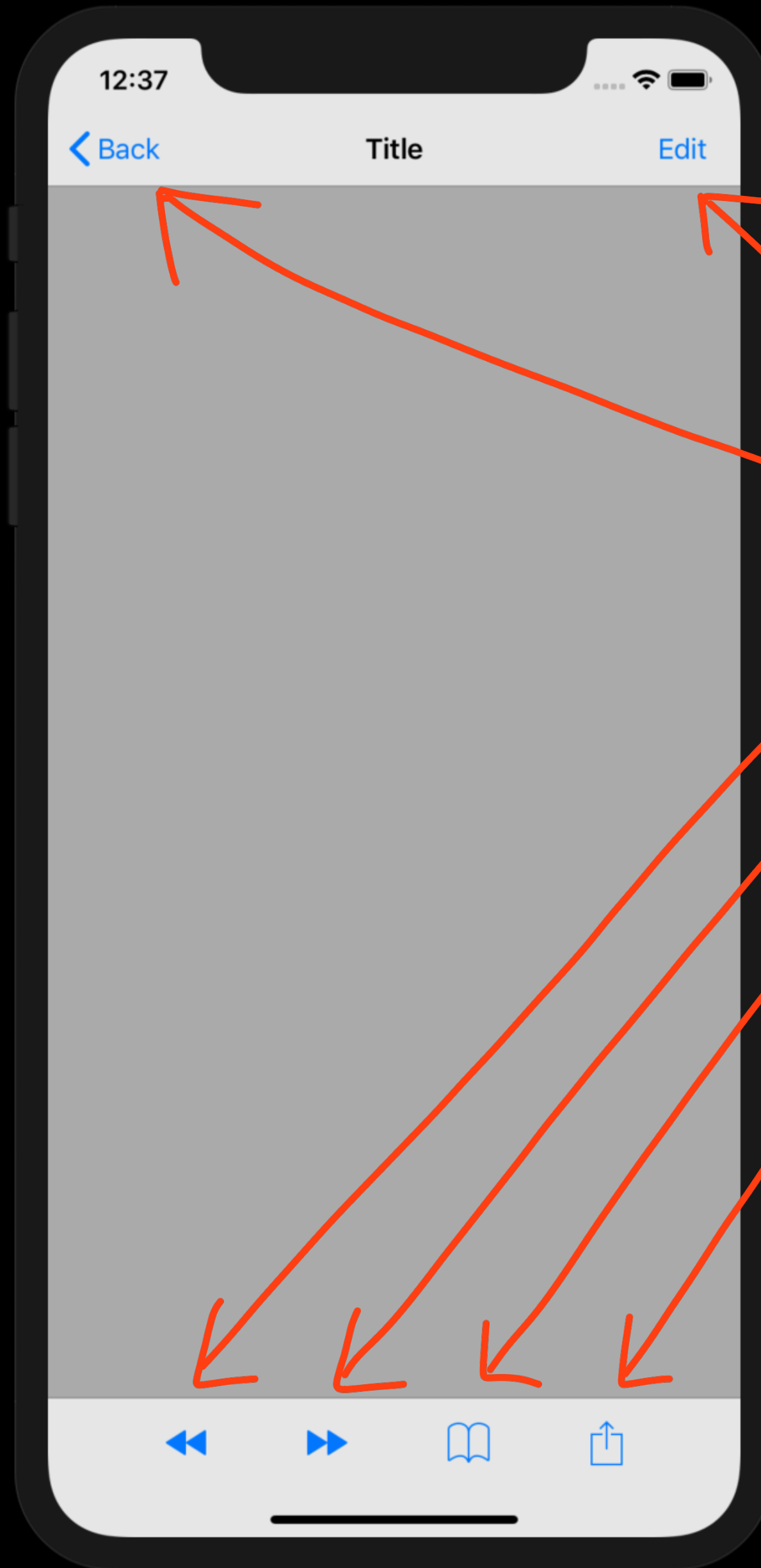


```
class KeyboardTabBarController: UITabBarController {  
  
    override var keyCommands: [UIKeyCommand]? {  
        var commands = super.keyCommands ?? []  
  
        if let items = tabBar.items {  
            commands += items.enumerated().map { index, tabBarItem in  
                UIKeyCommand(title: tabBarItem.title ?? "",  
                    action: #selector(scrollToNumberedTab),  
                    input: String(index + 1),  
                    modifierFlags: .command)  
            }  
        }  
    }  
  
    return commands  
}  
}
```

```
@objc func scrollToNumberedTab(_ sender: UIKeyCommand) {  
    guard let keyInput = sender.input,  
          let targetTabNumber = Int(keyInput) else {  
        return  
    }  
  
    selectedIndex = targetTabNumber - 1  
}
```

Build up components

- UITabBarController
- UINavigationController
- UITableView



Actions

```

class KeyboardNavigationController: UINavigationController {

    override var keyCommands: [UIKeyCommand]? {
        var commands = super.keyCommands ?? []

        if let topViewController = topViewController {
            let canGoBack: Bool = // ...
            if (canGoBack) {
                commands += [UIKeyCommand(input: "[",
                                           modifierFlags: .command,
                                           action: #selector(goBackFromKeyCommand))]
            }

            let navItem = topViewController.navigationItem
            let commandFromBarItem: (UIBarButtonItem) -> UIKeyCommand? = {
                // Convert from UIBarButtonItem to UIKeyCommand...
            }
            commands += navItem.leadingBarButtonItems.compactMap(commandFromBarItem)
            commands += navItem.trailingBarButtonItems.compactMap(commandFromBarItem).reversed()
            commands += topViewController.toolbarItems.compactMap(commandFromBarItem)
        }
        return commands
    }
}

```

```
class KeyboardBarButtonItem: UIBarButtonItem {  
    var keyEquivalentInput: String?  
    var keyEquivalentModifierFlags: UIKeyModifierFlags = []  
}
```

Build up components

- UITabBarController
- UINavigationController
- UITableView

PDF 32000-1:2008

The **Domain**, **Encode**, and **Size** entries determine how the function's input variable values are mapped into the sample table. For example, if **Size** is [21 31], the default **Encode** array shall be [0 20 0 30], which maps the entire domain into the full set of sample table entries. Other values of **Encode** may be used.

To explain the relationship between **Domain**, **Encode**, **Size**, **Decode**, and **Range**, we use the following notation:

$$y = \text{Interpolate}(x, x_{\min}, x_{\max}, y_{\min}, y_{\max})$$

$$= y_{\min} + \left((x - x_{\min}) \times \frac{y_{\max} - y_{\min}}{x_{\max} - x_{\min}} \right)$$

For a given value of x , Interpolate calculates the y value on the line defined by the two points (x_{\min}, y_{\min}) and (x_{\max}, y_{\max}) .

Table 39 – Additional entries specific to a type 0 function dictionary

Key	Type	Value
Size	array	<i>(Required)</i> An array of m positive integers that shall specify the number of samples in each input dimension of the sample table.
BitsPerSample	integer	<i>(Required)</i> The number of bits that shall represent each sample. (If the function has multiple output values, each one shall occupy BitsPerSample bits.) Valid values shall be 1, 2, 4, 8, 12, 16, 24, and 32.
Order	integer	<i>(Optional)</i> The order of interpolation between samples. Valid values shall be 1 and 3, specifying linear and cubic spline interpolation, respectively. Default value: 1.
Encode	array	<i>(Optional)</i> An array of $2 \times m$ numbers specifying the linear mapping of input values into the domain of the function's sample table. Default value: [0 (Size ₀ - 1) 0 (Size ₁ - 1) ...].
Decode	array	<i>(Optional)</i> An array of $2 \times n$ numbers specifying the linear mapping of sample values into the range appropriate for the function's output values. Default value: same as the value of Range .
<i>other stream attributes</i>	(various)	<i>(Optional)</i> Other attributes of the stream that shall provide the sample values, as appropriate (see Table 5).

When a sampled function is called, each input value x_i , for $0 \leq i < m$, shall be clipped to the domain:

$$x'_i = \min(\max(x_i, \text{Domain}_{2i}), \text{Domain}_{2i+1})$$

That value shall be encoded:

$$e_i = \text{Interpolate}(x'_i, \text{Domain}_{2i}, \text{Domain}_{2i+1}, \text{Encode}_{2i}, \text{Encode}_{2i+1})$$

That value shall be clipped to the size of the sample table dimension:

$$e'_i = \min(n,$$



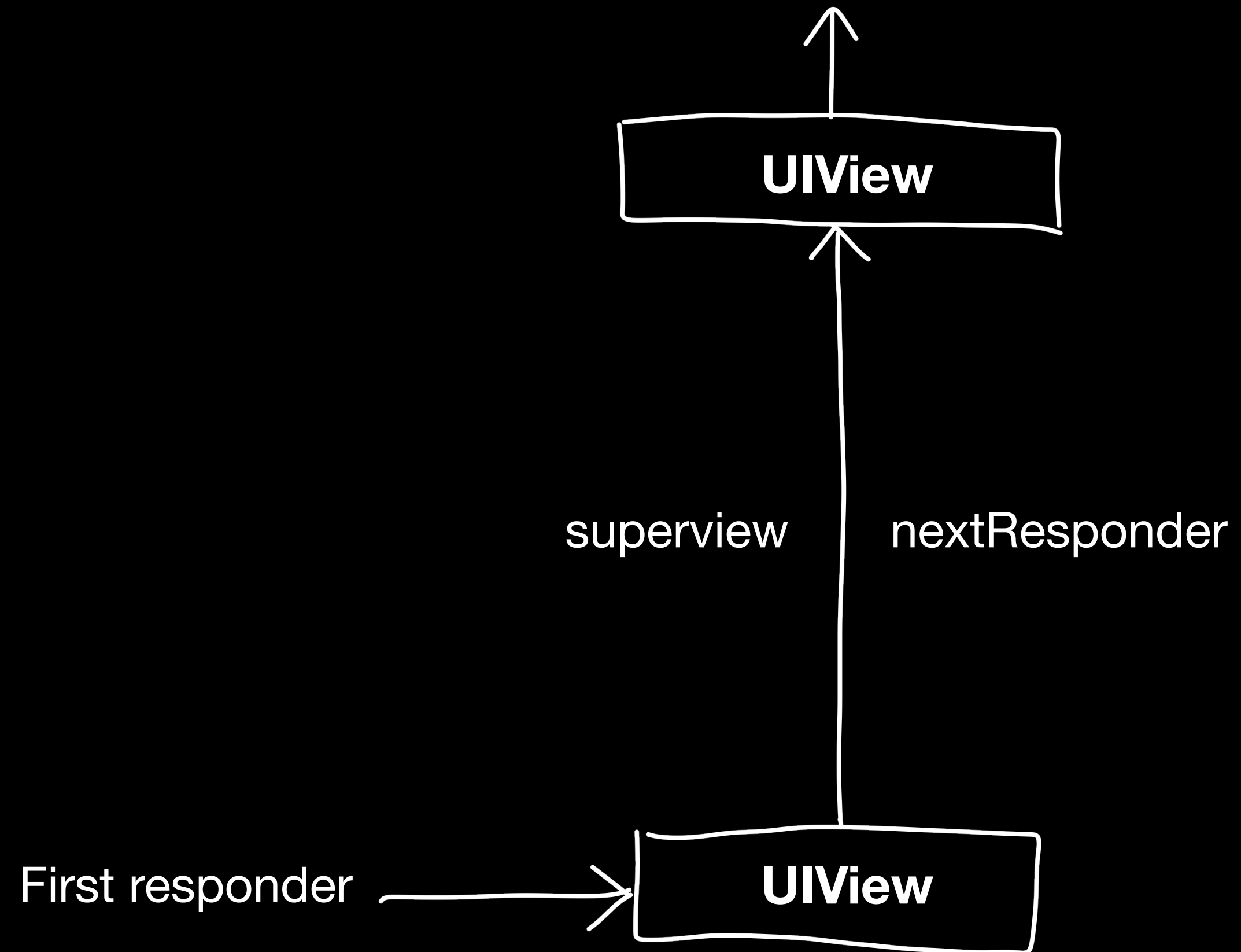
The encoded input value e'_i is used to determine output values from the nearest surrounding values in the sample table. Each output value r_j , for $0 \leq j < n$, shall then be decoded.

```
override var keyCommands: [UIKeyCommand]? {  
    var commands = super.keyCommands ?? []  
    commands += [  
        UIKeyCommand(input: UIKeyCommand.inputUpArrow,  
            modifierFlags: [],  
            action: #selector(selectAbove)),  
        UIKeyCommand(input: UIKeyCommand.inputDownArrow,  
            modifierFlags: [],  
            action: #selector(selectBelow)),  
        UIKeyCommand(input: "\r",  
            modifierFlags: [],  
            action: #selector(activateSelection)),  
    ]  
    return commands  
}
```

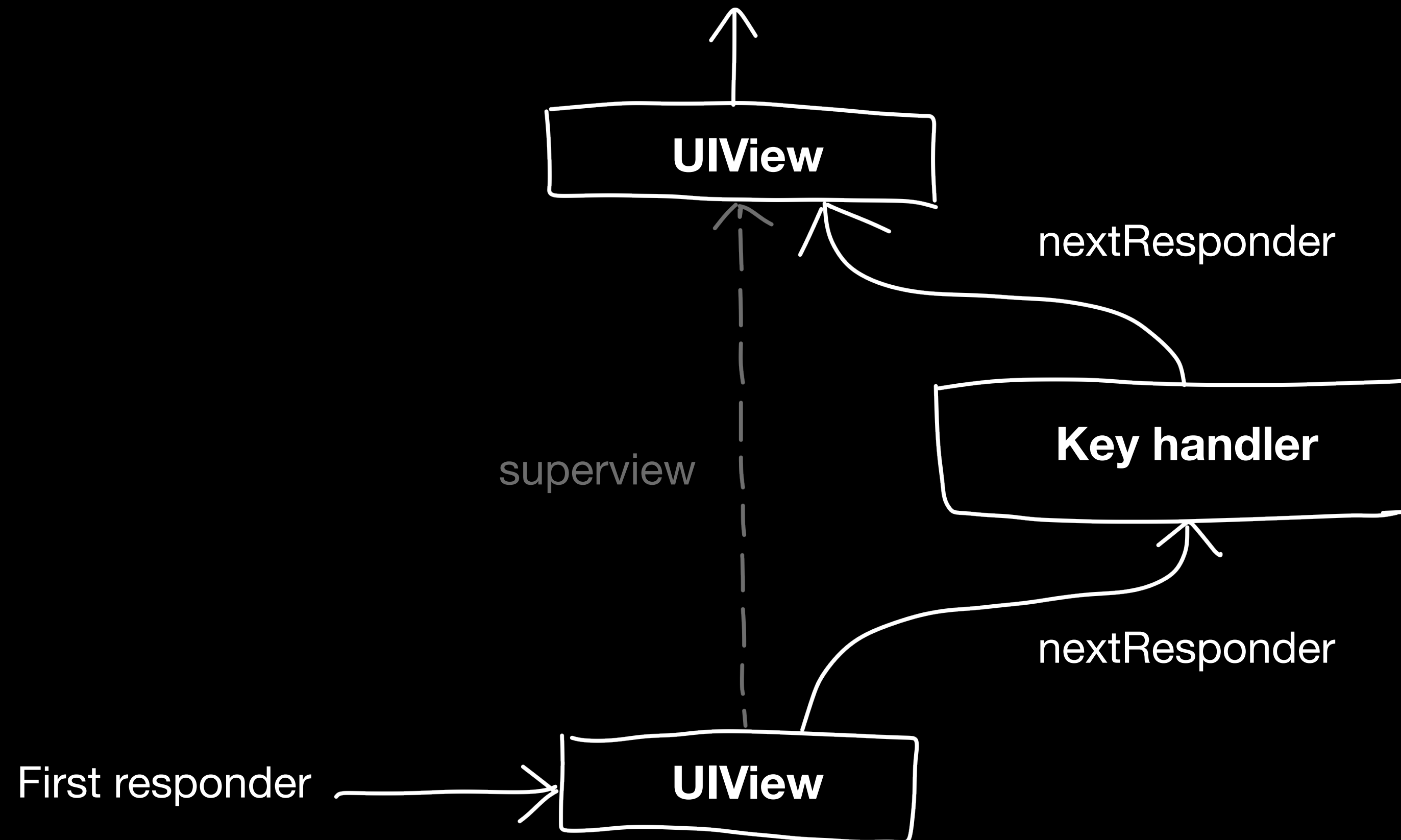
```
// Update the table view selection programatically.  
selectRow(at: indexPath, animated: false, scrollPosition: .none)  
scrollToRow(at: indexPath, at: scrollPosition, animated: true)
```

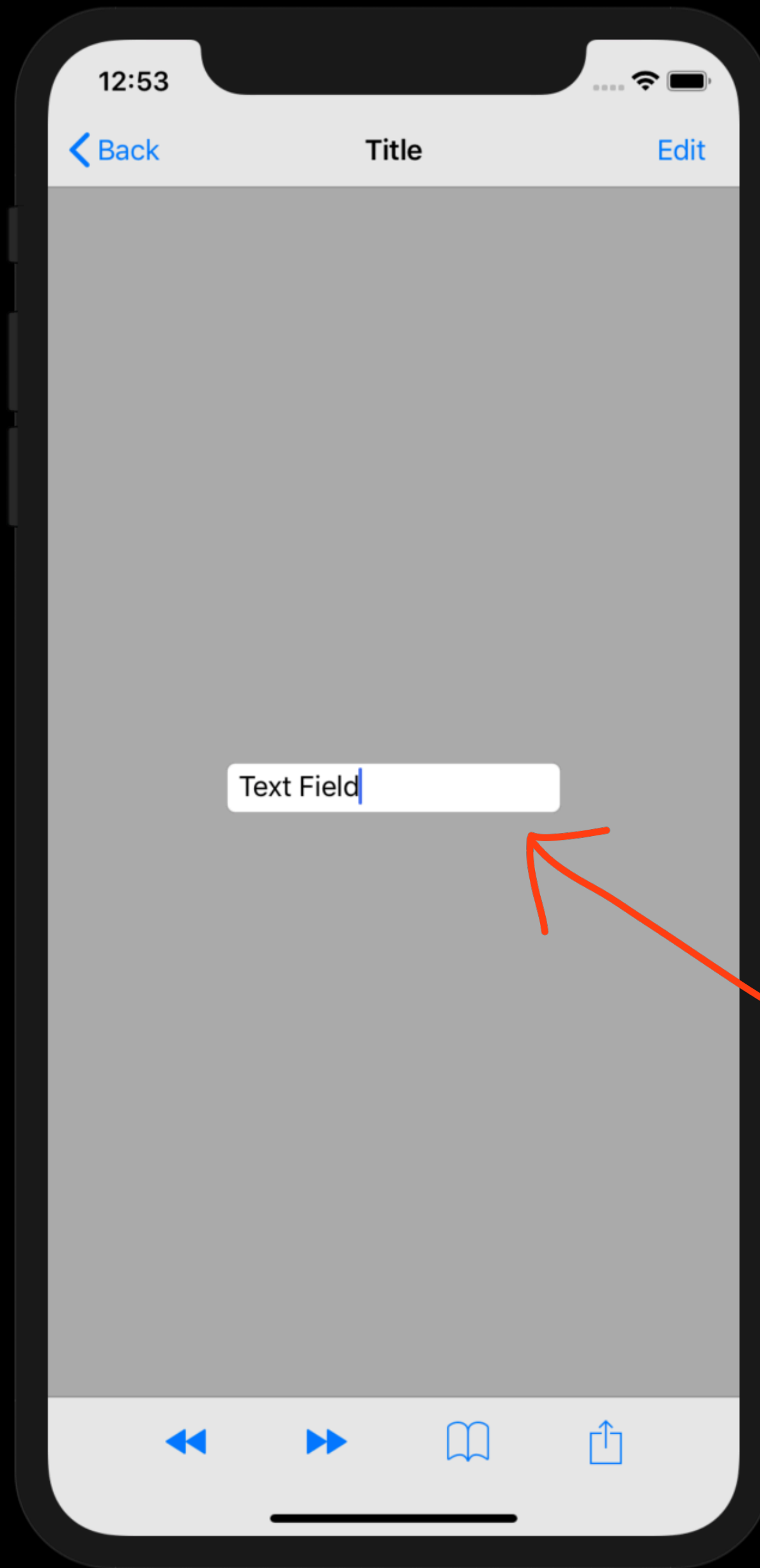
```
@objc func activateSelection() {  
    guard let indexPathForSelectedRow = indexPathForSelectedRow else {  
        return  
    }  
    delegate?.tableView?(self, didSelectRowAt: indexPathForSelectedRow)  
}
```


Responder chain injection



Responder chain injection





First responder



HUMAN RESOURCES MANAGEMENT SYSTEM

NAME

John
 LAST
 Appleseed
 FIRST
 MIDDLE SUFFIX PREFIX
 Mr

ADDRESS

7440-7498 S Hanna St
 STREET
 Fort Wayne IN
 CITY STATE ZIP

TELEPHONE

(555) 123-1234 (555) 666-7777
 HOME WORK

EMERGENCY CONTACT INFORMATION

(555) 444-3333
 PHONE
 Elizabeth Chapman Sister
 NAME RELATIONSHIP

SOCIAL SECURITY NO. (MANDATORY)

111-12-1111

SEX

MALE
 FEMALE

DATE OF BIRTH

08/26/1774
 MONTH/DAY/YEAR

EDUCATION (HIGHEST LEVEL AND YEAR)

- HIGH SCHOOL DIPLOMA
- TRADE CERTIFICATE
- COLLEGE - NO DEGREE
 - ASSOCIATE'S DEGREE
 - BACHELOR'S DEGREE
 - MASTER'S DEGREE
 - PROFESSIONAL DEGREE
- PH.D.
- OTHER DOCTORATE

John Appleseed

EMPLOYEE SIGNATURE

Digitally signed by John Appleseed
 DN: cn=John Appleseed, o=GB, email=john@domain.com
 Date: 2013.12.19 12:45:26 Z.

December 2019

Day Week **Month** Year



Mon	Tue	Wed	Thu	Fri	Sat	Sun
						1 1. Advent
2 Bank Holiday (Scotland) • Android Team Hangout 13:00 • iOS Team Hangout 13:00 • Web Team Hangout 13:00 • Team Hangout 14:00 • Core Hangout 14:30 • Marketing Team Hangout 14:30	3	4	5	6 Nikolo	7	8 2. Advent Mariä Empfängnis
9 • Android Team Hangout 13:00 • iOS Team Hangout 13:00 • Web Team Hangout 13:00 • Team Hangout 14:00 • Core Hangout 14:30 • Marketing Team Hangout 14:30	10	11	12 Jonathan's Birthday	13 Experimental Friday	14	15 3. Advent
16 • Android Team Hangout 13:00 • iOS Team Hangout 13:00 • Web Team Hangout 13:00 • Team Hangout 14:00 • Core Hangout 14:30 • Marketing Team Hangout 14:30	17	18	19	20	21	22 4. Advent
23 • Android Team Hangout 13:00 • iOS Team Hangout 13:00 • Web Team Hangout 13:00 • Team Hangout 14:00 • Core Hangout 14:30 • Marketing Team Hangout 14:30	24 Christmas Eve Heiligabend	25 Christmas Day Weihnachtstag	26 Boxing Day Stefanitag	27 Marketing Friday	28	29

Today

Calendars

Inbox

```
if UIResponder.isTextInputActive {  
    keyCommands = keyCommands.filter { keyCommand in  
        keyCommand.doesConflictWithTextInput == false  
    }  
}
```



Search or jump to...



Pull requests Issues Marketplace Explore



douglashill / KeyboardKit

Unwatch 2

Star 180

Fork 3

Code

Issues 0

Pull requests 0

Actions

Projects 0

Wiki

Security

Insights

Settings

A framework that makes it easy to add hardware keyboard control to iOS and Mac Catalyst apps.

Edit

Manage topics

74 commits

1 branch

0 packages

2 releases

2 contributors

MIT

Branch: master

New pull request

Create new file

Upload files

Find file

Clone or download



douglashill Fix ScrollViewKeyHandler swallowing unused actions

Latest commit 8bc8b74 7 days ago

KeyboardKit.xcodeproj	Add responder chain debugging helper	7 days ago
KeyboardKit	Fix ScrollViewKeyHandler swallowing unused actions	7 days ago
KeyboardKitDemo	Add KeyboardTableViewController and KeyboardCollectionViewController	14 days ago
.gitignore	Add generated string keys file and add resources for CocoaPods	13 days ago
Keyboard-Kit.podspec	Update version to 1.0.1	12 days ago
License.txt	Add initial KeyboardKit project	last month
README.md	Add deletion in table views using UITableViewCellEditingStyleDelete	11 days ago

README.md

KeyboardKit

KeyboardKit makes it easy to add hardware keyboard control to iOS and Mac Catalyst apps.

Keyboard control is a standard expectation of Mac apps. It's important on iOS too because a hardware keyboard improves speed and ergonomics, which makes an iPad an even more powerful productivity machine.

Apps created with AppKit tend to have better support for keyboard control compared to UIKit-based apps. I believe the principal reason for this is that most AppKit components respond to key input out of the box, while most UIKit components do not. KeyboardKit aims to narrow this gap by providing subclasses of UIKit components that respond to key commands.

Status

This project is relatively new and is under active development. So far these components are available:

- KeyboardTableView (Controller): A UITableView (Controller) subclass that supports navigation and selection using arrow keys and space or return, including wrapping back to the top/bottom and selecting the top/bottom item

KeyboardKit

- Extends UIKit components to add keyboard control in a reusable way
- User experience closely modelled on AppKit
- Designed to be fast and responsive
- Avoids subclassing internally
- Filters out text input conflicts
- Page Up, Page Down, Home, End

Keyboard control in UIKit apps

Douglas Hill • @qdoug
iOS Conf SG 2020