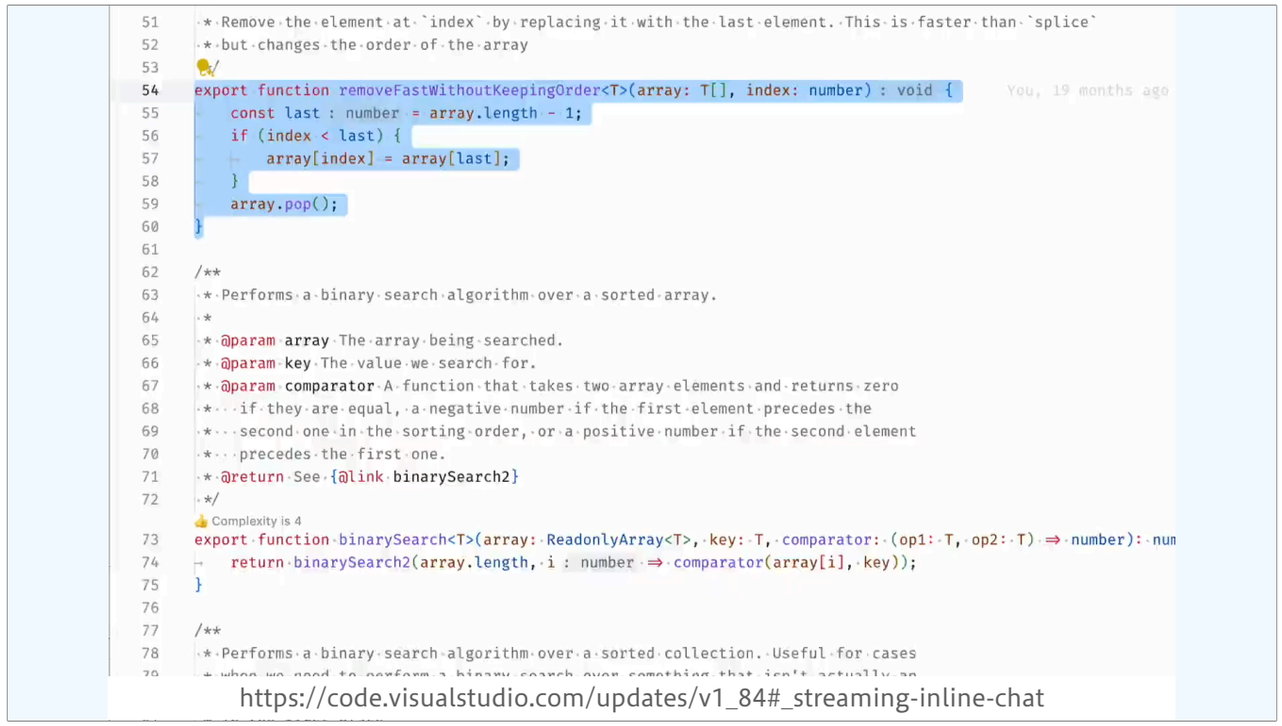
Enhancing Applications with Accessibility API

kishikawa katsumi

@kishikawakatsumi@hachyderm.io
kishikawakatsumi

Hi, I'm katsumi. Today, I'm excited to talk to you about how to add AI assistant functionality to existing applications using the Accessibility API.



https://code.visualstudio.com/updates/v1_84#_streaming-inline-chat

It has become normal to use AI assistants for programming tasks. This video is a demonstration of GitHub Copilot on VS Code.
AI returns very accurate results for programming tasks such as explaining functions,

and refactoring.
Human programmers can save a lot of time by taking advantage of AI.
Have you ever wanted to be able to use an AI assistant not only in a specific IDE, but anywhere, for example, your favorite text editor or a web browser?
In this session, I will present core technologies to realize such a request.

Integrate AI Assistants Into Any Apps

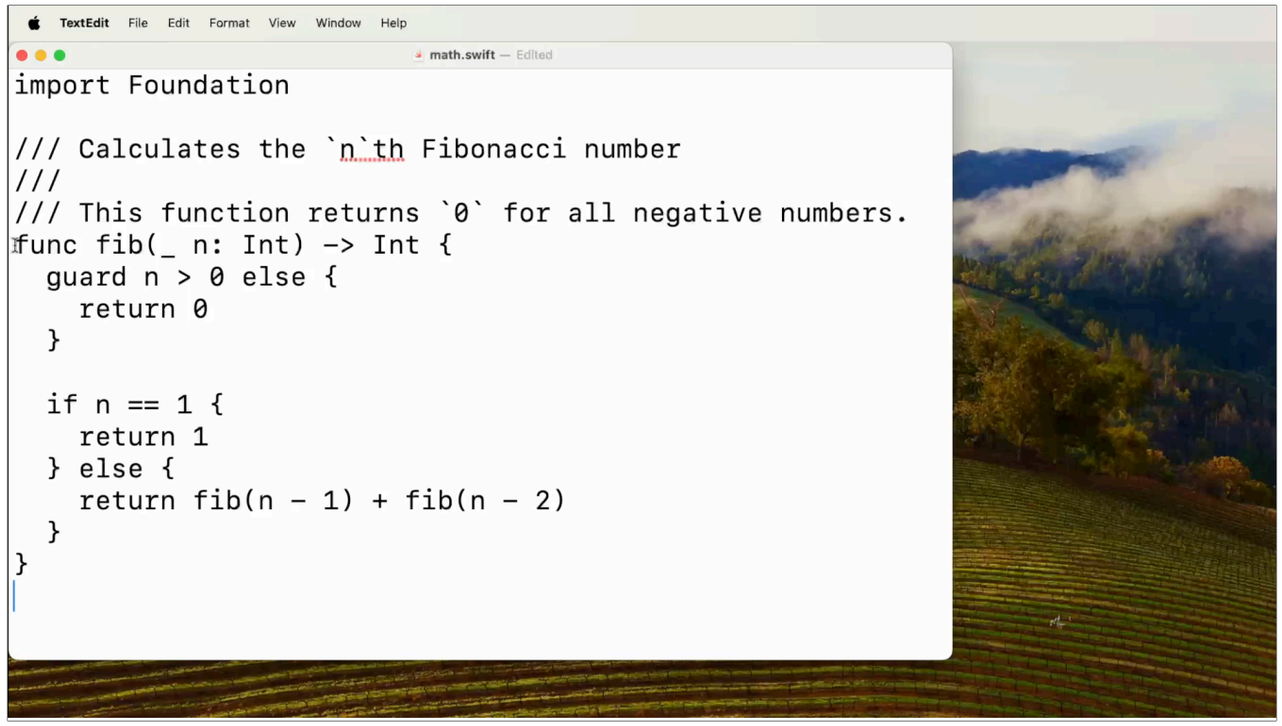
- In your favorite text editor
- In your web browser
- Or comic readers

Let's get started.
I am going to explain how to integrate AI assistant functionality into existing applications.

Sample Code

➡️ https://github.com/kishikawakatsumi/tryswift2024

In this session I will show four sample applications. These will illustrate how to use the Accessibility API. They are all available as sample code in my GitHub repository.



This video demonstrates the first application.
Select text and press the keyboard shortcut to ask the AI about the selected text.
The application shown is TextEdit.app, which is pre-installed on the Mac. Of course, it does not have a plug-in mechanism.
As in the VS Code example shown earlier, I let AI refactor the selected code.

Core Technologies

- Retrieve selected text for any application
- Monitor global key & mouse events

One of the core technologies required to achieve such functionality is to retrieve text displayed by other applications.
Another is the monitoring of global key and mouse events to trigger this functionality at arbitrary timing, such as when a specific shortcut is pressed.

Accessibility API

- Allows access to UI elements of the system or other applications
- Enables manipulation of UI elements of other applications

The Accessibility API makes the impossible possible. It to be used in testing and automation software, allows to access and manipulate UI elements of the system and other applications.

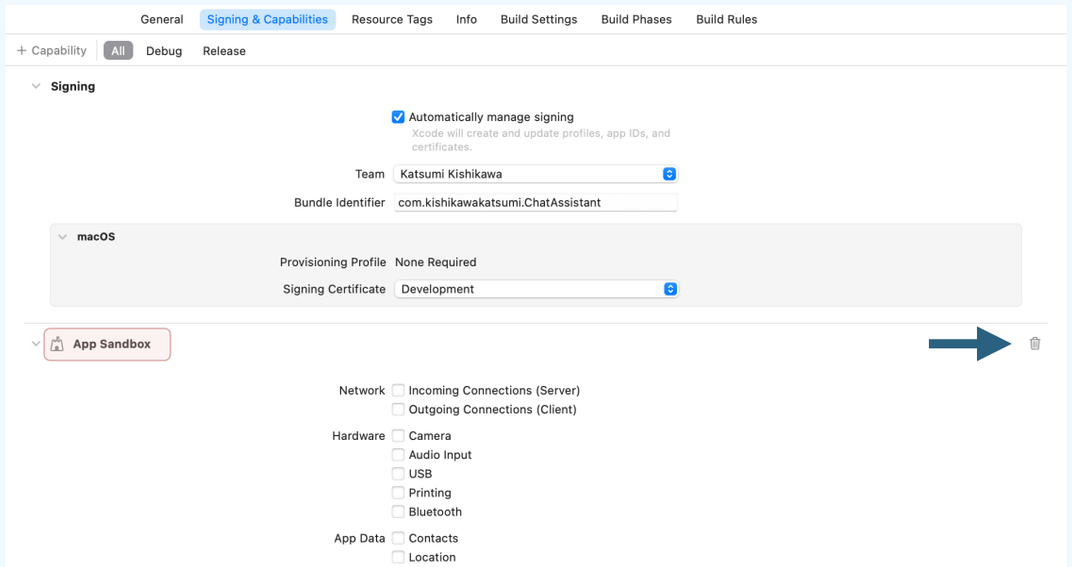
Enabling Accessibility API

Sandbox and Privacy Settings

- Disable Sandbox
- Grant Accessibility API Permissions

It is very powerful, so, to use the Accessibility API, you need to fulfill the prerequisites.
The Accessibility API can read information from other applications. To prevent abuse, applications must be explicitly granted permission by the end user.
Also, the sandbox must be disabled.
In summary, to use the Accessibility API, two steps are required in advance: disabling the sandbox and granting permission to the application by the end user.

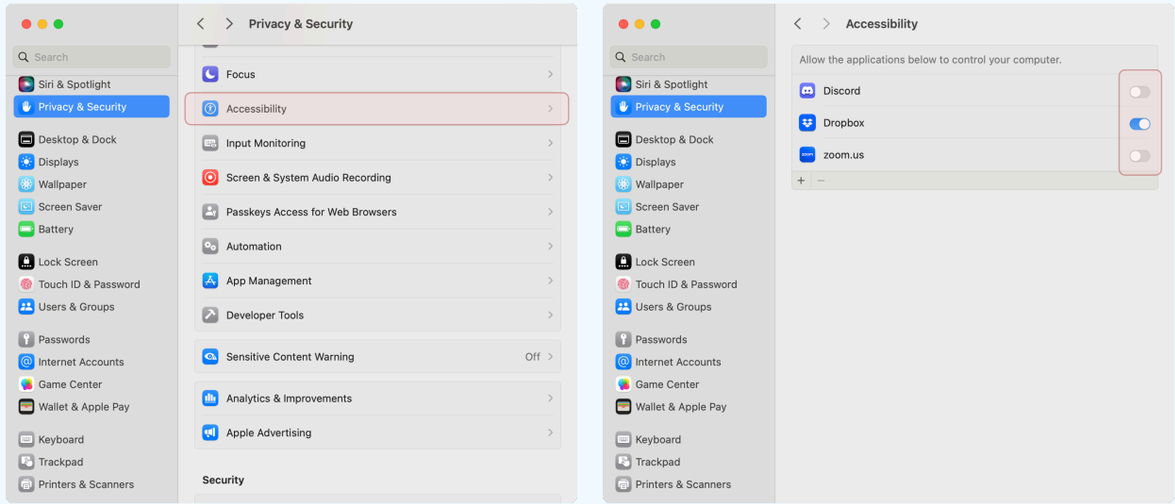
Disable Sandbox



To disable the sandbox, remove the “App Sandbox” section, by pressing the trash button on the “Signing & Capabilities” tab, in Xcode.

Grant Accessibility Permission

Privacy Settings



To grant permissions for the Accessibility API, add the application for which you want to grant permissions to “Accessibility”, under “Privacy & Security” in “System Settings”. Then, turn on the toggle switch.

Grant Accessibility Permission

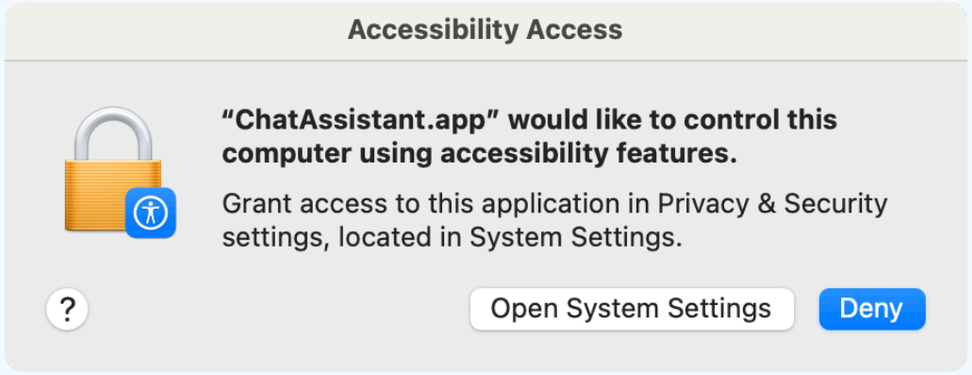
AXISProcessTrustedWithOptions(_:)

```
func AXIsProcessTrustedWithOptions(_ options: CFDictionary?) -> Bool
                                   [kAXTrustedCheckOptionPrompt: true]
```

It is not easy for users to add applications to “Privacy & Security”, which is located DEEP in the “System Settings”.
The Accessibility API provides functions to help with that.
The `AXISProcessTrustedWithOptions` function does two things: it adds the application to the list of “Accessibility” settings, in the “Privacy & Security” settings.

Grant Accessibility Permission

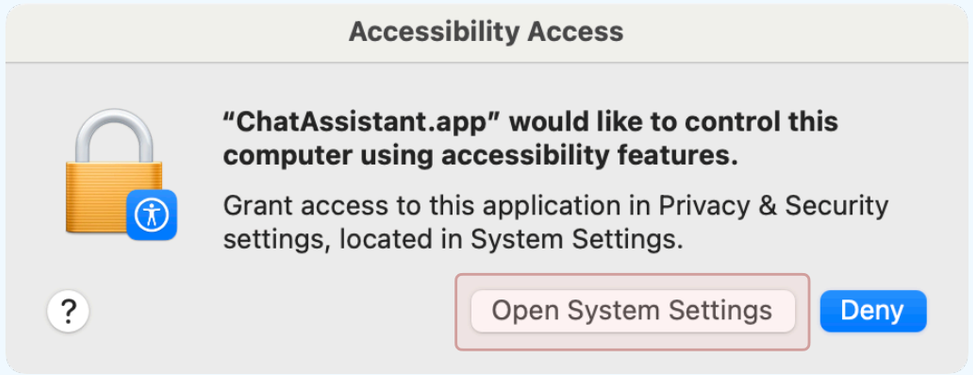
AXISProcessTrustedWithOptions(_:)



It also displays a dialog message, prompting the user to grant that permission,

Grant Accessibility Permission

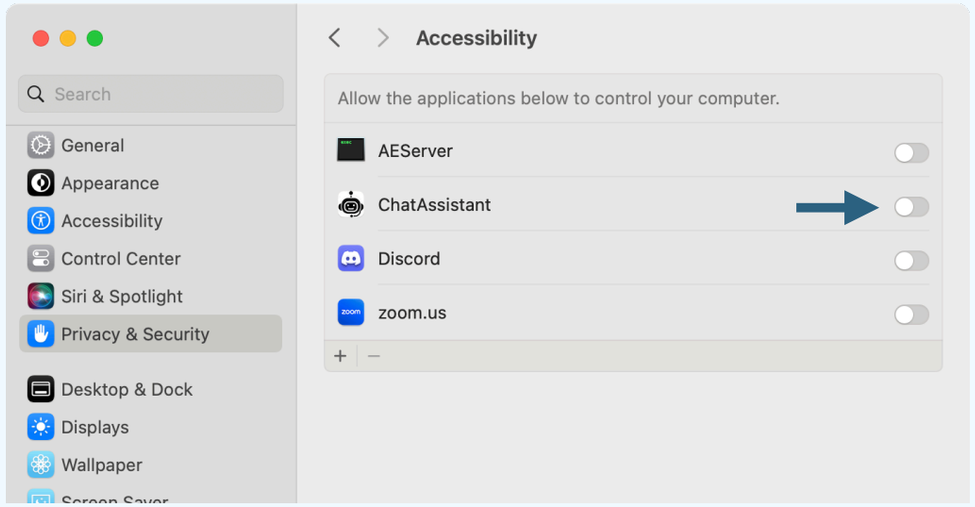
AXISProcessTrustedWithOptions(_:)



and allows the user to open the settings directly from the “Open System Settings” button.

Grant Accessibility Permission

AXISProcessTrustedWithOptions(_:)



The "Open System Settings" button opens the “System Settings” as shown.
All that's left is, the user simply turns on the toggle switch.

Basic Usage

AXUIElementCopyAttributeValue(_:_:_:)

```
func AXUIElementCopyAttributeValue(
    _ element: AXUIElement,
    _ attribute: CFString,
    _ value: UnsafeMutablePointer<CFTypeRef?>
) -> AXError
```

Let's take a look at the basic usage of the Accessibility API.

You can get the value of any UI element using the `AXUIElementCopyAttributeValue` function.

Basic Usage

AXUIElementCopyAttributeValue(_:_:_:)

```
func AXUIElementCopyAttributeValue(
    _ element: AXUIElement,
    _ attribute: CFString,
    _ value: UnsafeMutablePointer<CFTypeRef?>
) -> AXError
```

The UI element containing the value you want to retrieve

The first argument is the UI element containing the value you want to retrieve. The type of object that can be passed is `AXUIElement`. `AXUIElement` is a proxy object for the actual UI element.

Basic Usage

AXUIElementCopyAttributeValue(_:_:_:)

```
func AXUIElementCopyAttributeValue(
    _ element: AXUIElement,
    _ attribute: CFString,
    _ value: UnsafeMutablePointer<CFTypeRef?>
) -> AXError
```

Key of the value you want to retrieve

The second argument is a key string that is the name of the value, you want to retrieve.

Basic Usage

AXUIElementCopyAttributeValue(_:_:_:)

```
func AXUIElementCopyAttributeValue(
    _ element: AXUIElement,
    _ attribute: CFString,
    _ value: UnsafeMutablePointer<CFTypeRef?>
) -> AXError
```

Value you want to retrieve

The last argument is the return value; since the Accessibility API is written in the C language, the return value is passed as an argument with a pointer.

Basic Usage

AXUIElementCopyAttributeValue(_:_:_:)

```
func AXUIElementCopyAttributeValue(
    _ element: AXUIElement,
    _ attribute: CFString,
    _ value: UnsafeMutablePointer<CFTypeRef?>
) -> AXError
```

Error code

The return value indicates the kind of error.

Retrieve Selected Text

kAXSelectedTextAttribute

```
var selectedTextValue: AnyObject?
let selectedTextValueError = AXUIElementCopyAttributeValue(
  focusedElement as! AXUIElement,
  kAXSelectedTextAttribute as CFString,
  &selectedTextValue
)
guard let selectedTextValue, selectedTextValueError == .success else {
  return
}
```

Let's look at a real-world example.

Retrieve Selected Text

kAXSelectedTextAttribute

```
var selectedTextValue: AnyObject?
let selectedTextValueError = AXUIElementCopyAttributeValue(
  focusedElement as! AXUIElement,
  kAXSelectedTextAttribute as CFString,
  &selectedTextValue
)
guard let selectedTextValue, selectedTextValueError == .success else {
  return
}
```

To retrieve the selected text, pass the `kAXSelectedTextAttribute` constant as the key.

Retrieve Selected Text

kAXSelectedTextAttribute

```
var selectedTextValue: AnyObject?
let selectedTextValueError = AXUIElementCopyAttributeValue(
  focusedElement as! AXUIElement,
  kAXSelectedTextAttribute as CFString,
  &selectedTextValue
)
guard let selectedTextValue, selectedTextValueError == .success else {
  return
}
```

In this code, the `AXUIElement` passed as the first argument is, the UI element that currently has focus.

Let me explain where this object came from.

`AXUIElement` is a tree structure similar to an actual UI element.

To traverse an `AXUIElement`, the root `AXUIElement` is required.

System Wide AXUIElement

AXUIElementCreateSystemWide()

```
let systemWideElement: AXUIElement = AXUIElementCreateSystemWide()
```

In this case, I created the system-wide `AXUIElement` as a root object, because I want it to work with a various applications, not just a specific application.

The system-wide `AXUIElement` is created with the `AXUIElementCreateSystemWide()` function.

System Wide AXUIElement

- Access system-wide keyboard focus, mouse cursor position, and other information.
- Access system-level UI elements that do not belong to a specific application (e.g., menu bar, notification center).
- Retrieve information about the context of the current user interaction, such as the active window or the UI element that currently has focus.

It provides an access point to system-wide accessibility information. In other words, `AXUIElement` objects created using this function are not limited to individual applications or processes, but allow references to any UI element on macOS.

Retrieve Focused UI Element

kAXFocusedUIElementAttribute

```
let systemWideElement = AXUIElementCreateSystemWide()

var focusedElement: AnyObject?

let focusedElementError = AXUIElementCopyAttributeValue(
  systemWideElement,
  kAXFocusedUIElementAttribute as CFString,
  &focusedElement
)
```

So, the focused `AXUIElement` can be retrieved from the system-wide `AXUIElement`.

The constant, that indicates the focused `AXUIElement` is, `kAXFocusedUIElementAttribute`.

This is the basic way to traverse the `AXUIElement` tree structure.

Retrieve Selected Text

kAXSelectedTextAttribute

```
var selectedTextValue: AnyObject?
let selectedTextValueError = AXUIElementCopyAttributeValue(
  focusedElement as! AXUIElement,
  kAXSelectedTextAttribute as CFString,
  &selectedTextValue
)
guard let selectedTextValue,
  selectedTextValueError == .success
else {
  return
}
```

Check the return value to see if the value was retrieved. A return value other than “`.success`” indicates failure for various reasons. For example, the sandbox is enabled, or an attempt to retrieve a value that is not contained, and so on.

```
/// Calculates the `n`th Fibonacci number
///
/// This function returns `0` for all negative numbers.
func fib(_ n: Int) -> Int {
  guard n > 0 else {
    return 0
  }

  if n == 1 {
    return 1
  } else {
    return fib(n - 1) + fib(n - 2)
  }
}
```

Next, monitor shortcut key events so that they can be triggered at any time.
A prompt input window appears with the shortcut key.

Monitor Global Key Event

NSEvent.addGlobalMonitorForEvents(matching:handler:)

```
open class NSEvent : NSObject, NSCopying, NSCoding {

  open class func addGlobalMonitorForEvents(
    matching mask: NSEvent.EventTypeMask,
    handler block: @escaping (NSEvent) -> Void
  ) -> Any?

}
```

There are several ways to monitor events globally.

First, I introduce `addGlobalMonitorForEvents`, which is a class method of `NSEvent`.

It can receive keystrokes and mouse events generated by other applications.

Monitor Global Key Event

```
NSEvent.addGlobalMonitorForEvents(
  matching: [.keyDown]
) { (event) in
  switch event.type {
  case .keyDown:
    guard
      event.modifierFlags.contains([.command, .control])
        && event.keyCode == kVK_ANSI_A
    else {
      return
    }

    ...

  default:
    break
  }
}
```

I want to react to keyboard shortcuts, monitor the `keydown` event.

Monitor Global Key Event

```
NSEvent.addGlobalMonitorForEvents(
  matching: [.keyDown]
) { (event) in
  switch event.type {
  case .keyDown:
    guard
      event.modifierFlags.contains([.command, .control])
        && event.keyCode == kVK_ANSI_A
    else {
      return
    }

    ...

  default:
    break
  }
}
```

Then, check the `modifierFlags` and `keyCode` properties to evaluate the shortcut key combination.

Monitor Global Key Event (Carbon API)

InstallEventHandler(...), RegisterEventHotKey(...)

- InstallEventHandler(_:_:_:_:_:_:)
- RegisterEventHotKey(_:_:_:_:_:_:)
- Carbon API
- **No accessibility permission required**

➡️ https://github.com/kishikawakatsumi/tryswift2024

As another way to set global shortcut keys, `InstallEventHandler` and `RegisterEventHotKey` functions.
This is Carbon API.
Although outdated, this API has the characteristic that it does not require accessibility permissions.
The code is too long to include in the slides, but there is sample code in my GitHub repository. Please see the details.

Monitor Global Key Event

Event Tap

```
let mask: CGEventMask = (1 << CGEventType.keyDown.rawValue)
let tap = CGEvent.tapCreate(
  tap: .cgSessionEventTap,
  place: .headInsertEventTap,
  options: .defaultTap,
  eventsOfInterest: mask,
  callback: { (proxy, type, event, refcon) in
    ...
  },
  userInfo: UnsafeMutableRawPointer(Unmanaged.passUnretained(self).toOpaque())
)
let runLoopSource = CFMachPortCreateRunLoopSource(kCFAllocatorDefault, tap, 0)
CFRunLoopAddSource(CFRunLoopGetCurrent(), runLoopSource, .commonModes)

CGEvent.tapEnable(tap: tap!, enable: true)
```

➡️ https://github.com/kishikawakatsumi/tryswift2024

Another API is Event Tap.
This is an API that allows you to literally “tap” events.

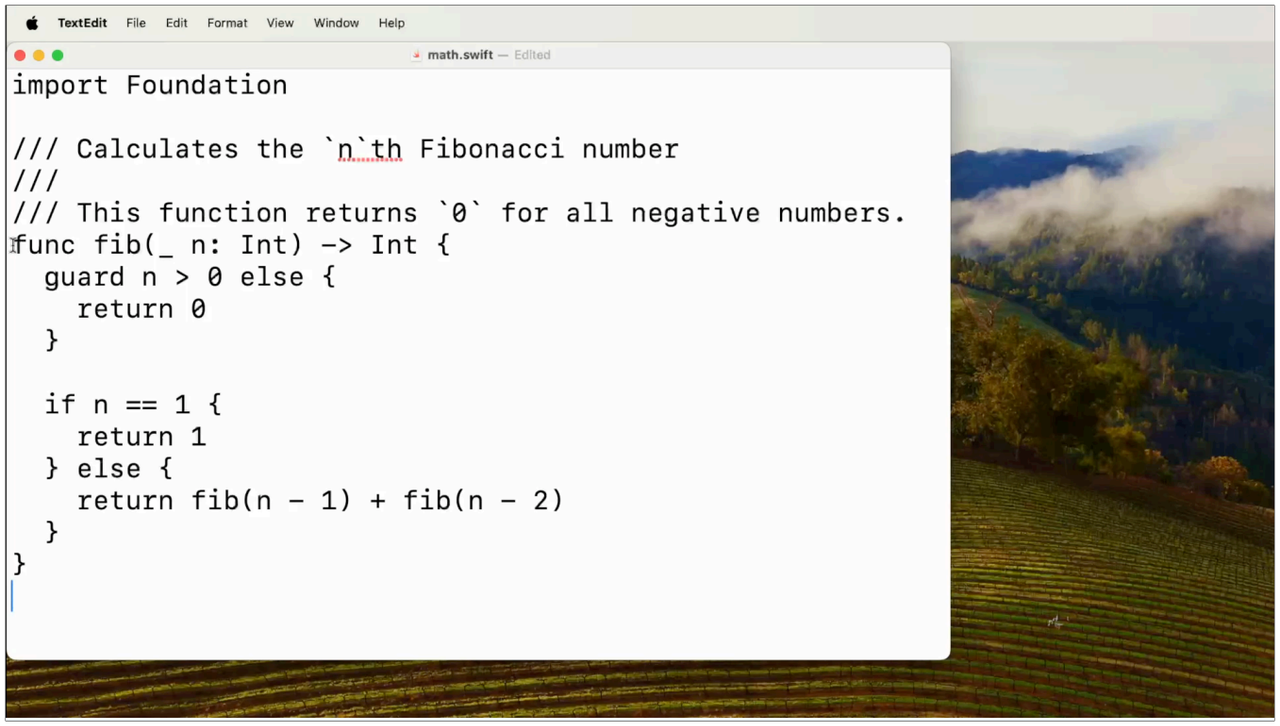
Monitor Global Key Event

Event Tap

- Can stop event propagation
- Can modify event

➡️ https://github.com/kishikawakatsumi/tryswift2024

This API can stop the propagation of events or modify events if necessary.
This is what the word “Tap” indicates.
See the sample code for more details.



So far, the selected text has been retrieved by pressing the shortcut key.
The remaining work is just to show a window to enter the prompts and request the OpenAI web API.
I won't explain about how to code UI and web API. Please see the sample code for details.

Retrieve Selected Text（From WebView）

```
var selectedTextMarkerRangeValue: AnyObject?
let selectedTextMarkerRangeValueError = AXUIElementCopyAttributeValue(
  focusedElement as! AXUIElement,
  "AXSelectedTextMarkerRange" as CFString,
  &selectedTextMarkerRangeValue
)

...

var stringForTextMarkerRangeValue: AnyObject?
let stringForTextMarkerRangeValueError =
  AXUIElementCopyParameterizedAttributeValue(
    focusedElement as! AXUIElement,
    "AXStringForTextMarkerRange" as CFString,
    selectedTextMarkerRangeValue,
    &stringForTextMarkerRangeValue
  )

...
```

If a WebView is used, the text cannot be retrieved this way.
For example, web browsers or a mail application.
Instead, use `AXSelectedTextMarkerRange` to get the selection,

Retrieve Selected Text（From WebView）

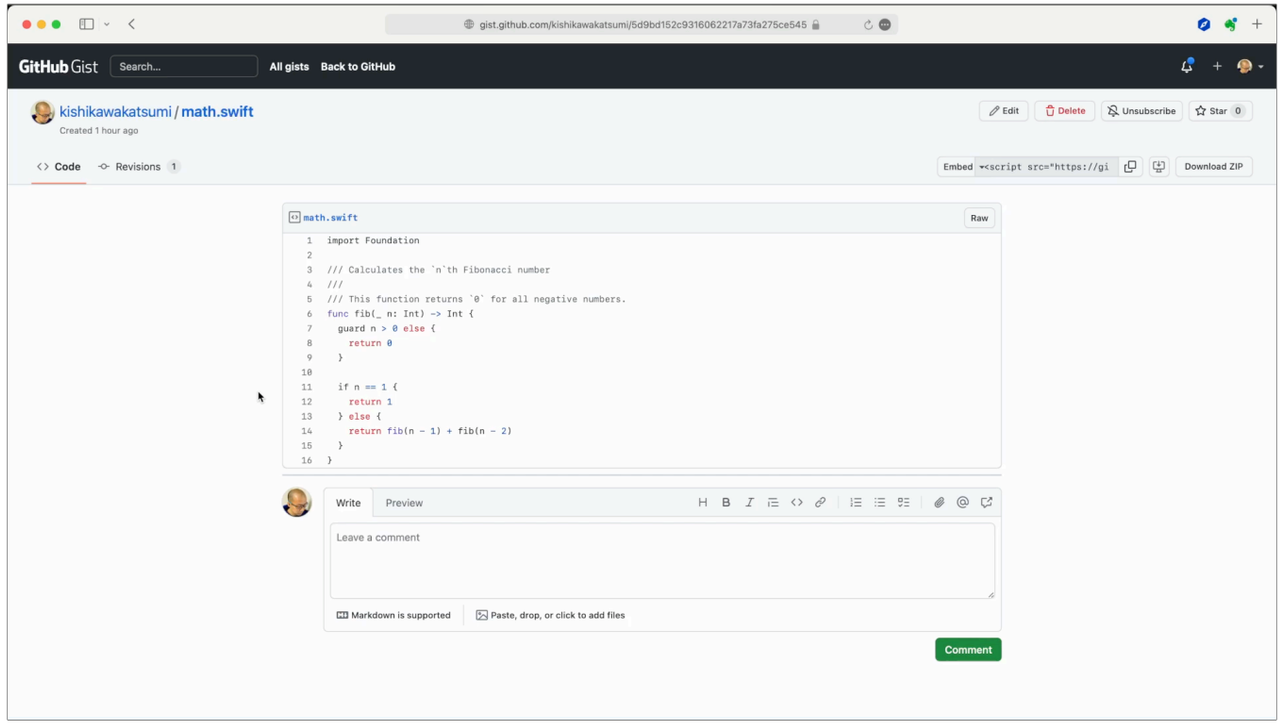
```
var selectedTextMarkerRangeValue: AnyObject?
let selectedTextMarkerRangeValueError = AXUIElementCopyAttributeValue(
  focusedElement as! AXUIElement,
  "AXSelectedTextMarkerRange" as CFString,
  &selectedTextMarkerRangeValue
)

...

var stringForTextMarkerRangeValue: AnyObject?
let stringForTextMarkerRangeValueError =
  AXUIElementCopyParameterizedAttributeValue(
    focusedElement as! AXUIElement,
    "AXStringForTextMarkerRange" as CFString,
    selectedTextMarkerRangeValue,
    &stringForTextMarkerRangeValue
  )

...
```

and `AXStringForTextMarkerRange` to get the text of the selection.



Now you can get the text from any application that uses WebView.
I mean, the procedure to retrieve values using the Accessibility API MAY fail depending on the target UI element. Therefore, it is recommended to try multiple methods and fallback on failure.

AXUIElementCopyParameterizedAttributeValue

```
var selectedTextMarkerRangeValue: AnyObject?
let selectedTextMarkerRangeValueError = AXUIElementCopyAttributeValue(
  focusedElement as! AXUIElement,
  "AXSelectedTextMarkerRange" as CFString,
  &selectedTextMarkerRangeValue
)

...

var stringForTextMarkerRangeValue: AnyObject?
let stringForTextMarkerRangeValueError =
  AXUIElementCopyParameterizedAttributeValue(
    focusedElement as! AXUIElement,
    "AXStringForTextMarkerRange" as CFString,
    selectedTextMarkerRangeValue,
    &stringForTextMarkerRangeValue
  )

...
```

The code example from earlier, the function below appears the first time.

This is `AXUIElementCopy`**`Parameterized`**`AttributeValue`.

It is similar to `AXUIElementCopyAttributeValue`,

AXUIElementCopyParameterizedAttributeValue

```
var selectedTextMarkerRangeValue: AnyObject?
let selectedTextMarkerRangeValueError = AXUIElementCopyAttributeValue(
  focusedElement as! AXUIElement,
  "AXSelectedTextMarkerRange" as CFString,
  &selectedTextMarkerRangeValue
)

...

var stringForTextMarkerRangeValue: AnyObject?
let stringForTextMarkerRangeValueError =
  AXUIElementCopyParameterizedAttributeValue(
    focusedElement as! AXUIElement,
    "AXStringForTextMarkerRange" as CFString,
    selectedTextMarkerRangeValue,
    &stringForTextMarkerRangeValue
  )

...
```

but takes an additional argument. You can get a more detailed value with the name of the key and an additional argument.
The argument passed here is the selection range,

AXUIElementCopyParameterizedAttributeValue

```
var selectedTextMarkerRangeValue: AnyObject?
let selectedTextMarkerRangeValueError = AXUIElementCopyAttributeValue(
  focusedElement as! AXUIElement,
  "AXSelectedTextMarkerRange" as CFString,
  &selectedTextMarkerRangeValue
)

...

var stringForTextMarkerRangeValue: AnyObject?
let stringForTextMarkerRangeValueError =
  AXUIElementCopyParameterizedAttributeValue(
    focusedElement as! AXUIElement,
    "AXStringForTextMarkerRange" as CFString,
    selectedTextMarkerRangeValue,
    &stringForTextMarkerRangeValue
  )

...
```

retrieved from the function above.
That is, it retrieves the text of the currently selected range.

How to Know Undocumented Keys?

```
var selectedTextMarkerRangeValue: AnyObject?
let selectedTextMarkerRangeValueError = AXUIElementCopyAttributeValue(
  focusedElement as! AXUIElement,
  "AXSelectedTextMarkerRange" as CFString,
  &selectedTextMarkerRangeValue
)

...

var stringForTextMarkerRangeValue: AnyObject?
let stringForTextMarkerRangeValueError =
  AXUIElementCopyParameterizedAttributeValue(
    focusedElement as! AXUIElement,
    "AXStringForTextMarkerRange" as CFString,
    selectedTextMarkerRangeValue,
    &stringForTextMarkerRangeValue
  )

...
```

These keys are not defined as constants.

How to know these undocumented keys?

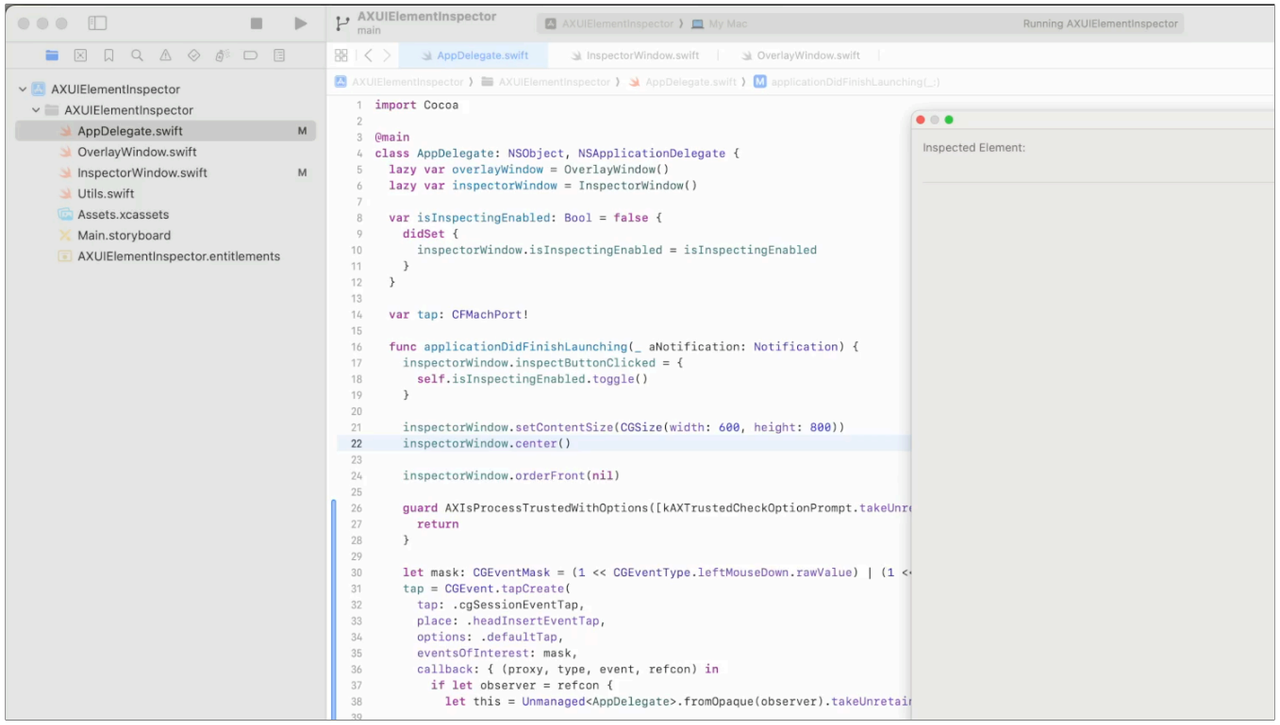
There is an API to search the names of such keys that are not defined as constants.

How to Know Undocumented Keys?

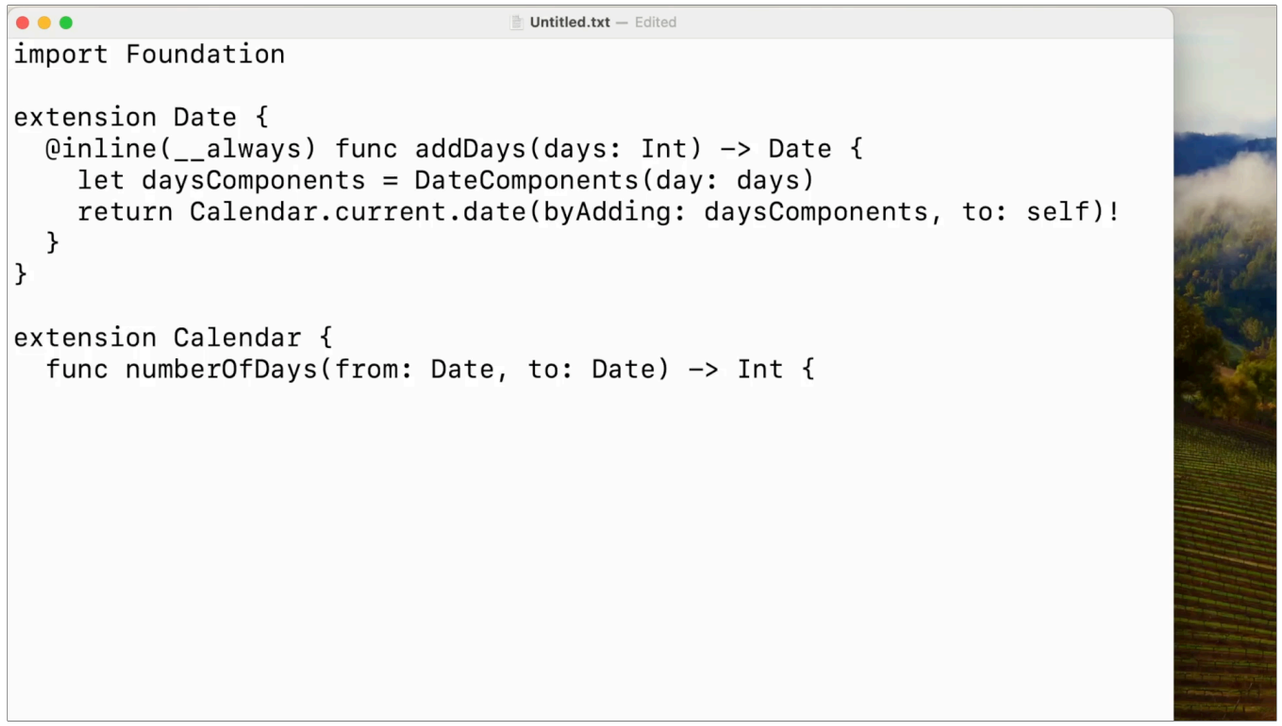
AXUIElementCopyAttributeNames(_:_:)

```
func AXUIElementCopyAttributeNames(
    _ element: AXUIElement,
    _ names: UnsafeMutablePointer<CFArray?>
) -> AXError
```

The `AXUIElementCopyAttribute`**`Names`** function returns a list of names of values that can be retrieved from an `AXUIElement`.



This is one of the sample applications in this session, which inspects the UI elements at the mouse cursor position.
It shows the value of the UI element at the cursor position and its tree structure to the window.
You can learn how to use the accessibility API with this application's source code.



The next sample application is, seamless integration between text editors and AI assistants.

Monitors text input to trigger functions. When a specific text, **`/assist`**, is typed, it triggers the function.

Then, the results are directly reflected to the editor.

Use a notification mechanism to achieve this functionality.

Notifications

```
var observer: AXObserver?
let observerCreateError = AXObserverCreate(
  app.processIdentifier,
  { (observer, element, notification, userData) in
    ...
  },
  &observer
)

AXObserverAddNotification(
  observer,
  appElement,
  kAXSelectedTextChangedNotification as CFString,
  nil
)
CFRunLoopAddSource(
  CFRunLoopGetCurrent(),
  AXObserverGetRunLoopSource(observer),
  .commonModes
)
```

Here is an example of code for receiving notifications.
First,

Notifications

```
var observer: AXObserver?
let observerCreateError = AXObserverCreate(
  app.processIdentifier,
  { (observer, element, notification, userData) in
    ...
  },
  &observer
)
```

```
AXObserverAddNotification(
  observer,
  appElement,
  kAXSelectedTextChangedNotification as CFString,
  nil
)
CFRunLoopAddSource(
  CFRunLoopGetCurrent(),
  AXObserverGetRunLoopSource(observer),
  .commonModes
)
```

create an observer that receives notifications.

Notifications

```
let app =
  NSRunningApplication
  .runningApplications(withBundleIdentifier: "com.apple.TextEdit")
  .first
```

```
          app.processIdentifier,
          { (observer, element, notification, userData) in
            ...
          },
          &observer
        )
```

```
        AXObserverAddNotification(
          observer,
          appElement,
          kAXSelectedTextChangedNotification as CFString,
          nil
        )
        CFRunLoopAddSource(
          CFRunLoopGetCurrent(),
          AXObserverGetRunLoopSource(observer),
          .commonModes
        )
```

Observers are created for each specific application.

Cannot receive system-wide notifications.

Notifications

```
var observer: AXObserver?
let observerCreateError = AXObserverCreate(
  app.processIdentifier,
  { (observer, element, notification, userData) in
    ...
  },
  &observer
)

AXObserverAddNotification(
  observer,
  appElement,
  kAXSelectedTextChangedNotification as CFString,
  nil
)
CFRunLoopAddSource(
  CFRunLoopGetCurrent(),
  AXObserverGetRunLoopSource(observer),
  .commonModes
)
```

Add the notification you want to receive to the observer.
In this example, observed kAXSelectedTextChangedNotification to monitor text input and changing cursor position.

Notifications

```
var observer: AXObserver?
let observerCreateError = AXObserverCreate(
  app.processIdentifier,
  { (observer, element, notification, userData) in
    ...
  },
  &observer
)

AXObserverAddNot
  observer,
  appElement,
  kAXSelectedTextChangedNotification as CFString,
  nil
)
CFRunLoopAddSource(
  CFRunLoopGetCurrent(),
  AXObserverGetRunLoopSource(observer),
  .commonModes
)
```

```
kAXSelectedTextChangedNotification
kAXValueChangedNotification
...
```

Notifications with the perfect content are almost never provided.
Therefore, notifications are used only for triggers, and the content is retrieved using the Accessibility API. That is how you can create a fine-grained UI.

Modify Text

```
var isAttributeSettableValue = DarwinBoolean(false)

let isAttributeSettableValueError = AXUIElementIsAttributeSettable(
  element,
  kAXValueAttribute as CFString,
  &isAttributeSettableValue
)

guard isAttributeSettableValueError == .success else {
  return
}

if isAttributeSettableValue.boolValue {
  AXUIElementSetAttributeValue(
    element,
    kAXValueAttribute as CFString,
    "\(precedingTextValue)\n\(codeBlock)" as CFString
  )
  return
}
```

To modify the text, check if the `AXUIElement` is editable.

Modify Text

```
var isAttributeSettableValue = DarwinBoolean(false)

let isAttributeSettableValueError = AXUIElementIsAttributeSettable(
  element,
  kAXValueAttribute as CFString,
  &isAttributeSettableValue
)

guard isAttributeSettableValueError == .success else {
  return
}

if isAttributeSettableValue.boolValue {
  AXUIElementSetAttributeValue(
    element,
    kAXValueAttribute as CFString,
    "\(precedingTextValue)\n\(codeBlock)" as CFString
  )
  return
}
```

To modify the text, check if the `AXUIElement` is editable.

The function `AXUIElementIsAttributeSettable` returns whether or not the `AXUIElement` can be modified.

Modify Text

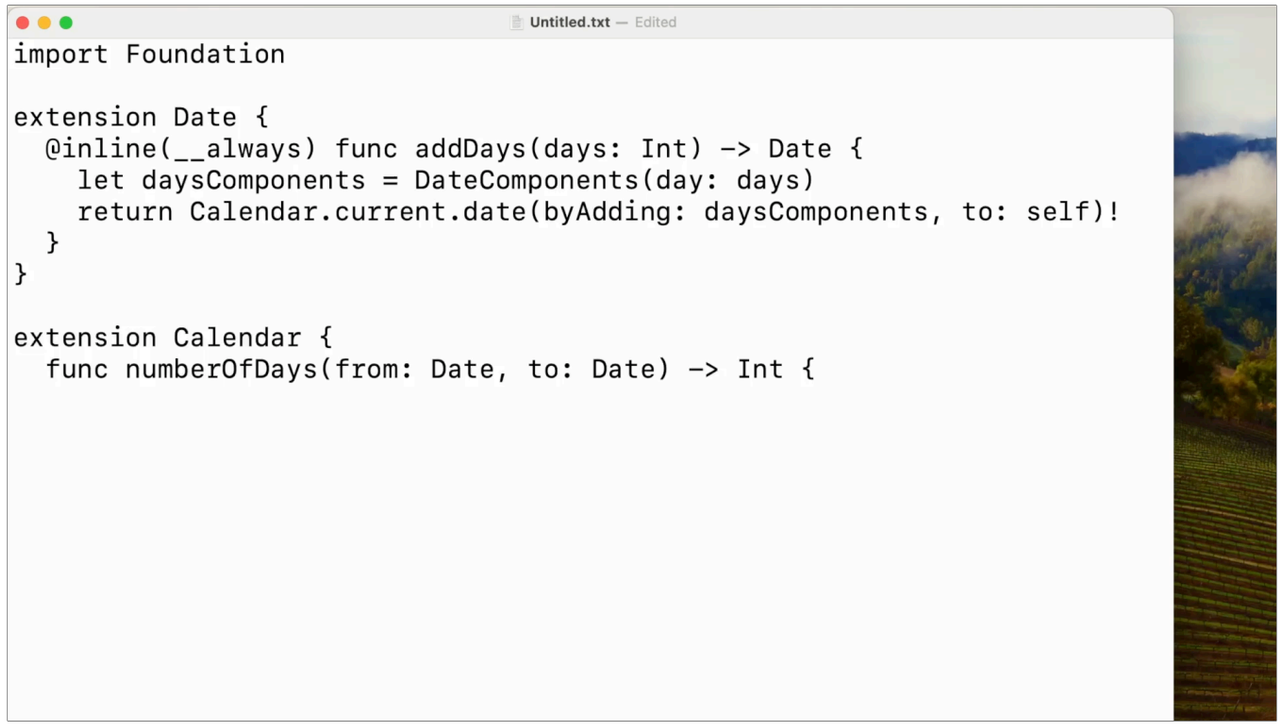
```
var isAttributeSettableValue = DarwinBoolean(false)

let isAttributeSettableValueError = AXUIElementIsAttributeSettable(
  element,
  kAXValueAttribute as CFString,
  &isAttributeSettableValue
)

guard isAttributeSettableValueError == .success else {
  return
}

if isAttributeSettableValue.boolValue {
  AXUIElementSetAttributeValue(
    element,
    kAXValueAttribute as CFString,
    "\(precedingTextValue)\n\(codeBlock)" as CFString
  )
  return
}
```

If it is editable, the content can be modified using the `AXUIElementSetAttributeValue` function.



That's it, we can now use the inline AI assistant like VS Code in TextEdiror.app.

The last example is an application that translates lines from comic books.
The Accessibility API does not provide a way to directly retrieve non-text content, such as images.
In this case, it uses the screen capture API to retrieve the content as an image. The Accessibility API is used to retrieve the coordinates of UI elements.

SCScreenshotManager.captureImage(...)

```
func captureScreen(rect: CGRect) async throws -> CGImage? {
  let content = try await SCShareableContent.excludingDesktopWindows(false, onScreenWindowsOnly: false)
  guard
    let mainDisplayID =
      NSScreen.main?.deviceDescription[NSDeviceDescriptionKey("NSScreenNumber")] as? CGDirectDisplayID,
    let display = content.displays.first(where: { $0.displayID == mainDisplayID })
  else { return nil }

  let streamConfig = SCStreamConfiguration()
  streamConfig.captureResolution = .automatic
  streamConfig.sourceRect = rect
  streamConfig.preservesAspectRatio = true
  streamConfig.scalesToFit = true
  streamConfig.capturesAudio = false
  streamConfig.excludesCurrentProcessAudio = true

  let filter = SCContentFilter(display: display, excludingWindows: [])

  let capturedImage = try await SCScreenshotManager.captureImage(
    contentFilter: filter, configuration: streamConfig
  )
  return capturedImage
}
```

The API for capturing screenshots is provided by ScreenCaptureKit, and screenshots can be captured by calling the `ScreenShotManager`'s `captureImage` class method.

Retrieve an Image

CGWindowListCreateImage(_:_:_:_:)

```
func captureScreen(rect: CGRect) -> CGImage? {
  let screenShot = CGWindowListCreateImage(
    rect,
    [.optionOnScreenOnly, .excludeDesktopElements],
    kCGNullWindowID,
    [.boundsIgnoreFraming, .bestResolution]
  )

  return screenShot
}
```

Another method used until just recently is the `CGWindowListCreateImage` function.

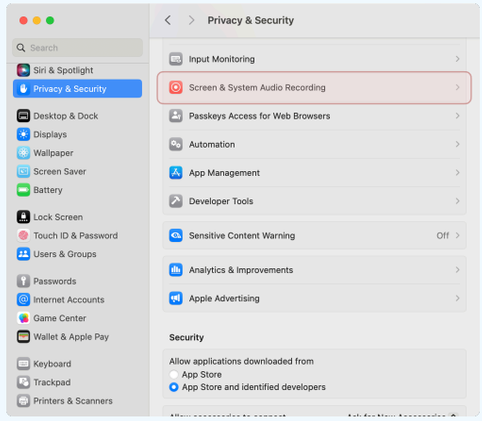
It has been deprecated in favor of ScreenCaptureKit.

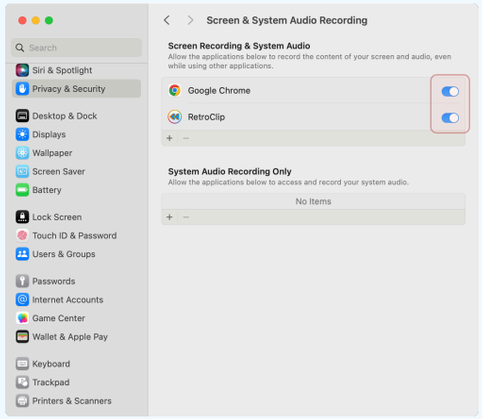
The sample code also shows how to use this function.

See the code for details.

Grant Screen Recording Permission

Privacy Settings



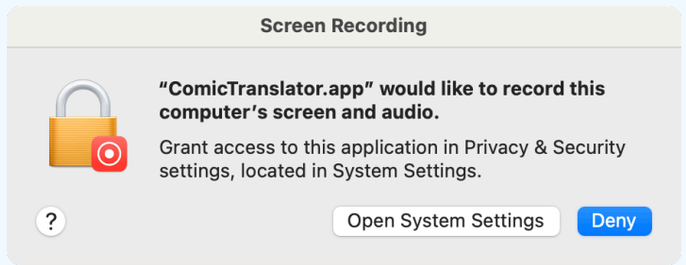


Screen recording permission is required to use the screenshot API.

Grant Screen Recording Permission

CGPreflightScreenCaptureAccess()

- CGPreflightScreenCaptureAccess()
  - Indicates whether screen recording is allowed
- CGRequestScreenCaptureAccess()
  - Present the system prompt



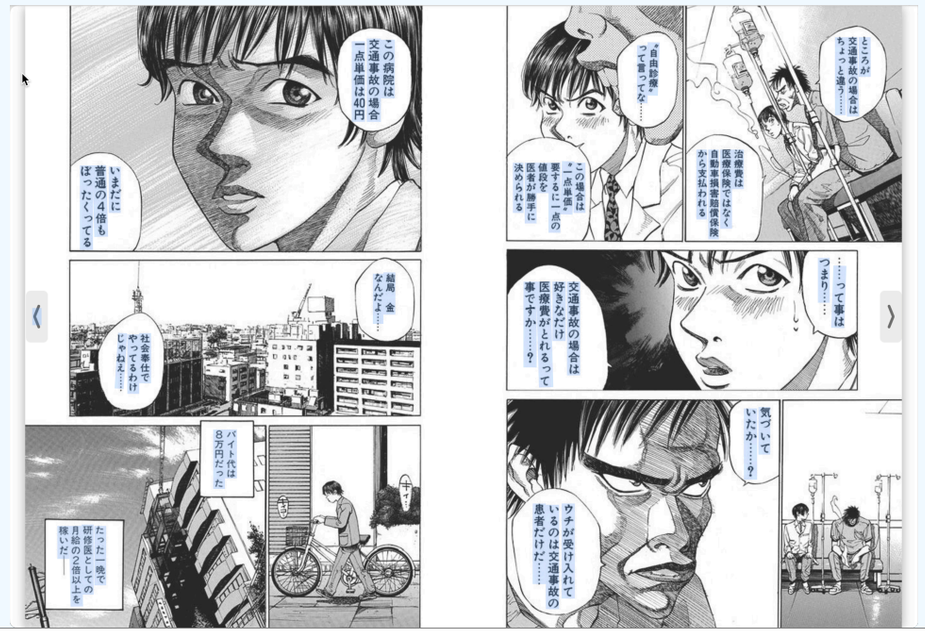
The `CGPreflightScreenCaptureAccess` function checks for the screen recording permission.

`CGRequestScreenCaptureAccess` displays a system dialog to ask for the permission.

Extract the Text from an Image

Use OCR to retrieve text from images.
To convert lines from Japanese comic books into text, It is tough to beat vertical writing.

VisionKit



Thanks to the VisionKit framework, vertical text can be recognized with amazing accuracy.

Extract the Text from an Image

ImageAnalyzer

```
let analyzer = ImageAnalyzer()
let analysis = try await analyzer.analyze(
  image,
  orientation: .up,
  configuration: configuration
)

let transcript = analysis.transcript
```

The image is analyzed in ImageAnalyzer of VisionKit.

Extract the Text from an Image

ImageAnalyzer

```
let analyzer = ImageAnalyzer()
let analysis = try await analyzer.analyze(
  image,
  orientation: .up,
  configuration: configuration
)

let transcript = analysis.transcript
```

The recognized text can be retrieved from the transcript property.

Then, the AI translates the retrieved text.
It is difficult to get the order of the lines correct automatically.
It may be possible to solve this problem by creating a UI that allows selection per panel, for example.
Accessibility APIs are very powerful, but to achieve the desired behavior, creativity is important, such as combining different frameworks or innovating the UI.

Wrap Up

- Window Chat Assistant
  - Basic Usage of Accessibility API
- Inline Chat Assistant
  - Notifications
  - Modify Content
- Comic Translator
  - Screen Capture API and OCR
- AXUIElement Inspector
  - Inspect Attribute Name of AXUIElement

So far, I have explained the basic usage of the Accessibility API, notifications, screenshots, and how to inspect the attributes of an `AXUIElement` using sample applications.

Sample Code

➡️ https://github.com/kishikawakatsumi/tryswift2024

Sample code can be accessed from my GitHub repository.
I hope you found that the accessibility API is very powerful and that you can create useful software with your own creativity.
I hope you will try it out for yourself.
Thank you for listening.

References

- Sample Code
  https://github.com/kishikawakatsumi/tryswift2024
- Accessing text value from any System wide Application via Accessibility API
  https://macdevelopers.wordpress.com/2014/01/31/accessing-text-value-from-any-system-wide-application-via-accessibility-api/
- How to get selected text and its coordinates from any system wide application using Accessibility API?
  https://macdevelopers.wordpress.com/2014/02/05/how-to-get-selected-text-and-its-coordinates-from-any-system-wide-application-using-accessibility-api/
- Why aren't the most useful Mac apps on the App Store?
  https://alinpanaitiu.com/blog/apps-outside-app-store/
- All about macOS event observation
  https://docs.google.com/presentation/d/1nEaiPUduh1vjks0rDVRTcJaEULbSWWh1tVdG2HF_XSU/htmlpresent