class ShareViewController: UIViewController {

var sharedText: String?

var textView: UITextView!

override func viewDidLoad() {

super.viewDidLoad()

textView = UITextView(frame: self.view.bounds)

self.view.addSubview(textView)

if let item = self.extensionContext?.inputItems.first as? NSExtensionItem,

let itemProvider = item.itemProviders.first,

itemProvider.canLoadObject(ofClass: NSString.self) {

itemProvider.loadObject(ofClass: NSString.self) {

[weak self] (text, error) in

if let text = text as? String {

DispatchQueue.main.async {

self.sharedText = text

self.textView.text = self.sharedText

}

}

}

}

}

}

func isContentValid() -> Bool {

return sharedText != nil && !sharedText!.isEmpty

}

func didSelectPost() {

self.extensionContext!.completeRequest(returningItems: [],

completionHandler: nil)

if let text = sharedText {

UserDefaults.standard.set(text, forKey: "sharedText")

UserDefaults.standard.synchronize()

}

}

func configurationItems() -> [Any]! {

// This method allows customization of the share sheet options.

return []

}

}

import MobileCoreServices

class ActionViewController: UIViewController {

@IBOutlet weak var textView: UITextView!

var originalText: String?

override func viewDidLoad() {

super.viewDidLoad()

// Get the item we are working with from the extension context.

if let inputItem = extensionContext?.inputItems.first as? NSExtensionItem,

let itemProvider = inputItem.itemProviders.first,

itemProvider.hasItemConformingToTypeIdentifier(UTType.plainText.identifier)

{

if itemProvider.canLoadObject(ofClass: NSString.self) {

itemProvider.loadObject(ofClass:NSString.self) { (text, error) in

if let text = text as? String {

DispatchQueue.main.async {

self.originalText = text

self.textView.text = text

}

}

}

}

}

}

@IBAction func done() {

// Return edited content to the host

let outputItem = NSExtensionItem()

let text = self.textView.text ?? ""

let outputItemProvider = NSItemProvider(item: text as NSSecureCoding,

typeIdentifier: UTType.plainText.identifier)

outputItem.attachments = [outputItemProvider]

self.extensionContext!.completeRequest(returningItems: [outputItem],

completionHandler: nil)

}

}

class FileProviderExtension: NSFileProviderExtension {

override func item(for identifier: NSFileProviderItemIdentifier) throws -> NSFileProviderItem {

return MyFileProviderItem(identifier: identifier)

}

override func providePlaceholder(at url: URL, completionHandler: @escaping (Error?) -> Void) {

let placeholderURL = NSFileProviderManager.placeholderURL(for: url)

let metadata: [FileAttributeKey: Any] = [.filename: url.lastPathComponent]

try? NSFileProviderManager.default.createPlaceholder(at: placeholderURL, withMetadata: metadata)

completionHandler(nil)

}

override func startProvidingItem(at url: URL, completionHandler: @escaping (Error?) -> Void) {

let sourceURL =

FileManager.default.temporaryDirectory.appendingPathComponent(url.lastPathComponent).fileSystemRepresentation

try? FileManager.default.copyItem(at: sourceURL, to: url)

completionHandler(nil)

}

override func stopProvidingItem(at url: URL) {

try? FileManager.default.removeItem(at: url)

}

}

class MyFileProviderItem: NSObject, NSFileProviderItem {

var itemIdentifier: NSFileProviderItemIdentifier

var parentItemIdentifier: NSFileProviderItemIdentifier = .rootContainer

var filename: String = "example.txt"

var typeIdentifier: String = UTType.plainText.identifier

init(identifier: NSFileProviderItemIdentifier) {

self.itemIdentifier = identifier

}

}

func startProvidingItem(at url: URL, completionHandler: @escaping (Error?) -> Void) {

let fileName = url.lastPathComponent

let sourceURL = FileManager.default.temporaryDirectory.appendingPathComponent(url.lastPathComponent).fileSystemRepresentation

let fileHandle = try! FileHandle(forReadingFrom: sourceURL)

let bufferSize = 1024 \* 1024 // 1MB buffer

while let data = try? fileHandle.read(upToCount: bufferSize) {

if data.isEmpty { break }

// Process the data chunk, also check for errors

}

completionHandler(nil)

}

import Photos

import PhotosUI

class PhotoEditingViewController: UIViewController, PHContentEditingController {

@IBOutlet weak var imageView: UIImageView!

var inputImage: UIImage?

func canHandle(\_ adjustmentData: PHAdjustmentData) -> Bool {

return adjustmentData.formatIdentifier == "com.example.MyPhotoEditingExtension"

}

func startContentEditing(with contentEditingInput: PHContentEditingInput, placeholderImage: UIImage) {

inputImage = contentEditingInput.displaySizeImage

imageView.image = inputImage

}

func finishContentEditing(completionHandler: @escaping (PHContentEditingOutput?) -> Void) {

let adjustmentData = PHAdjustmentData(formatIdentifier: "com.example.MyPhotoEditingExtension", formatVersion: "1.0", data: Data())

let output = PHContentEditingOutput(contentEditingInput: contentEditingInput)

if let jpegData = imageView.image?.jpegData(compressionQuality: 1.0) {

try? jpegData.write(to: output.renderedContentURL)

}

output.adjustmentData = adjustmentData

completionHandler(output)

}

func cancelContentEditing() {

// Handle cancellation

}

}

class KeyboardViewController: UIInputViewController {

private var nextKeyboardButton: UIButton!

override func viewDidLoad() {

super.viewDidLoad()

setupNextKeyboardButton()

}

private func setupNextKeyboardButton() {

nextKeyboardButton = UIButton(type: .system)

nextKeyboardButton.setTitle(NSLocalizedString("Next Keyboard Button", comment: "Next Keyboard Button Title"), for: [])

nextKeyboardButton.sizeToFit()

nextKeyboardButton.translatesAutoresizingMaskIntoConstraints = false

view.addSubview(nextKeyboardButton)

NSLayoutConstraint.activate([

nextKeyboardButton.leftAnchor.constraint(equalTo: view.leftAnchor),

nextKeyboardButton.bottomAnchor.constraint(equalTo: view.bottomAnchor)

])

nextKeyboardButton.addTarget(self,

action:#selector(advanceToNextInputMode) for: .touchUpInside)

}

@IBAction func keyTapped(\_ sender: UIButton) {

(textDocumentProxy as UIKeyInput)?.insertText(sender.title(for: .normal)

?? "")

}

}

import Intents

class IntentHandler: INExtension, MyCustomIntentHandling {

func handle(intent: MyCustomIntent, completion: @escaping (MyCustomIntentResponse) -> Void) {

let response = MyCustomIntentResponse(code: .success, userActivity: nil)

completion(response)

}

override func handler(for intent: INIntent) -> Any {

return self

}

}

import Intents

import IntentsUI

func donateShortcut() {

let intent = MyCustomIntent()

intent.suggestedInvocationPhrase = "Do something"

let userActivity = NSUserActivity(activityType: "com.example.myShortcut")

userActivity.title = "Do something"

userActivity.suggestedInvocationPhrase = "Do something"

intent.userActivity = userActivity

let interaction = INInteraction(intent: intent, response: nil)

interaction.donate { error in

if let error = error {

print("Donation failed: \(error.localizedDescription)")

} else {

print("Successfully donated interaction")

}

}

}

@main

struct MyAppClip: App {

var body: some Scene {

WindowGroup {

ContentView()

}

}

}

struct ContentView: View {

var body: some View {

VStack {

Text("Welcome to My App Clip")

.font(.largeTitle)

.padding()

Button(action: {

// Action for the button

}) {

Text("Get Started")

.font(.title2)

.padding()

.background(Color.blue)

.foregroundColor(.white)

.cornerRadius(10)

}

}

}

}

@main

struct MyAppClip: App { // For SwiftUI, UIKit projects should use AppDelegate

var body: some Scene {

WindowGroup {

ContentView()

.onOpenURL { url in

handleURL(url)

}

}

}

func handleURL(\_ url: URL) {

// Handle the URL and provide context to the App Clip,

// parse specific parameters using URLComponents

print("App Clip invoked with URL: \(url)")

}

}

import UserNotifications

import UserNotificationsUI

class NotificationViewController: UIViewController, UNNotificationContentExtension {

@IBOutlet weak var titleLabel: UILabel!

@IBOutlet weak var bodyLabel: UILabel!

func didReceive(\_ notification: UNNotification) {

titleLabel.text = notification.request.content.title

bodyLabel.text = notification.request.content.body

}

}

func scheduleNotification() {

let content = UNMutableNotificationContent()

content.title = "Hello"

content.body = "This is a custom notification."

content.categoryIdentifier = "myNotificationCategory"

let trigger = UNTimeIntervalNotificationTrigger(timeInterval: 5, repeats: false)

let request = UNNotificationRequest(identifier: "testNotification", content: content, trigger: trigger)

UNUserNotificationCenter.current().add(request, withCompletionHandler: nil)

}

// Call this function to schedule a notification

scheduleNotification()

@UIApplicationMain

class AppDelegate: UIResponder, UIApplicationDelegate {

func application(\_ application: UIApplication,

didFinishLaunchingWithOptions launchOptions:

[UIApplication.LaunchOptionsKey: Any]?) -> Bool {

let center = UNUserNotificationCenter.current()

center.requestAuthorization(options: [.alert, .sound, .badge]) { granted, error in }

let category = UNNotificationCategory(

identifier: "myNotificationCategory",

actions: [],

intentIdentifiers: [],

options: [])

center.setNotificationCategories([category])

return true

}

}