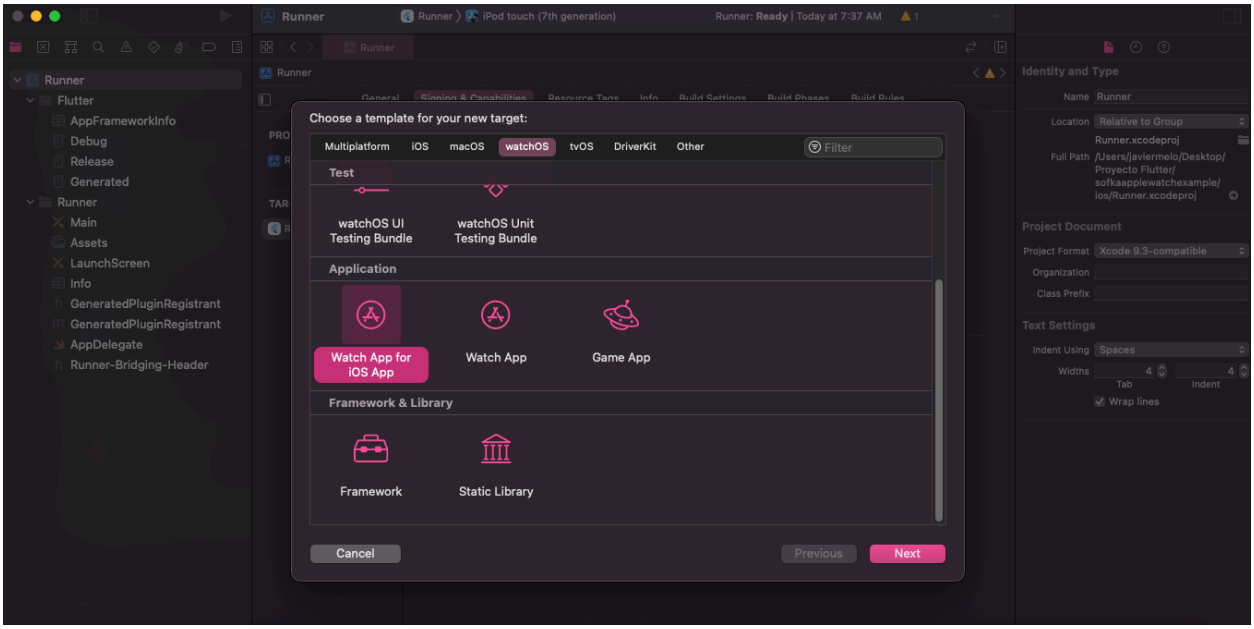
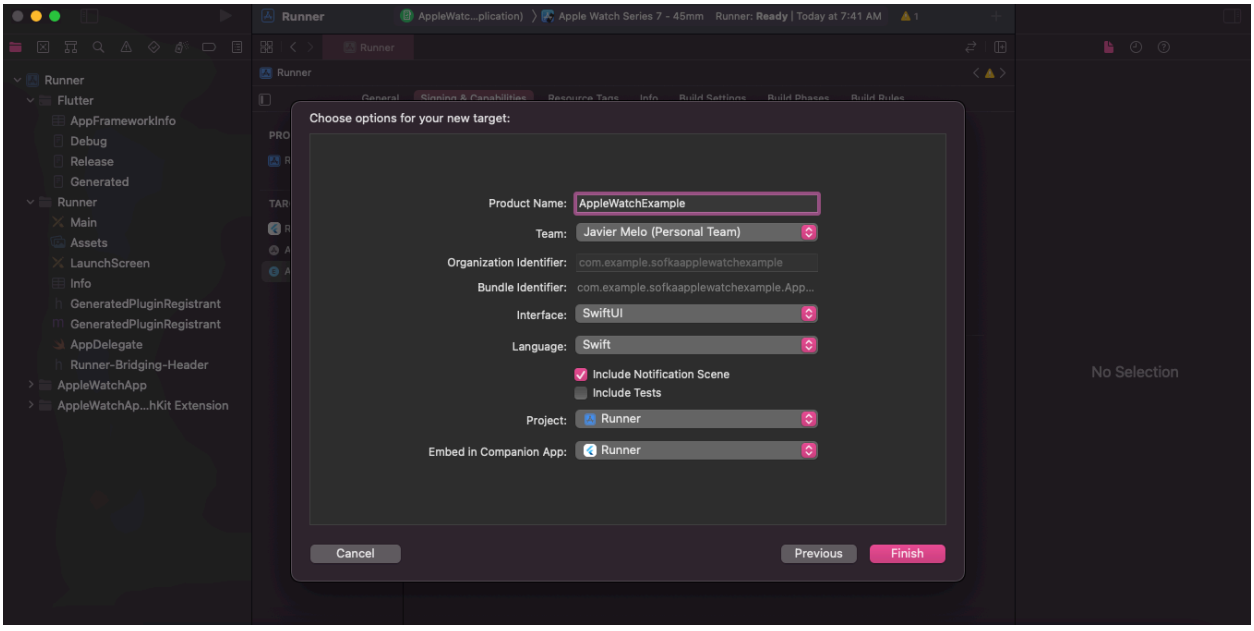
Implementación de extensión de Apple Watch(SwiftUI) en proyecto Flutter

Añadir Apple Watch Extension en el proyecto Flutter

1.Una vez creado el proyecto flutter de manera común se debe abrir el proyecto Xcode para agregar la extensión del Apple Watch. Para esto se debe seleccionar File>New>Target.



2.Se debe seleccionar que el tipo de interfaz sea SwiftUI y el lenguaje igual, se debe deseleccionar el “Include Tests” y el “Include Notification Scene”. Una vez hecho esto solo se debe dar click en finalizar.



Añadir el canal que permite recibir información y enviarla al Apple Watch

1.Se debe ir al archivo AppDelegate y crear el método que permita inicializar el canal y definir cada uno de los métodos que se esperar de dicho canal. Lo llamaremos con el identificador “com.sofkaexample.watch” y por razones del ejercicio el método principal que llamaremos sera flutterToWatch. Este case es el encargado de recibir los datos de Flutter y posteriormente enviarla al Apple Watch, pero por ahora solo se esta comunicando con el proyecto iOS.

```
//Inicializar canal de flutter
  private func initFlutterChannel() {
        if let controller = window?.rootViewController as? FlutterViewController {
            let channel = FlutterMethodChannel(
                name: "com.sofkaexample.watch",
                binaryMessenger: controller.binaryMessenger)

            channel.setMethodCallHandler({ [weak self] (
                call: FlutterMethodCall,
                result: @escaping FlutterResult) -> Void in
                switch call.method {
                case "flutterToWatch":
                    guard let watchSession = self?.session, watchSession.isPaired, watchSession.isReachable, let methodData =
                        call.arguments as? [String: Any], let method = methodData["method"], let data = methodData["data"] as? Any
                        else {
                        result(false)
                        return
                    }

                    let watchData Show the Source Control navigator ": method, "data": data]

                    // Pass the receiving message to Apple Watch
                    watchSession.sendMessage(watchData, replyHandler: nil, errorHandler: nil)
                    result(true)
                default:
                    result(FlutterMethodNotImplemented)
                }
            })
        }
    }
}
```

Añadir el canal que permite enviar y recibir información del Apple Watch

1. Así como se creo el canal que recibe información desde el proyecto Swift y enviarla Apple Watch, Se debe crear el método desde flutter que permita crear la comunicación. El canal en Flutter debe llevar el mismo identificador previamente definido “com.sofkaexample.watch” y el mismo nombre del caso “flutterToWatch”. Por temas del ejercicio lo haremos en la misma clase(main.dart), pero es recomendable según la arquitectura de la aplicación separarlo en otra clase que sirva de Source de todos los métodos que se crearan para enviar o recibir información.

```
class _MyHomePageState extends State<MyHomePage> {
  int _counter = 0;
  final channel = const MethodChannel('com.sofkaexample.watch');
  void _incrementCounter() async {
    setState(() {
      _counter++;
    });
    await channel.invokeMethod(
        "flutterToWatch", {"method": "sendCounterToNative", "data": _counter});
  }
```

2. Asimismo cómo se crea otro método que es el encargado de recibir la información del Apple Watch y tenerla en Flutter. Con este ejemplo se tendría una comunicación bidireccional entre Flutter y el AppleWatch. Para este caso lo que se enviaría es el incremento de un contador y lo que se recibiría es el incremento del mismo contador pero hecho desde el Apple Watch.

```dart
Future<void> _initFlutterChannel() async {
  channel.setMethodCallHandler((call) async {
    // Receive data from Native
    switch (call.method) {
      case "sendCounterToFlutter":
        _counter = call.arguments["data"]["counter"];
        _incrementCounter();
        break;
      default:
        break;
    }
  });
}

@override
void initState() {
  super.initState();
  _initFlutterChannel();
}
```

Crear la interfaz Gráfica en el Apple Watch y el ViewModel

1. Para esta parte se requiere de dos partes la de crear el ViewModel y la de la propia interfaz del Apple Watch, para que la vista se comunique con el ViewModel define una instancia hacia el mismo de tipo @ObservableObject. En este código se gráfica el valor del contador recibido de Flutter pero así mismo se modifica desde el Apple Watch y al mismo tiempo se envía mediante un método en el ViewModel a Flutter.

```swift
import SwiftUI

struct ContentView: View {
    @ObservedObject var viewModel: WatchViewModel =
        WatchViewModel()
    var body: some View {
        VStack {
                Text("Contador: \(viewModel.counter)")
                    .padding()
                Button(action: {
                    viewModel.sendDataMessage(for:
                        .sendCounterToFlutter, data:
                        ["counter": viewModel.counter + 1])
                }) {
                    Text("Añadir")
                }
            }
    }
}

struct ContentView_Previews: PreviewProvider {
    static var previews: some View {
        ContentView()
    }
}
```

2. En el view model se definen todas las operaciones de recepción de la información y de envío de la información, se crea la WCSessionDelegate que es la encargada de esto.

```swift
import Foundation
import WatchConnectivity

class WatchViewModel: NSObject, ObservableObject {
    var session: WCSession
    @Published var counter = 0

    // Añadir más casos de recepción
    enum WatchReceiveMethod: String {
        case sendCounterToNative
    }

    // Añadir más casos de envío
    enum WatchSendMethod: String {
        case sendCounterToFlutter
    }

    init(session: WCSession = .default) {
        self.session = session
        super.init()
        self.session.delegate = self
        session.activate()
    }

    // Este método es el que se activa a través del Boton en el ContentView, recibe el valor aumentado del contador para posteriormente
        enviarlo a Flutter
    func sendDataMessage(for method: WatchSendMethod, data: [String: Any] = [:]) {
        sendMessage(for: method.rawValue, data: data)
    }

}
```

```swift
extension WatchViewModel: WCSessionDelegate {

    func session(_ session: WCSession, activationDidCompleteWith activationState: WCSessionActivationState, error: Error?) {

    }

    // Recibe el mensaje del AppDelegate.swift
    func session(_ session: WCSession, didReceiveMessage message: [String : Any]) {
        DispatchQueue.main.async {
            guard let method = message["method"] as? String, let enumMethod = WatchReceiveMethod(rawValue: method) else {
                return
            }

            switch enumMethod {
            case .sendCounterToNative:

                self.counter = (message["data"] as? Int) ?? 0
            }
        }
    }

    // Envía el mensaje desde el Apple Watch a Flutter
    func sendMessage(for method: String, data: [String: Any] = [:]) {
        guard session.isReachable else {
            return
        }
        let messageData: [String: Any] = ["method": method, "data": data]
        session.sendMessage(messageData, replyHandler: nil, errorHandler: nil)
    }

}
```

Añadir el WatchSession que permitiría hacer de puente entre Flutter, El proyecto iOS y el Proyecto del AppleWatch

1. En el AppDelegate.swift se debe incur el canal de flutter y verificar si la WCSession esta soportado.

```swift
@UIApplicationMain
@objc class AppDelegate: FlutterAppDelegate {
  override func application(
    _ application: UIApplication,
    didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?
  ) -> Bool {

      initFlutterChannel()
      if WCSession.isSupported() {
          session = WCSession.default;
          session?.delegate = self;
          session?.activate();

      }

    GeneratedPluginRegistrant.register(with: self)
    return super.application(application, didFinishLaunchingWithOptions: launchOptions)
  }
```

2. Se crea el AppDelegate encargado de la comunicación entre Flutter, El proyecto iOS y el Proyecto del Apple Watch.

```swift
extension AppDelegate: WCSessionDelegate {

    func session(_ session: WCSession, activationDidCompleteWith activationState: WCSessionActivationState, error: Error?) {

    }

    func sessionDidBecomeInactive(_ session: WCSession) {

    }

    func sessionDidDeactivate(_ session: WCSession) {

    }

    func session(_ session: WCSession, didReceiveMessage message: [String : Any]) {
        DispatchQueue.main.async {
            if let method = message["method"] as? String, let controller = self.window?.rootViewController as? FlutterViewController {
                let channel = FlutterMethodChannel(
                    name: "com.sofkaexample.watch",
                    binaryMessenger: controller.binaryMessenger)
                channel.invokeMethod(method, arguments: message)
            }
        }
    }
}
```

El resultado final sera una app Flutter comunicada con una App Extension para el Apple Watch

Link al Github del proyecto—>**https://github.com/javiermelosofka/FlutterToAppleSofkaDemo**

Referencias

- https://medium.com/kbtg-life/adding-apple-watch-to-flutter-app-via-flutter-method-channel-f1443532d94e
- https://docs.flutter.dev/development/platform-integration/ios/apple-watch