**Integration of epstudio\_sdk\_flutter\_module Plugin**

**Android Integration**

**1. Prepare the AAR Package**

* Obtain the built AAR package. For this project, the package is provided as repo.tar.gz. Extract it to obtain the full path of the folder, for example: /xxx/xxx/repo.

**2. Configure the Repository**

* In the settings.gradle file, add the repository:

String storageUrl = System.env.FLUTTER\_STORAGE\_BASE\_URL ?: "https://storage.googleapis.com"

repositories {

maven {

url '/xxx/xxx/repo'

}

maven {

url "$storageUrl/download.flutter.io"

}

}

**3. Add Dependencies**

* In the build.gradle file, add the necessary dependencies and configure the profile:

dependencies {

// Dependency for WebSocket

implementation 'org.java-websocket:Java-WebSocket:1.5.3'

// Project dependencies

debugImplementation 'com.neuramatrix.epstudio\_sdk\_flutter\_module:flutter\_debug:1.0'

profileImplementation 'com.neuramatrix.epstudio\_sdk\_flutter\_module:flutter\_profile:1.0'

releaseImplementation 'com.neuramatrix.epstudio\_sdk\_flutter\_module:flutter\_release:1.0'

}

android {

buildTypes {

profile {

initWith debug

}

}

}

**4. Authorization**

* In the AndroidManifest.xml, add the required permissions for Bluetooth and location services, and configure FlutterActivity:

<manifest>

<!-- Location permission -->

<uses-permission android:name="android.permission.ACCESS\_COARSE\_LOCATION"/>

<!-- Bluetooth permissions -->

<uses-permission android:name="android.permission.BLUETOOTH"/>

<uses-permission android:name="android.permission.BLUETOOTH\_ADMIN"/>

<uses-permission android:name="android.permission.BLUETOOTH\_SCAN"/>

<uses-permission android:name="android.permission.BLUETOOTH\_CONNECT"/>

<uses-permission android:name="android.permission.BLUETOOTH\_ADVERTISE"/>

<uses-permission android:name="android.permission.BLUETOOTH\_PRIVILEGED" tools:ignore="ProtectedPermissions"/>

<!-- Flutter integration -->

<activity

android:name="io.flutter.embedding.android.FlutterActivity"

android:configChanges="orientation|keyboardHidden|keyboard|screenSize|locale|layoutDirection|fontScale|screenLayout|density|uiMode"

android:hardwareAccelerated="true"

android:theme="@style/Theme.ExampleAndroid"

android:windowSoftInputMode="adjustResize"

/>

</manifest>

**5. Integration with Startup Method**

* **Project Startup Entry:** Start FlutterEngine. The socketPort is the port for the SocketServer started within the Flutter plugin, with a default value of 3000.

// Create FlutterEngine

val flutterEngine = FlutterEngine(this)

// Start executing Dart code to pre-warm the FlutterEngine.

flutterEngine.dartExecutor.executeDartEntrypoint(

DartExecutor.DartEntrypoint.createDefault(),

listOf("socketPort=3000")

)

// Cache the FlutterEngine to be used by FlutterActivity.

FlutterEngineCache.getInstance().put("epstudio\_sdk\_flutter", flutterEngine)

// Request Bluetooth-related permissions

requestMultiplePermissions.launch(arrayOf(

android.Manifest.permission.ACCESS\_FINE\_LOCATION,

android.Manifest.permission.BLUETOOTH\_SCAN,

android.Manifest.permission.BLUETOOTH\_CONNECT

))

**6. Socket Client**

typealias ReceiveListener = ((str: String) -> Unit)

// WebSocket client utility class

object WebSocketUtil {

var webSocketClient: WebSocketClient

var listenerMap = mutableMapOf<String, ReceiveListener>()

const val port = 3000

init {

// URL address

val serverURI = URI.create("ws://127.0.0.1:$port/socket.io/?transport=websocket")

webSocketClient = object : WebSocketClient(serverURI) {

override fun onOpen(handshakedata: ServerHandshake?) {

Log.i("onOpen", "WebSocket connected=============================")

onMsg("onOpen WebSocket connected=============================")

}

override fun onWebsocketPong(conn: WebSocket?, f: Framedata?) {

Log.i("onWebsocketPong", "Connection: $conn Received: $f")

conn?.send("30")

}

override fun onMessage(message: String?) {

message?.let {

if (it.startsWith("42")) {

val res = JSONArray(it.substring(2))

if (res[0].equals("data")) {

onData(res[1].toString())

} else {

onMsg(res[1].toString())

}

} else {

Log.i("onMessage", "Received server message: $message")

}

}

}

override fun onClose(code: Int, reason: String?, remote: Boolean) {

Log.i("onClose", "WebSocket closed")

onMsg("onClose WebSocket closed")

}

override fun onError(ex: Exception?) {

Log.i("onError", "Server status: ${ex?.toString()}")

onMsg("onError Server status: ${ex?.toString()}")

}

}

}

fun onMsg(msg: String?) {

listenerMap["msg"]?.invoke(msg ?: "null")

}

fun onData(data: String?) {

listenerMap["data"]?.invoke(data ?: "null")

}

}

// Receive msg type data

WebSocketUtil.listenerMap["msg"] = { str: String ->

println(str)

}

// Receive data type data

WebSocketUtil.listenerMap["data"] = { str: String ->

println(str)

}

var socketClient: WebSocketClient = WebSocketUtil.webSocketClient

// Establish connection

socketClient.connect()

// Start scanning

val params = JSONArray()

params.put("startScan")

val a = "42$params"

socketClient.send(a)

// Stop scanning

val params = JSONArray()

params.put("stopScan")

val a = "42$params"

socketClient.send(a)

// Connect to device

val deviceList = JSONArray()

deviceList.put(item.address)

val params = JSONArray()

params.put("deviceConnect")

params.put(deviceList)

val a = "42$params"

socketClient.send(a)

// Disconnect device

val deviceList = JSONArray()

deviceList.put(item.address)

val params = JSONArray()

params.put("deviceDisconnect")

params.put(deviceList)

socketClient.send("42$params")

// Start data collection

val deviceList = JSONArray()

deviceList.put(item.address)

val params = JSONArray()

params.put("deviceCollect")

params.put(deviceList)

val a = "42$params"

socketClient.send(a)

// Stop data collection

val deviceList = JSONArray()

deviceList.put(item.address)

val params = JSONArray()

params.put("deviceStopCollect")

params.put(deviceList)

socketClient.send("42$params")

// Set channel (supports multiple selections; if channels is an empty array, the first channel will be enabled by default)

channelCheckBox.setOnClickListener {

val data = JSONObject()

data.put("address", item.address)

val channels = JSONArray()

for (c in item.channels!!) {

if (c.id == channel.id) {

if (!c.isChecked) channels.put(c.id.toInt())

} else {

if (c.isChecked) channels.put(c.id.toInt())

}

}

data.put("channels", channels)

val params = JSONArray()

params.put("setChannelStatus")

params.put(data)

val a = "42$params"

socketClient.send(a)

}

// Set sampling rate (single selection)

samplingRateCheckBox.setOnClickListener {

val data = JSONObject()

data.put("address", item.address)

data.put("value", samplingRate)

val params = JSONArray()

params.put("setSamplingRate")

params.put(data)

val a = "42$params"

socketClient.send(a)

}

// Set gain (single selection)

gainCheckBox.setOnClickListener {

val data = JSONObject()

data.put("address", item.address)

data.put("value", gain)

val params = JSONArray()

params.put("setGain")

params.put(data)

val a = "42$params"

socketClient.send(a)

}

**RN SDK Integration with Flutter Module in Android**

1. **Prepare AAR Package**  
   Prepare the built AAR package, which is provided as repo.tar.gz in this project. Extract it to obtain the full path of the folder, for example, /xxx/xxx/repo.
2. **Configure Repository**  
   Add the repository in the settings.gradle file:

String storageUrl = System.env.FLUTTER\_STORAGE\_BASE\_URL ?: "https://storage.googleapis.com"

repositories {

maven {

url '/xxx/xxx/repo'

}

maven {

url "$storageUrl/download.flutter.io"

}

}

**3. Add Dependencies**  
Add the necessary dependencies and configure the profile in the build.gradle file:

dependencies {

// Project dependencies

debugImplementation 'com.neuramatrix.epstudio\_sdk\_flutter\_module:flutter\_debug:1.0'

profileImplementation 'com.neuramatrix.epstudio\_sdk\_flutter\_module:flutter\_profile:1.0'

releaseImplementation 'com.neuramatrix.epstudio\_sdk\_flutter\_module:flutter\_release:1.0'

}

android {

buildTypes {

profile {

initWith debug

}

}

}

**4. Authorization**  
Add permissions and configure FlutterActivity in the AndroidManifest.xml:

<manifest>

<!-- Location permission -->

<uses-permission android:name="android.permission.ACCESS\_COARSE\_LOCATION"/>

<!-- Bluetooth permissions -->

<uses-permission android:name="android.permission.BLUETOOTH"/>

<uses-permission android:name="android.permission.BLUETOOTH\_ADMIN"/>

<uses-permission android:name="android.permission.BLUETOOTH\_SCAN"/>

<uses-permission android:name="android.permission.BLUETOOTH\_CONNECT"/>

<uses-permission android:name="android.permission.BLUETOOTH\_ADVERTISE"/>

<uses-permission android:name="android.permission.BLUETOOTH\_PRIVILEGED" tools:ignore="ProtectedPermissions"/>

<!-- Flutter integration -->

<activity

android:name="io.flutter.embedding.android.FlutterActivity"

android:configChanges="orientation|keyboardHidden|keyboard|screenSize|locale|layoutDirection|fontScale|screenLayout|density|uiMode"

android:hardwareAccelerated="true"

android:theme="@style/Theme.ExampleAndroid"

android:windowSoftInputMode="adjustResize"

/>

</manifest>

**5. Integration with Startup Method**  
Start the FlutterEngine and configure the project startup entry:

// Create FlutterEngine

val flutterEngine = FlutterEngine(this)

// Start executing Dart code to pre-warm the FlutterEngine.

flutterEngine.dartExecutor.executeDartEntrypoint(

DartExecutor.DartEntrypoint.createDefault(),

listOf("socketPort=3000")

)

// Cache the FlutterEngine to be used by FlutterActivity.

FlutterEngineCache.getInstance().put("epstudio\_sdk\_flutter", flutterEngine)

// Note: Manually enable app location and nearby devices permissions.

**6.Manually Enable App Location and Nearby Devices Permissions**  
Manually configure permissions for the app as needed.

**iOS Integration with Flutter Module**

**Step 1: Prepare the Flutter Framework**

1. **Switch to the Flutter Module Directory**  
   Navigate to the root directory of the Flutter module in your command line and run the following command to build the iOS frameworks:

flutter build ios-framework --cocoapods --output=some/path/MyApp/Flutter/

This command generates three integration packages: Debug, Profile, and Release, which can be provided to the required parties.

2. **Download and Extract the Package**  
Once the package (flutter\_frameworks.zip) is downloaded and extracted, you can get the full path for integration. Add the Flutter import in the Podfile:

pod 'Flutter', :podspec => 'some/path/MyApp/Flutter/[build mode]/Flutter.podspec'

1. **Note:** You must hardcode the [build mode] value. For example, use Debug during development if you need to use flutter attach, and use Release when you’re ready to ship the app.

**Step 2: Integrate Other Libraries**

1. **Add Frameworks to Xcode**  
   Drag the frameworks from some/path/MyApp/Flutter/Release/ into your target project in Xcode under **General > Frameworks, Libraries, and Embedded Content**. Select "Embed & Sign" from the Embed dropdown list.
2. **Update Build Settings**  
   In Xcode, navigate to **Build Settings** and add $(PROJECT\_DIR)/Flutter/ to **Framework Search Paths**.

**Important:** Always use the Flutter.framework and App.framework from the same directory. Mixing frameworks from different directories (e.g., Profile/Flutter.framework and Debug/App.framework) can cause the app to fail to run.

**Step 3: Add Required iOS Permissions**

1. **Update Info.plist**  
   Add the following keys to your Info.plist file to request necessary permissions for location and Bluetooth usage:

<key>NSBluetoothPeripheralUsageDescription</key>

<string>是否允许app使用蓝牙以便发送和接收数据？</string>

<key>NSBluetoothAlwaysUsageDescription</key>

<string>是否允许app使用蓝牙以便发送和接收数据？</string>

<key>NSLocationWhenInUseUsageDescription</key>

<string>是否允许app使用位置信息？</string>

<key>NSLocationAlwaysUsageDescription</key>

<string>是否允许app使用位置信息？</string>

<key>NSLocationAlwaysAndWhenInUseUsageDescription</key>

<string>是否允许app使用位置信息？</string>

<key>Privacy - NSLocationWhenInUseUsageDescription</key>

<string>是否允许app使用位置信息？</string>

<key>NSLocationUsageDescription</key>

<string>是否允许app使用位置信息？</string>

**Step 4: Register FlutterEngine in AppDelegate**

1. **Add FlutterEngine Registration**  
   In your AppDelegate.swift, register the FlutterEngine as follows:

import UIKit

import Flutter

import FlutterPluginRegistrant

@main

class AppDelegate: FlutterAppDelegate {

lazy var flutterEngine = FlutterEngine(name: "epstudio\_sdk\_flutter")

override func application(\_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

flutterEngine.run()

GeneratedPluginRegistrant.register(with: self.flutterEngine)

return super.application(application, didFinishLaunchingWithOptions: launchOptions)

}

}

**Step 5: Request Bluetooth and Location Permissions**

1. **Location Permissions**  
   Implement the following code to request location permissions in your app:

let locationManager = CLLocationManager()

switch CLLocationManager.authorizationStatus() {

case .notDetermined:

locationManager.requestWhenInUseAuthorization()

break

case .restricted, .denied:

// Disable location features

break

case .authorizedWhenInUse, .authorizedAlways:

// Enable location features

break

}

**2. Bluetooth Permissions**  
When the app is launched for the first time, the system will display a prompt to request Bluetooth permissions.

**Step 6: Integration Complete**

With the epstudio\_sdk\_flutter\_module integration and permissions in place, you can now use the functionalities provided by the epstudio\_sdk\_flutter\_module.

**7. Implementing WebSocket Functionality Using the Third-Party Library Starscream**

Here's how to implement WebSocket functionality using the Starscream library in an iOS application:

**1. Initialization**

First, you need to initialize the WebSocketManager:

var manager: WebSocketManager = WebSocketManager.shared

**2. Connecting to the Socket**

To establish a connection to the WebSocket server:

@objc func connect() {

self.manager.connectSocket("")

}

func connectSocket(\_ parameters: Any?) {

guard let url = URL(string: "http://127.0.0.1:3000/socket.io/?transport=websocket") else {

return

}

self.isActivelyClose = false

var request = URLRequest(url: url)

request.timeoutInterval = 5

// Add headers if necessary

webSocket = WebSocket(request: request)

webSocket?.delegate = self

webSocket?.connect()

}

**3. Connecting to a Device and Starting Data Collection**

To connect to a device and start collecting data:

@objc func start() {

let para = "42[\"deviceConnect\",[{\"address\":\"4A803816-60CB-4FA3-8B91-DF182622BE5D\",\"fs\":1000,\"channels\":[1,2]}]]"

self.manager.sendMessageData(para)

}

**4. Stopping Data Collection and Disconnecting the Device**

To stop collecting data and disconnect the device:

@objc func stop() {

let para = "42[\"deviceDisconnect\",[{\"address\":\"4A803816-60CB-4FA3-8B91-DF182622BE5D\"}]]"

self.manager.sendMessageData(para)

}

**5. Event Handling**

To handle different WebSocket events:

func didReceive(event: WebSocketEvent, client: WebSocket) {

switch event {

case .connected(let headers):

isConnected = true

delegate?.webSocketManagerDidConnect(manager: self)

// Handle successful connection logic, such as resending failed messages

print("WebSocket is connected: \(headers)")

case .disconnected(let reason, let code):

isConnected = false

let error = NSError(domain: reason, code: Int(code), userInfo: nil) as Error

delegate?.webSocketManagerDidDisconnect(manager: self, error: error)

self.connectType = .disconnect

if self.isActivelyClose {

self.connectType = .closed

} else {

self.connectType = .disconnect

destroyHeartBeat() // Stop the heartbeat timer

if self.isHaveNet {

reConnectSocket() // Reconnect

} else {

initNetWorkTestingTimer()

}

}

print("WebSocket is disconnected: \(reason) with code: \(code)")

case .text(let string):

delegate?.webSocketManagerDidReceiveMessage(manager: self, text: string)

// Use notifications for global data handling

let dic = ["text": string]

NotificationCenter.default.post(name: NSNotification.Name(rawValue: "webSocketManagerDidReceiveMessage"), object: dic)

print("Received text: \(string)")

case .binary(let data):

print("Received data: \(data.count)")

case .ping(\_):

print("ping")

case .pong(\_):

print("pong")

case .viabilityChanged(\_):

break

case .reconnectSuggested(\_):

break

case .cancelled:

isConnected = false

case .error(let error):

isConnected = false

handleError(error)

}

}

This implementation covers the initialization of the WebSocket connection, connecting to and disconnecting from devices, starting and stopping data collection, and handling various WebSocket events. This setup allows your iOS app to effectively manage WebSocket connections using the Starscream library.

**Second Integration Method: Using CocoaPods and Flutter SDK**

**1. Download and Integrate epstudio\_sdk\_flutter\_module**

1. **Download epstudio\_sdk\_flutter\_module to your local machine.**
2. **Integrate epstudio\_sdk\_flutter\_module:**
   * Locate the Podfile in your iOS application. If there isn’t one, create a Podfile.
   * In the Podfile, add the following code:

ruby

flutter\_application\_path = '../epstudio\_sdk\_flutter\_module' # Replace with the relative or absolute path to epstudio\_sdk\_flutter\_module

load File.join(flutter\_application\_path, '.ios', 'Flutter', 'podhelper.rb')

* + For each target that requires Flutter integration, execute:

ruby

target 'MyApp' do

install\_all\_flutter\_pods(flutter\_application\_path)

end

* + In the post\_install section of the Podfile, call flutter\_post\_install(installer):

ruby

post\_install do |installer|

flutter\_post\_install(installer) if defined?(flutter\_post\_install)

end

* + Run pod install to install the dependencies.

**2. Add iOS Permissions**

1. **Add necessary permissions** in your Info.plist file for location and Bluetooth:

xml

<key>NSBluetoothPeripheralUsageDescription</key>

<string>是否允许app使用蓝牙以便发送和接收数据？</string>

<key>NSBluetoothAlwaysUsageDescription</key>

<string>是否允许app使用蓝牙以便发送和接收数据？</string>

<key>NSLocationWhenInUseUsageDescription</key>

<string>是否允许app使用位置信息？</string>

<key>NSLocationAlwaysUsageDescription</key>

<string>是否允许app使用位置信息？</string>

<key>NSLocationAlwaysAndWhenInUseUsageDescription</key>

<string>是否允许app使用位置信息？</string>

<key>Privacy - NSLocationWhenInUseUsageDescription</key>

<string>是否允许app使用位置信息？</string>

<key>NSLocationUsageDescription</key>

<string>是否允许app使用位置信息？</string>

**3. Register FlutterEngine in AppDelegate.swift**

1. **Add the following code** to your AppDelegate.swift:

swift

import UIKit

import Flutter

import FlutterPluginRegistrant

@main

class AppDelegate: FlutterAppDelegate {

lazy var flutterEngine = FlutterEngine(name: "epstudio\_sdk\_flutter")

override func application(\_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

flutterEngine.run()

GeneratedPluginRegistrant.register(with: self.flutterEngine)

return super.application(application, didFinishLaunchingWithOptions: launchOptions)

}

}

**4. Request Bluetooth and Location Permissions**

1. **Location Permissions:**

swift

let locationManager = CLLocationManager()

switch CLLocationManager.authorizationStatus() {

case .notDetermined:

locationManager.requestWhenInUseAuthorization()

break

case .restricted, .denied:

// Disable location features

break

case .authorizedWhenInUse, .authorizedAlways:

// Enable location features

break

}

1. **Bluetooth Permissions:**
   * The system will prompt the user to grant Bluetooth permissions the first time the app is launched.

**5. Completion**

Once you have integrated epstudio\_sdk\_flutter\_module and enabled the required permissions, you can start using the module's features.

**6. Implement WebSocket Functionality Using Starscream**

1. **Initialization:**

swift

var manager: WebSocketManager = WebSocketManager.shared

1. **Connect to WebSocket:**

swift

@objc func connect() {

self.manager.connectSocket("")

}

func connectSocket(\_ parameters: Any?) {

guard let url = URL(string: "http://127.0.0.1:3000/socket.io/?transport=websocket") else {

return

}

self.isActivelyClose = false

var request = URLRequest(url: url)

request.timeoutInterval = 5

webSocket = WebSocket(request: request)

webSocket?.delegate = self

webSocket?.connect()

}

1. **Connect Device and Start Data Collection:**

swift

@objc func start() {

let para = "42[\"deviceConnect\",[{\"address\":\"4A803816-60CB-4FA3-8B91-DF182622BE5D\",\"fs\":1000,\"channels\":[1,2]}]]"

self.manager.sendMessageData(para)

}

1. **Stop Data Collection and Disconnect Device:**

swift

@objc func stop() {

let para = "42[\"deviceDisconnect\",[{\"address\":\"4A803816-60CB-4FA3-8B91-DF182622BE5D\"}]]"

self.manager.sendMessageData(para)

}

1. **Event Listener:**

swift

func didReceive(event: WebSocketEvent, client: WebSocket) {

switch event {

case .connected(let headers):

isConnected = true

delegate?.webSocketManagerDidConnect(manager: self)

print("WebSocket is connected: \(headers)")

case .disconnected(let reason, let code):

isConnected = false

let error = NSError(domain: reason, code: Int(code), userInfo: nil) as Error

delegate?.webSocketManagerDidDisconnect(manager: self, error: error)

self.connectType = .disconnect

if self.isActivelyClose {

self.connectType = .closed

} else {

self.connectType = .disconnect

destroyHeartBeat()

if self.isHaveNet {

reConnectSocket()

} else {

initNetWorkTestingTimer()

}

}

print("WebSocket is disconnected: \(reason) with code: \(code)")

case .text(let string):

delegate?.webSocketManagerDidReceiveMessage(manager: self, text: string)

let dic = ["text": string]

NotificationCenter.default.post(name: NSNotification.Name(rawValue: "webSocketManagerDidReceiveMessage"), object: dic)

print("Received text: \(string)")

case .binary(let data):

print("Received data: \(data.count)")

case .ping(\_):

print("ping")

case .pong(\_):

print("pong")

case .viabilityChanged(\_):

break

case .reconnectSuggested(\_):

break

case .cancelled:

isConnected = false

case .error(let error):

isConnected = false

handleError(error)

}

}

**RN Integration Method: Integrating Frameworks in Xcode**

**1. Generate Frameworks**

* **Switch to the Flutter module directory**: Navigate to the root directory of your Flutter module in the command line and run:

bash

flutter build ios-framework --cocoapods --output=some/path/MyApp/Flutter/

This command generates three integration packages: Debug, Profile, and Release. In the example, the integration packages are located in example\_RN/Flutter and can be used directly.

**2. Integrate Frameworks into Xcode**

* **Drag and drop the generated frameworks**: In Xcode, drag the generated frameworks from Finder (e.g., some/path/MyApp/Flutter/Release/) into your target project under **General > Frameworks, Libraries, and Embedded Content**.
  + **Embed Settings**: Select "Embed & Sign" in the Embed dropdown list.
  + **Update Build Settings**:
    - Add $(PROJECT\_DIR)/Flutter/Release/ to **Framework Search Paths**.

**Important**: Always use the Flutter.framework and App.framework from the same directory. Mixing frameworks from different directories (e.g., Profile/Flutter.framework and Debug/App.framework) can cause the app to fail to run.

**3. Add iOS Permissions**

* **Add the following permissions** to your Info.plist file for location and Bluetooth:

xml

<key>NSBluetoothPeripheralUsageDescription</key>

<string>是否允许app使用蓝牙以便发送和接收数据？</string>

<key>NSBluetoothAlwaysUsageDescription</key>

<string>是否允许app使用蓝牙以便发送和接收数据？</string>

<key>NSLocationWhenInUseUsageDescription</key>

<string>是否允许app使用位置信息？</string>

<key>NSLocationAlwaysUsageDescription</key>

<string>是否允许app使用位置信息？</string>

<key>NSLocationAlwaysAndWhenInUseUsageDescription</key>

<string>是否允许app使用位置信息？</string>

<key>Privacy - NSLocationWhenInUseUsageDescription</key>

<string>是否允许app使用位置信息？</string>

<key>NSLocationUsageDescription</key>

<string>是否允许app使用位置信息？</string>

**4. Register FlutterEngine at Project Startup**

**Swift Project (AppDelegate.swift):**

swift

import UIKit

import Flutter

import FlutterPluginRegistrant

@main

class AppDelegate: FlutterAppDelegate {

lazy var flutterEngine = FlutterEngine(name: "epstudio\_sdk\_flutter")

override func application(\_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

flutterEngine.run()

GeneratedPluginRegistrant.register(with: self.flutterEngine)

return super.application(application, didFinishLaunchingWithOptions: launchOptions)

}

}

**Objective-C Project (AppDelegate.h and AppDelegate.mm):**

**AppDelegate.h**

objc

#import <RCTAppDelegate.h>

#import <UIKit/UIKit.h>

#import "Flutter/Flutter.h"

@interface AppDelegate : RCTAppDelegate

@property (nonatomic, strong) FlutterEngine \*flutterEngine;

@end

**AppDelegate.mm**

objc

#import "AppDelegate.h"

#import <React/RCTBundleURLProvider.h>

#import <FlutterPluginRegistrant/GeneratedPluginRegistrant.h>

#import <epstudio\_sdk\_flutter/epstudio\_sdk\_flutter-Swift.h>

@implementation AppDelegate

- (BOOL)application:(UIApplication \*)application didFinishLaunchingWithOptions:(NSDictionary \*)launchOptions {

self.moduleName = @"MyRNApp";

self.initialProps = @{}; // Custom initial props for React Native

self.flutterEngine = [[FlutterEngine alloc] initWithName:@"my flutter engine"];

[self.flutterEngine run]; // Runs the default Dart entrypoint

[GeneratedPluginRegistrant registerWithRegistry:self.flutterEngine]; // Connects plugins

return [super application:application didFinishLaunchingWithOptions:launchOptions];

}

@end

**5. Run the Project**

After completing the steps above, you can run your project, and it should be integrated with the epstudio\_sdk\_flutter\_module and ready to use the provided functionalities.

**Socket Data Descriptions**

**1. Collected Data**

**Original Data:**

plaintext

'42["data",{"timestamp":1689313361883,"dataType":"EP","data":[{"deviceId":"F9:DC:B2:3E:B4:BD","timestamp":1689313361832,"data":{"1":[-0.63,-5.25],"2":[0.000,-0.0046]},"packageNumbers":[122,123],"performance":{"packageLossRate":0.0}}]}]'

**Description:**

json

{

"timestamp": "Current push timestamp",

"dataType": "EP: Collected Data",

"data": [{

"deviceId": "Device MAC address",

"timestamp": "Device Bluetooth upload packet timestamp",

"data": {

"Channel number": "List of data"

},

"packageNumbers": ["List of packet numbers included in the current push data"],

"performance": {

"packageLossRate": "Current packet loss rate (if packet numbers are consecutive, it can be considered as delayed packet loss)"

}

}]

}

**2. Device List**

**Original Data:**

plaintext

'42["DEVICE",[{"address":"F9:DC:B2:3E:B4:BD","name":"MH11202.01.8","isConnecting":true,"isCollecting":false,"isStimulating":false,"isLeadOffDetecting":false,"samplingRate":1000,"samplingRates":[250,500,1000,2000,4000,8000],"gain":1,"gains":[1,2,4,6,8,12,24],"batteryLevel":96,"deviceMode":0,"supportCollect":true,"supportStimulate":false,"supportLeadOff":false,"enableCollect":true,"enableStimulate":false,"isMutex":true,"channels":[{"id":"1","name":"1","supportCollect":true,"supportStimulate":false,"isStimulating":false,"isCollecting":false,"isChecked":true,"leadOffStatus":null},{"id":"2","name":"2","supportCollect":true,"supportStimulate":false,"isStimulating":false,"isCollecting":false,"isChecked":true,"leadOffStatus":null}],"leadOffStatusNegative":null}]]'

**Description:**

json

[{

"address": "Device MAC address",

"name": "Device name",

"isConnecting": "Is the device connected",

"isCollecting": "Is data being collected",

"isStimulating": "Is stimulation active",

"isLeadOffDetecting": "Is lead-off detection active",

"samplingRate": "Current sampling rate",

"samplingRates": ["List of supported sampling rates"],

"gain": "Current gain",

"gains": ["List of supported gains"],

"batteryLevel": "Current battery level",

"deviceMode": "Device mode",

"supportCollect": "Does the device support data collection",

"supportStimulate": "Does the device support stimulation",

"supportLeadOff": "Does the device support lead-off detection",

"enableCollect": "Is data collection enabled",

"enableStimulate": "Is stimulation enabled",

"isMutex": "Is the device in mutex mode",

"channels": [{

"id": "Channel number",

"name": "Channel name",

"supportCollect": "Does the channel support data collection",

"supportStimulate": "Does the channel support stimulation",

"isStimulating": "Is the channel stimulating",

"isCollecting": "Is data being collected on this channel",

"isChecked": "Is the channel selected",

"leadOffStatus": "Lead-off detection status"

}],

"leadOffStatusNegative": "Negative lead-off status"

}]

**3. Alarm Information**

**Original Data:**

plaintext

'42["WARN",{"createTime":"2023-07-14 13:42:44","alarmType":"丢包率过高","alarmLevel":"高","deviceId":"F9:DC:B2:3E:B4:BD","deviceName":"MH11202.01.8","alarmContent":"设备[MH11202.01.8]数据采集丢包率高于10%！","alarmValue":52.81}]'

**Description:**

json

{

"createTime": "Time of the alarm",

"alarmType": "Type of alarm (e.g., Low battery, Device disconnected, High packet loss rate)",

"alarmLevel": "Alarm level (Low, High)",

"deviceId": "Device MAC address",

"deviceName": "Device name",

"alarmContent": "Content of the alarm",

"alarmValue": "Value associated with the alarm (e.g., packet loss rate, battery level)"

}

**4. Error Information**

**Original Data:**

plaintext

'42["ERROR","Failed to connect device F9:DC:B2:3E:B4:BD"]'

**5. Information**

**Original Data:**

plaintext

'42["INFO","Device F9:DC:B2:3E:B4:BD loss 1 packages"]'

**6. Data Record Information**

**Original Data:**

plaintext

'42["RECORD",{"storagePath":"/storage/emulated/0/Android/data/com.neuramatrix.epstudio\_sdk\_flutter\_module.host/files/2023\_07\_30\_170845","timeStamp":"2023-07-30 17:08:45","updateTime":"2023-07-30 17:08:45","endTime":"2023-07-30 17:18:45"}]'

**Description:**

json

{

"storagePath": "File storage path",

"timeStamp": "Recording start time",

"updateTime": "Update time",

"endTime": "Recording end time"

}

**7. Status Information**

**Original Data:**

plaintext

'42["GLOBAL\_INFO",{"blueAdapterState":true}]'

**Description:**

json

{

"blueAdapterState": "Is Bluetooth enabled"

}

**8. Analysis Results**

**Original Data:**

plaintext

'42["ALGORITHM",{"type":"SLEEP","path":"/storage/emulated/0/Android/data/com.neuramatrix.epstudio\_sdk\_flutter\_module.host/files/test02","data":{"times":[1718959952313,1718959982311,1718960012310,1718960042307,1718960072305,1718960102304,1718960132302,1718960162300,1718960192299,1718960222354,1718960252352,1718960282350,1718960312347,1718960342345,1718960372344,1718960402342,1718960432340,1718960462339,1718960492336,1718960522305,1718960552304,1718960582302,1718960612300,1718960642298],"datas":[0,1,2,2,1,0,0,1,1,1,1,1,1,1,1,2,2,2,2,2,2,3,3,0],"score":71.26632523829996}}]'

**Description:**

json

{

"type": "Type of analysis (e.g., SLEEP for sleep analysis)",

"path": "Data directory",

"data": {

"times": ["List of timestamps"],

"datas": ["List of corresponding analysis results for each timestamp"],

"score": "Analysis score"

}

}

**Sleep Analysis Results:**

* times: List of timestamps.
* datas: List of corresponding sleep stages for each timestamp, where:
  + 0: Wake
  + 1: N1 (light sleep)
  + 2: N2 (deeper sleep)
  + 3: N3 (deep sleep)
  + 4: REM (rapid eye movement sleep)
* score: Overall sleep analysis score.

**Sending Data Descriptions**

**1. Scan Devices**

**Original Data:**

plaintext

'42["startScan"]'

**Description:**

* Scans for devices for 3 seconds by default.

**2. Stop Scanning**

**Original Data:**

plaintext

'42["stopScan"]'

**Description:**

* Stops the scanning process.

**3. Connect Device**

**Original Data:**

plaintext

'42["deviceConnect",["F9:DC:B2:3E:B4:BD"]'

**Description:**

* Connects to the device with the specified MAC address.

**4. Disconnect Device**

**Original Data:**

plaintext

'42["deviceDisconnect",["F9:DC:B2:3E:B4:BD"]'

**Description:**

* Disconnects the device with the specified MAC address.

**5. Start Data Collection**

**Original Data:**

plaintext

'42["deviceCollect",["F9:DC:B2:3E:B4:BD"]'

**Description:**

* Starts data collection from the device with the specified MAC address.

**6. Stop Data Collection**

**Original Data:**

plaintext

'42["deviceStopCollect",["F9:DC:B2:3E:B4:BD"]'

**Description:**

* Stops data collection from the device with the specified MAC address.

**7. Set Channels**

**Original Data:**

plaintext

'42["setChannelStatus",{"address":"F9:DC:B2:3E:B4:BD","channels":[1,2]}]'

**Description:**

json

{

"address": "Device MAC address",

"channels": "List of channels to be enabled (options from DEVICE.channels)"

}

**8. Set Sampling Rate**

**Original Data:**

plaintext

'42["setSamplingRate",{"address":"F9:DC:B2:3E:B4:BD","value":1000}]'

**Description:**

json

{

"address": "Device MAC address",

"value": "Sampling rate value (options from DEVICE.samplingRates)"

}

**9. Set Gain**

**Original Data:**

plaintext

'42["setGain",{"address":"F9:DC:B2:3E:B4:BD","value":1}]'

**Description:**

json

{

"address": "Device MAC address",

"value": "Gain value (options from DEVICE.gains)"

}

**10. Start Recording Data**

**Original Data:**

plaintext

'42["startRecord",{"folderName":null,"position":null}]'

**Description:**

json

{

"folderName": "Name of the folder for recording data (default to current time if null)",

"position": "Absolute path of the root directory for recording data (default to 'files' in Android or 'Documents' in iOS if null)"

}

**11. Stop Recording Data**

**Original Data:**

plaintext

'42["stopRecord"]'

**Description:**

* Stops the data recording process.

**12. Request Global Information**

**Original Data:**

plaintext

'42["globalInfo"]'

**Description:**

* Requests global status information.

**13. Run Algorithm**

**Original Data:**

plaintext

'42["algorithm",{"type":"SLEEP","path":"/storage/emulated/0/Android/data/com.neuramatrix.epstudio\_sdk\_flutter\_module.host/files/2023\_07\_30\_170845"}]'

**Description:**

json

{

"type": "Analysis type, e.g., SLEEP for sleep analysis",

"path": "Absolute path of the root directory for recorded data"

}

**Force Detection Data Parsing**

**Force Detection Process**

1. **Prepare 2 seconds of collected data**: For each channel, collect 2 seconds of data and store it in a collection.
2. **Calculate RMS (Root Mean Square)**:
   * **Square and Sum**: Square each value and sum them up.
   * **Average**: Divide the sum by the number of data points.
   * **Square Root**: Take the square root of the average.
3. **Round RMS**: Round the RMS value to the nearest integer.
4. **Prepare Force Value**: Initialize the force value to 400. If the value is 0, set it to 0.001.
5. **Calculate Force Index Percentage**: Compute (RMS / Force Value) \* 100 as the force index percentage. If the result is greater than 100, set it to 100.

**Java Implementation**

**Key Components:**

1. **Device ID (deviceId):**
   * **This is a unique identifier for the device, represented as a string (e.g., "EE:20:B7:1B:99:56").**
2. **Channel ID (channelId):**
   * **This is the identifier for the specific channel on the device that is being monitored or analyzed, represented as an integer (e.g., 1).**
3. **Current Force Value (value):**
   * **This is the initial or current force value used in calculations, represented as a double (e.g., 20000).**
4. **Current Force Index (currentValue):**
   * **This stores the calculated force index, which is determined based on the RMS of the data and the force value, represented as a double (e.g., 0 initially).**
5. **Example Data (exampleData):**
   * **This is a map that holds a list of data points for each channel. In this case, it only contains data for channel 1. The data consists of a series of double values that represent the collected data points.**

private Map<Integer, List<Double>> exampleData = Map.of(1, List.of(  
 4174.309342820226, 641.6818009610215, -2955.0298319334543, -7425.757822586362, -12313.161503551411,  
 -16534.50133911075, -18938.389993882243, -18389.639335770655, -14616.95358602464, -9157.112444717612,  
 -4103.8172593125055, 104.82709821981553, 4634.607113759717, 10147.458115444737, 15542.122229976623,  
 18960.52468111404, 19327.380007640633, 16602.085562882246, 11670.630530186929, 6319.8769007886585,  
 2019.323958402878, -1427.7067706032103, -5240.140946148029, -9918.41029733645, -14602.13105054028,  
 -17912.484026700375, -18853.42085419537, -16933.266016441456, -12483.617812141893, -7026.002541529626,  
 -2106.58519139298, 2187.9434391401883, 6666.216475568173, 11584.107962590875, 16254.282152116968,  
 19265.52142880656, 19096.87207930378, 15466.41423890114, 10014.662523556617, 4917.013503747963,  
 874.7379617042607, -2946.5808225424116, -7278.779427258407, -11896.832930523378, -15995.46954334018,  
 -18404.512683145585, -17960.380797053076, -14622.954943184159, -9765.616182979546, -4870.984283614787,  
 -333.4476919011722, 4273.561089906783, 9129.914078743714, 13849.366773058777, 17780.282459608745,  
 19608.69131042328, 17689.170247920294, 12377.31468233408, 6531.5919118379825, 2274.5276108329126,  
 -1039.1059741991776, -4919.022747577066, -9533.8892277095, -13983.622886292287, -17438.974830073246,  
 -19023.64281822811, -17569.69310409698, -13025.59186164895, -7333.011878083926, -2448.8129803266347,  
 1687.1945206953096, 6226.091948195848, 11357.496242032954, 15995.450917228067, 18889.40548816169,  
 19152.58825136529, 16283.772374254157, 11034.676981684403, 5586.03067124213, 1438.1302375289524,  
 -2115.976103074514, -6483.121890853792, -11610.163964878448, -16107.92305262448, -18705.121712104825,  
 -18617.06408327128, -15496.894724404381, -10279.122098393302, -4947.560631238346, -488.80440645349154,  
 3788.529760575533, 8605.226227543695, 13585.605894326698, 17621.726855804125, 19306.136587894696,  
 17546.87270398566, 12826.431841269077, 7297.909414883907, 2770.2613965605415, -839.9260822049255,  
 -4602.116031979669, -9009.054448053663, -13682.688255464702, -17572.79649773566, -19107.341371976043,  
 -17268.253773775534, -12821.654985795205, -7659.087198392779, -2956.6182173111447, 1457.070503564857,  
 6161.919444833868, 11198.743757525532, 16019.921310618098, 19467.35008641795, 19737.47178306381,  
 15981.181499884697, 10049.52782900323, 4821.116912967016, 1049.0257919967407  
));  
// ---------------------------- Data Collection Parsing Class ----------------------------

@Data

@NoArgsConstructor

@AllArgsConstructor

public class EPData {

private String deviceId;

private Long timestamp;

private Map<Integer, List<Double>> data;

@Nullable

private Integer packageNumber;

public EPData(String deviceId, Long timestamp, Map<Integer, List<Double>> data) {

this.deviceId = deviceId;

this.timestamp = timestamp;

this.data = data;

}

}

// ------------------------------ Calculation Section ------------------------------

// All collected data received

List<EPData> dataList = Lists.newArrayList(

new EPData(

"EE:20:B7:1B:99:56",

1690876623785L,

exampleData

)

);

// Filter the collected data to find the specific device's data

List<EPData> filterList = dataList.stream().filter(item -> Objects.equals(item.getDeviceId(), deviceId)).toList();

if (filterList.isEmpty()) return;

EPData deviceData = filterList.get(0);

// Collected data from the specified channel on the device

List<Double> channelValue = Lists.newArrayList(deviceData.getData().get(channelId));

// Default RMS value

double rms = 0.0;

// If the channel data is not empty, proceed with calculation (if empty, exit the calculation and retain the current force index)

if (!channelValue.isEmpty()) {

// Sort the collected data

Collections.sort(channelValue);

// Calculate RMS (Root Mean Square)

// 1. Sum of squares

// Accumulate the squares of each value

double x = 0;

for (double num : channelValue) {

x += num \* num;

}

// 2. Average

// Sum of squares / number of data points in the channel

double average = x / channelValue.size();

// 3. Square root

if (average < 0) {

// If the square root result is NaN, set rms to 0

rms = 0;

} else {

// If the square root result is not NaN, set rms to the square root value

rms = Math.sqrt(average);

}

// Discard the fractional part of rms

rms = (int) rms;

// If the force value is 0, set it to 0.001

if (value == 0) {

value = 0.001;

}

// rms / force value (rounded to 6 decimal places)

String tempStr = String.format("%.6f", (rms / value));

if (Double.parseDouble(tempStr) >= 1) {

// If the division result is greater than or equal to 1, the force index is 100%

currentValue = 100;

} else {

// If the division result is less than 1, multiply by 100

currentValue = Double.parseDouble(tempStr) \* 100;

}

log.info("Device {} Channel {} Current Force Index: {}", deviceId, channelId, currentValue);

}

// ---------------------------- Dart Code ----------------------------

// Device ID

String deviceId = widget.channelParams['deviceId'];

// Channel ID

int channelId = widget.channelParams['channelId'];

// All collected data received

List<EPData> dataList = globalState.dataList;

// Find the specific device's data from the received data

EPData deviceData = dataList.firstWhere((item) => item.deviceId == deviceId);

// Collected data from the specified channel on the device

List channelValue = jsonDecode(jsonEncode(deviceData.data[channelId]));

// Default RMS value

double rms = 0.0;

// If the channel data is not empty, proceed with calculation (if empty, exit the calculation and retain the current force index)

if (channelValue.isNotEmpty) {

// Sort the collected data

channelValue.sort();

// Calculate RMS (Root Mean Square)

// 1. Sum of squares

// Accumulate the squares of each value

double x = channelValue.reduce((value, element) {

return value + element.abs() \* element.abs();

});

// 2. Average

// Sum of squares / number of data points in the channel

double average = x / channelValue.length;

// 3. Square root

if (sqrt(average).isNaN) {

// If the square root result is NaN, set rms to 0

rms = 0;

} else {

// If the square root result is not NaN, set rms to the square root value

rms = sqrt(average);

}

// Discard the fractional part of rms

rms = double.parse(rms.toStringAsFixed(0));

// If the force value is 0, set it to 0.001

if (value == 0) {

value = 0.001;

}

// rms / force value (rounded to 6 decimal places)

String tempStr = (rms / value).toStringAsFixed(6);

if (double.parse(tempStr) >= 1) {

// If the division result is greater than or equal to 1, the force index is 100%

currentValue = 100;

} else {

// If the division result is less than 1, multiply by 100

currentValue = NumUtil.multiply(double.parse(tempStr), 100);

}

}