AHEAD OF WHAT'S POSSIBLE™

Analog Devices, Inc.

www.analog.com

Integration Guide - Sdk With Flutter Application

Revision List

Table 1: Revision List

Revision	Date	Description
0. 1	5-April-2021	Initial Draft
0.2	8-April-2020	Reviewed and updated formatting
1.0	9-April-2020	Baselined for 1.0.0

Copyright, Disclaimer Statements

Copyright Information

Disclaimer

Table of Contents

List of Figures

List of Tables

1 Introduction

This document is a getting started guide for the users to use sdk in their flutter application. It gives details on the package, features, and explains the sdk communication with sample flutter application.

The SDK provides an easy to use API for the ADI Study Watch device. The structure of the flutter application project is as follows:

HealthWearable (Root Directory)	sdk- Contains header and source files
	flutterClasses- Contains Wrapper files for sdk
	lib- Application related files for UI and Functionality
	android - Generated platform code for android
	iOS - Generated platform code for iOS

Figure 1: Folder structure of Flutter Application

1.1 Purpose

This purpose of this document is to explain the SDK architecture and how an application can use the SDK to communicate with the firmware.

1.2 Scope

The different sections covered in this document are:

- Introduction
- Setting up SDK for Flutter App
- Steps to use SDK to communicate with Firmware
- Understanding SDK

2 Setting up the SDK for Flutter

The SDK is not designed to talk directly to the hardware. It requires the user to provide functions for transmitting to and receiving data from the hardware.

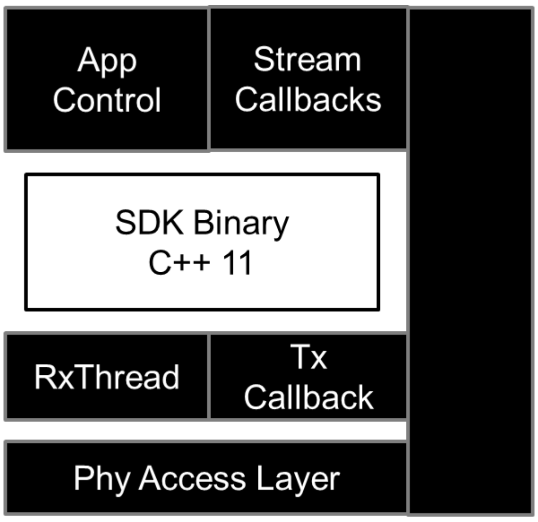


Figure 2: Typical Application Structure

To use the sdk in the flutter application, the first step is to include the header and source files to the project.

The SDK source and header files can be found in ***'healthwearable\sdk'*** folder.

2.1 Android

- Include the header and source file in cmake to build dynamic library.
- Below is the cmake file to create the shared library for android.
- Include this CMakeLists.txt file to the ***'healthwearable\android'*** folder

```
cmake_minimum_required(VERSION 3.4.1)  # for example

set(PROJECT_BASE_NAME "gen4sdk")

project (${PROJECT_BASE_NAME})

set(INC_DIR "../sdk/inc")

set(FLUTTER_INC_DIR "../flutterClasses/header")

set(CPP_SOURCE_DIR "../sdk/src")

set(FLUTTER_CPP_SOURCE_DIR "../flutterClasses/source")

file(GLOB CPP_SOURCES "${CPP_SOURCE_DIR}/*" "${FLUTTER_CPP_SOURCE_DIR}/*")

include_directories(${INC_DIR})

include_directories(${FLUTTER_INC_DIR})

add_library( ${PROJECT_BASE_NAME}

             # Sets the library as a shared library.
             SHARED
             # Provides a relative path to your source file(s).
             ${CPP_SOURCES})
```

- Finally, add an externalNativeBuild section to ‘*healthwearable\android\app\build.gradle’.*

```
 externalNativeBuild {
       // Encapsulates your CMake build configurations.
        cmake {
            // Provides a relative path to your CMake build script.
            path "../CMakeLists.txt"
        }
}
```

2.2 iOS

- On iOS we need to statically link the header and source files.
- Open ***healthwearable\ios\Runner.xcodeproj*** through xCode Editor. Add the header and source files to project.

3 Flutter Sdk Wrapper

A sdk wrapper, written in dart, is used for interaction between the application and the SDK API's

- As a first step we need to load the sdk library for android and iOS.

```
final DynamicLibrary nativeGen3SDK = Platform.isAndroid
    ? DynamicLibrary.open("libgen4sdk.so")
    : DynamicLibrary.process();
```

- Create the object by loading the library. This object is used as a reference for calling the sdk functions. Refer the code snippet in ***'lib\services\sdk_lib\fl_sdk_wrapper.dart'*** file
- FFI package allows us to call native C/C++ functions on both iOS and Android platforms without using Platform Channels.
- Below is an example of using ***'start_stream'/'stop_stream'*** APIs in dart.

CPP: (flutterClasses\header\CPPtoC_Wrapper.h)

```
void start_stream(SENSORS sensor);

void stop_stream(SENSORS sensor);
```

Flutter: (lib\services\sdk_lib\fl_sdk_wrapper.dart)

```
typedef L1_Func = Uint8 Function(Uint8 stream);
typedef l1_func = int Function(int stream);
final l1StreamStart =
    nativeGen3SDK.lookupFunction<L1_Func, l1_func>("start_stream");

final l1StreamStop =
    nativeGen3SDK.lookupFunction<L1_Func, l1_func>("stop_stream");
```

4 Steps to use the SDK in Flutter Application

Users need to implement the below methods to establish communication between SDK and flutter application.

4.1 BLE Communication

- The application communicates with the watch through BLE connection.
- The app uses the *flutter_blue* package for BLE connection.
- Follow the below steps to implement the BLE layer.

4.1.1 Import Package

```
import 'package:flutter_blue/flutter_blue.dart';
```

4.1.2 Obtain an instance

```
final FlutterBlue bleInstance = FlutterBlue.instance;
```

4.1.3 Scan for Devices

```
// Scanning for 5 seconds
bleInstance.startScan(timeout: Duration(seconds: 5));

// Listen to scan results
var subscription = bleInstance.scanResults.listen((results) {
    // do something with scan results
    for (ScanResult r in results) {
        print('${r.device.name} found! rssi: ${r.rssi}');
    }
});
```

4.1.4 Connect

```
  // Connect to the device
await device.connect();
```

4.1.5 Discover Services and Characteristics

Discover for the below service and characteristics is present in the studywatch.
Study Watch uses the Nordic UART Service **is 6E400001-B5A3-F393-E0A9-E50E24DCCA9E**
This service exposes two characteristics: one for transmitting and one for receiving (as seen from the peer).

- **RX Characteristic (UUID: 6E400002-B5A3-F393-E0A9-E50E24DCCA9E)**
  The peer can send data to the device by writing to the RX Characteristic of the service.
- **TX Characteristic (UUID: 6E400003-B5A3-F393-E0A9-E50E24DCCA9E)**
  If the peer has enabled notifications for the TX Characteristic, the application can send data to the peer as notifications.

```
 _services = await bleDevice?.discoverServices();
       for (BluetoothService service in _services) {
         var characteristics = service.characteristics;
         for (BluetoothCharacteristic c in characteristics) {
           if (c.uuid == writecharacteristic) {
             //print('\nble data instance  Write Characteristic\n');
           }
           if (c.uuid == readcharacteristic) {
             _setNotifyforReadBleData(c);
           }
         }
```

4.1.6 Write SDK Packet to BLE

Application uses the below API for writing the packet from SDK to BLE.

```
Future writeData(List<int> value) async {
   for (BluetoothService service in _services) {
     var characteristics = service.characteristics;
     for (BluetoothCharacteristic c in characteristics) {
       if (c.uuid == writecharacteristic) {
         await c.write(value, withoutResponse: true);
       }
     }
   }
 }
```

4.1.7 Receiving BLE Callback

Set the notification of BLE receiving characteristics to true. So that whenever the packet is received from the watch, it will trigger the BLE receiver callback.

```
  void _setNotifyforReadBleData(BluetoothCharacteristic charc) async {
    await charc.setNotifyValue(true);
    _readDataSubscription = charc.value.listen((value) {
      print('r:$value');
      if (_rxController.hasListener) {
        _rxController.add(value);
      } else {
        //print('rxListner not registered');
      }
    });
  }
```

4.2 Transmission (Tx) Port Listener

- Data from SDK to BLE layer is sent via a transmission port listener.
- Application calls APIs in SDK when data is to be transmitted to the watch.
- Only complete M2M2 packets should be sent to the watch
- The SDK will only provide complete M2M2 packets to the transmission listener. This function transmits over the physical layer.

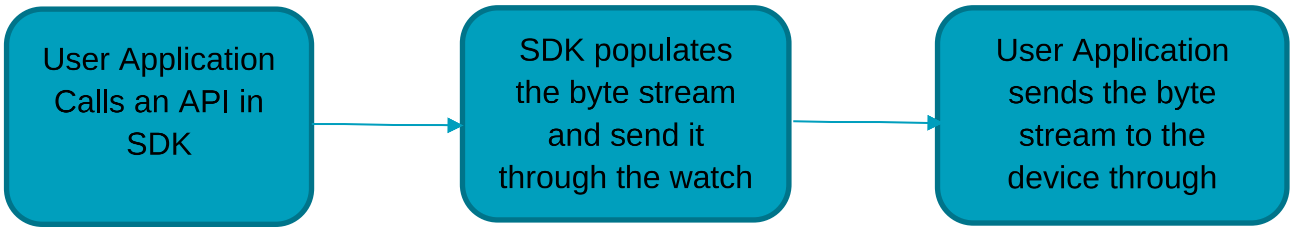

User Application Calls an API in SDK
SDK populates the byte stream and send it through the watch
User Application sends the byte stream to the device through

Figure 3: Tx Flow

```
if (initializeApi(NativeApi.initializeApiDLData) != 0) {
      throw "Failed to initialize Dart API"; }
final txPortListener = ReceivePort()
      ..listen((data) {
        List<String> receivedStrData = data.split(":");
        receivedStrData.removeLast();

        List<int> cmdPacket =
            receivedStrData.map((data) => int.parse(data)).toList();
      bledatainstance.writeData(cmdPacket);
      });

final int txSendPort = txPortListener.sendPort.nativePort;
registerSendPort(txSendPort);
```

4.3 BLE Receiver Callback

Once the packet is sent through to the application layer, the response of the M2M2 packet will be received at the BLE receiving callback listener. Application layer then dispatches the packet to the SDK for decoding.

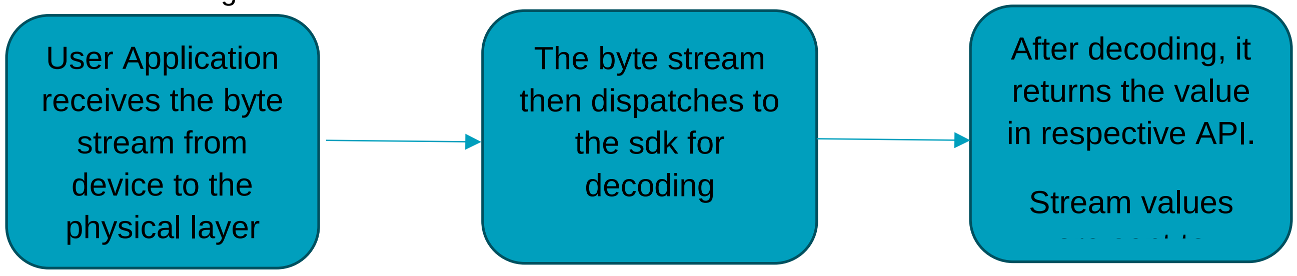

User Application receives the byte stream from device to the physical layer
The byte stream then dispatches to the sdk for decoding
After decoding, it returns the value in respective API.
Stream values

Figure 4: Rx Flow

```
void _setNotifyforReadBleData(BluetoothCharacteristic charc) async {
    await charc.setNotifyValue(true);
    _readDataSubscription = charc.value.listen((value) {
      print('r:$value');
      dispatchData(value);
    });
}

void dispatchData(List<int> bytes) {
    if (bytes.length == 0) {
      return;
    }
      Pointer<Uint8> rcvdDataptr = allocate<Uint8>(count: bytes.length);

    for (int i = 0; i < bytes.length; i++) {
      rcvdDataptr[i] = bytes[i];
    }

    var dispatchSDK = nativeGen3SDK
        .lookupFunction<DISPATCH_FUNC, dispatch_Func>("dispatch_SDK");

    dispatchSDK(rcvdDataptr, bytes.length);
    free(rcvdDataptr);
  }
```

4.4 Initialize SDK

Once the BLE connection and transmission callback is implemented, the SDK needs to be initialized with the respective platform. Platforms supported in this release include.

```
final int platform = Platform.isAndroid
        ? SDK_PLATFORM.android.index
        : SDK_PLATFORM.ios.index;
```

```
initializeSDK(platform);
```

Once the platform is set, the application is ready to call any sdk request to communicate with watch.

5 Getting Started With SDK

5.1 Applications Details

The SDK exports several classes that allow interaction with the watch. The Table below shows the various applications that are available. See ***sdk/doc/doxygen/html/classwatch.html*** for more details.

These apps expose all the M2M2 commands of the watch. Through these classes, a developer has full access to the watch.

Accelerometer application	adxl_app
EDA application	eda_app
ECG application	ecg_app
AD5940 application	ad5940_app
Temperature application	temperature_app
ADPD application	adpd_app
PPG application	ppg_app
SyncPPG application	syncppg_app
ADPD4K application	adpd4000_app
PM application	pm_app
File System application	fs_app
Pedometer application	pedometer_app
BCM application	bcm_app
Display application	display_app

Table 2: Applications List

5.2 L1 Commands

These represent Level 1, or L1, commands. L1 commands provide primitive/fundamental access to the watch. Each L1 command is mapped to an equivalent M2M2 command.

They are essentially primitives and can be segmented as follows:

- System apps
  - pm_app: PM core system application
  - fs_app: file system application
- PPG related apps
  - adxl_app: Accelerometer ADXL362 application
  - adpd_app: ADPD4K optical sensor application
  - ppg_app: Heart rate measurement application
  - syncppg_app: Synchronized of adpd and adxl data application
- EDA, ECG and Temperature apps
  - EDA_app: Electrodermal activity application
  - ECG_app: Electrocardiogram sensor application

  - temperature_app: Temperature application

5.3 L2 commands

L2 commands encapsulate a group of L1 commands to make the API usage easier.
For example, consider the following L1 command set to start PPG,

Start PPG via L1:

```
l1LoadADPDcfg(
            SENSOR_ADPD4000_DEVICE_ID.ADPD4000_DEVICE_4000_G.index + 40);
doClockCalibration([0x08]); //chipIdRegAddress = 0x08
l1SetPPGLcfg(ppgLcfgId[PPG_LCFG_ID.ADPD4000]);
l1FSSubscribe(sensor.index);
l1StreamStart(SENSORS_SDK.PPG.index);
```

Stop PPG via L1:

```
l1StreamStop(SENSORS_SDK.PPG.index);
l1FSUnSubscribe(sensor.index);
```

The equivalent L2 commands are,
Start PPG via L2:

```
l2startppg();
```

Stop PPG via L2:

```
l2stopppg();
```

5.4 Example of using SDK to stream ECG Data

The L2StreamTest can be used to get stream data. In this case, we focus on the ECG data while using our callback function for ECG
Where ECG carries the heart rate and ECG signal.

Below explains the arguments of stream L2streamtest

5.4.1 Register the ECG port listener for reading the ECG stream values

```
void ecgPortListener() {
    final interactiveECGCppRequests = ReceivePort()
      ..listen((data) {
        List<String> receivedStrData = data.split(",");

        double _timestamp = double.parse(receivedStrData[0]);
        double _ecgvalue = double.parse(receivedStrData[1]);
        double _bpm = double.parse(receivedStrData[2]);

        if (_ecgStreamController?.hasListener == true) {
          _ecgStreamController?.add(new ECGdataModel(
            timeStamp: _timestamp,
            ecgData: _ecgvalue,
            leadData: 0.0,
            bpm: _bpm,
          ));
        }
      });
    final int ecgNativePort = interactiveECGCppRequests.sendPort.nativeP
ort;
    registerECGPort(ecgNativePort);
  }
```

5.4.2 Start ECG via L2

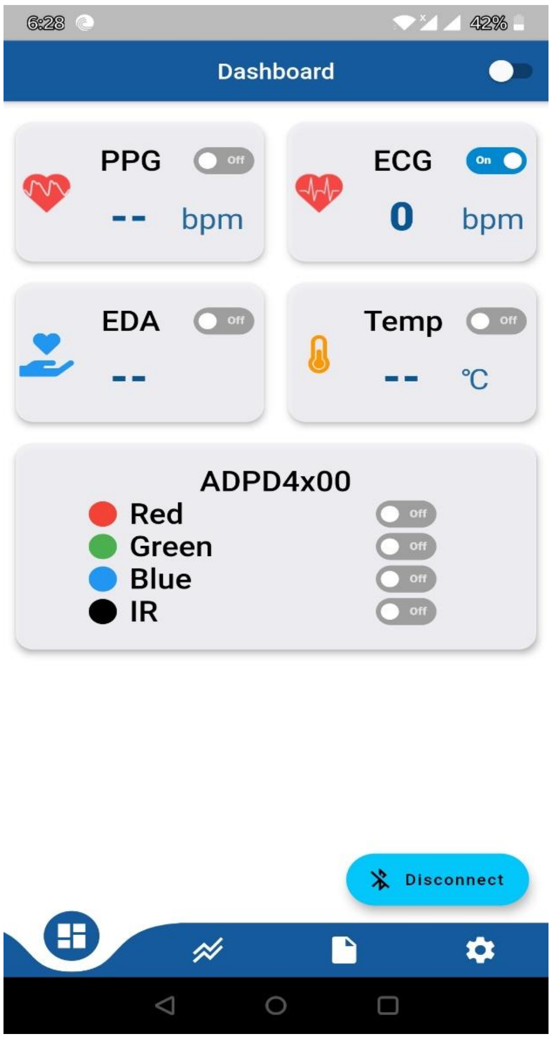
```
ecgPortListener();

l2startecg();
```

5.4.3 Stop ECG via L2

```
l2stopecg();
```

5.4.4 Sample Flutter ECG Plot

In the flutter sample, ECG stream is started using L2 commands and its value is displayed in the dashboard.



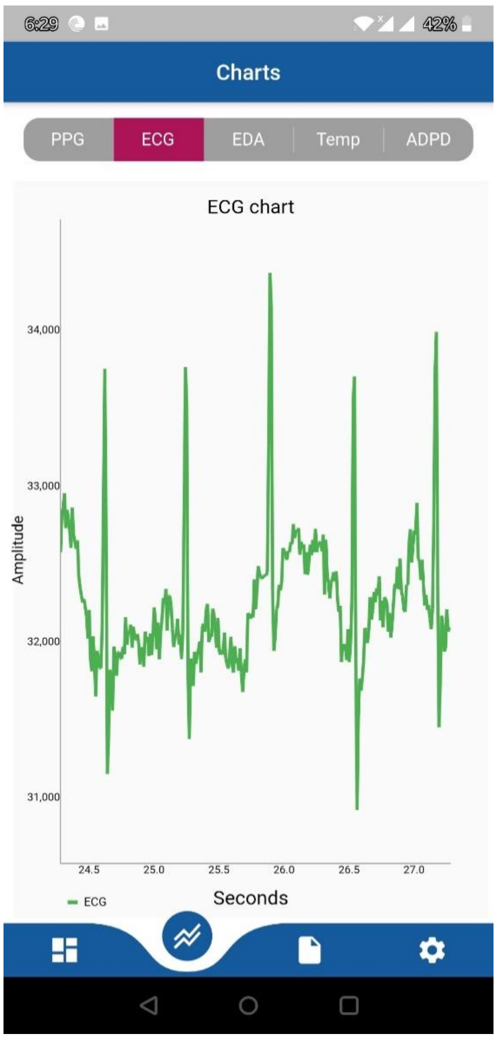


Figure 5: Flutter App ECG Stream Plot

Terminology

Table 3: Terminology

Term	Description
SDK	Software Development Kit
PPG	Photoplethysmography
ECG	Electrocardiography
SPO2	Saturated Peripheral Oxygen
ODR	Output Data Rate
MCU	Micro Controller Unit
AGC	Automatic Gain Control
HRM	Heart Rate Measurement
MIPS	Million Instructions Per Second

References

Table 4: References

Reference No.	Description
[1]	SDK documentation – sdk/doc/doxygen/
[2]	Applications Wavetool
[3]	ADI_VSM_Watch_IV_Firmware
[4]	Healthcare_Watch_Datasheet