

ANALOG DEVICES, INC.

[www.analog.com](http://www.analog.com/)

|  |
| --- |
| **Getting Started With Gen4SDK Framework** |
| REV 0.1, February 2021 |

Table of Contents

[1 Introduction 5](#_Toc64460288)

[2 Setting up the SDK. 5](#_Toc64460289)

[BLE Communication 6](#_Toc64460290)

[Transmission (Tx) Callback 6](#_Toc64460291)

[Receiving Callback 8](#_Toc64460292)

[Initialize SDK 9](#_Toc64460293)

[3 Getting Started With SDK 9](#_Toc64460294)

[L1 Commands 10](#_Toc64460295)

[Basic SDK Operation of L1 Commands 11](#_Toc64460296)

[L1 vs L2 commands 11](#_Toc64460297)

[4 Stream ADXL Data from the App using L1 Commands 12](#_Toc64460298)

[5 Stream ECG Data from app using L2 Commands 14](#_Toc64460299)

List of Figures

[Figure 1: Typical App Arrangement 6](#_Toc64460300)

[Figure 2: Transmission callback from iOS samples 7](#_Toc64460301)

[Figure 3: Assigning the Callback 7](#_Toc64460302)

[Figure 4: Receiving Callback of BLE 8](#_Toc64460303)

[Figure 5: Initialize SDK 9](#_Toc64460304)

[Figure 6: Application list 10](#_Toc64460305)

[Figure 7: Getting version using L1 Command 11](#_Toc64460306)

[Figure 8: ADXL Data on iOS Sample 14](#_Toc64460307)

[Figure 9: ECG Data on iOS Sample 15](#_Toc64460308)

**Copyright, Disclaimer & Trademark Statements**

**Copyright Information**

Copyright (c) 2021 Analog Devices, Inc. All Rights Reserved. Redistribution and use in source and binary forms, with or without modification, are permitted (subject to the limitations in the disclaimer below) provided that the following conditions aremet:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of Analog Devices, Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

**Disclaimer**

NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCHDAMAGE

**Trademark and Service Mark Notice**

Analog Devices, the Analog Devices logo, Blackfin, SHARC, TigerSHARC, CrossCore, VisualDSP, VisualDSP++, EZ-KIT Lite, EZ-Extender, SigmaStudio and Collaborative are the exclusive trademarks and/or registered trademarks of Analog Devices, Inc (“Analog Devices”).

All other brand and product names are trademarks or service marks of their respective owners.

Analog Devices’ Trademarks and Service Marks may not be used without the express written consent of Analog Devices, such consent only to be provided in a separate written agreement signed by Analog Devices. Subject to the foregoing, such Trademarks and Service Marks must be used according to Analog Devices’ Trademark Usage guidelines. Any licensee wishing to use Analog Devices’ Trademarks and Service Marks must obtain and follow these guidelines for the specific marks at issue.

# Introduction

This document is a getting started guide for the iOS SDK. It gives details on the package, features, and explains the sdk with an iOS sample code. The flow explained here will be similar for all platforms.

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

The SDK is distributed as a binary for different platforms. The standard library includes support for the following platforms:

* Windows (32 and 64 bit)
  + - CPP
* Linux
  + - CPP
  + iOS

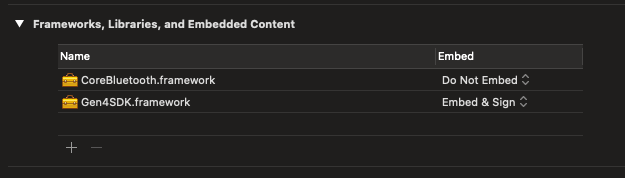
***Note:*** *For information regarding Python SDK, kindly visit our* [*Python SDK Release Page*](https://pypi.org/project/adi-study-watch/)

The following sections will refer to the iOS sample, located in *samples/iOS*.

**Note: To use this iOS Framework, xCODE is required**.

# Setting up the SDK.

To use the Gen4SDK.Framework, the framework needs to be linked to that project. Then import <Gen4SDK/Gen4SDK.h> to use the SDK APIs to communicate with the watch.



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.

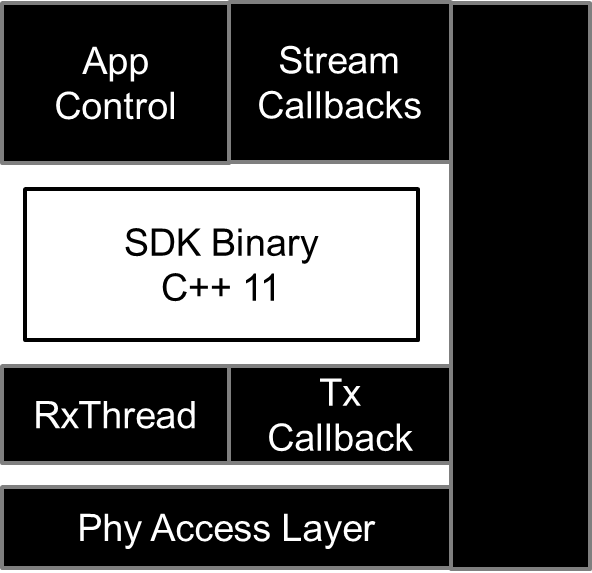


Figure 1: Typical App Arrangement

## BLE Communication

Before transmitting and receiving of data from sdk, the application should be first connected to the watch. The app uses the CoreBluetooth.Framework for iOS ble communication. The code snippet for ble interface and delegates can be found in [*samples\ios\gen4sdkBLE.m*](samples/ios/gen4sdkBLE.m)*.*

Once the watch is connected, the app should discover the services and characteristics provided by the watch. Then the app should set the notification of the read characteristic to listen to the response of the watch. Once the ble layer is completely implemented, the app can now use the sdk to communicate with the watch.

## Transmission (Tx) Callback

Transmission of data from sdk to ble layer is done via a callback function – which is called by the SDK when data is to be transmitted to the watch. Please note, only complete M2M2 packets should be sent to the watch – the SDK will only provide complete M2M2 packets to the transmissions callback. This function transmits over the physical layer.

User Application Calls an API in SDK

SDK populates the byte stream and sends it through the watch physical callback

User Application sends the byte stream to the device through physical layer(UART/BLE)

Figure 2: Transmission callback from iOS samples

**// File - gen4sdkWrapper.mm**

// SDK forms the packet and sends to this call-back. This call-back then sends the packet through physical layer (BLE)

class phy\_cb: public watch\_phy\_callback {

public:

void call(std::vector<uint8\_t> pkt) {

std::cout << "This is PHY, got something to write:" << std::endl;

NSMutableData \*packetdata = [[NSMutableData alloc]init];

for (int i = 0; i < pkt.size(); i++) {

[packetdata appendBytes:&pkt[i] length:1];

std::cout << " " << std::hex << std::setfill('0') << std::setw(2) << (int) pkt[i];

}

// Write the data through BLE from this callback

[wrapperswift CallSwiftFunction:packetdata];

};

void sys\_alert\_call(uint8\_t alert)

{

std::cout << "SDK Alert Message -" << alert << std::endl;

};

};

Note, that the SDK instance: g\_mw in introduced. The class *watch*,which is the SDK class, is initialized by assigning to the transmit callback function. The *watch* instance g\_mw will be used throughout the application.

**// File - gen4sdkWrapper.mm**

/\*\*

Initializing Callback to Write the data to BLE Layer

\*/

phy\_cb \*g\_phy\_callback = new phy\_cb();

/\*\*

Initializing Callback for Watch

@param g\_phy\_callback defines the callback to write the data to BLE

\*/

watch g\_mw = watch(\*g\_phy\_callback);

Figure 3: Assigning the Callback

## Receiving Callback

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

After decoding, it returns the value in respective API.

Stream values are sent to respective stream callbacks.

The byte stream then dispatches to the sdk for decoding

User Application receives the byte stream from device to the physical layer (BLE)

**// File - gen4sdkBLE.m**

//BLE receiver callback. When the data is received it will trigger this callback

-(void)peripheral:(CBPeripheral \*)peripheral didUpdateValueForCharacteristic:(CBCharacteristic \*)characteristic error:(NSError \*)error

{

NSData \*ReceivedPacket = characteristic.value;

// dispatch the received data to SDK.

[WRapperObjective Dispatchpacket:ReceivedPacket];

NSLog(@"Packet-DATA-%@", characteristic.value);

}

Figure 4: Receiving Callback of BLE

**// File - gen4sdkWrapper.mm**

/\*\*

Dispatching packet that we received from the watch to the Library

@param receivedbyte Receiving Packet from the BLE Layer

\*/

-(void)Dispatchpacket : (NSMutableData \*)receivedbyte

{

std::vector<uint8\_t> g\_received\_pkt;

Byte payloadadpdbuffer[receivedbyte.length];

[receivedbyte getBytes:&payloadadpdbuffer length:receivedbyte.length];

for (int i = 0; i < receivedbyte.length; i++) {

g\_received\_pkt.push\_back(payloadadpdbuffer[i]);

std::cout << " " << std::hex << std::setfill('0') << std::setw(2) << (int) g\_received\_pkt[i];

}

g\_mw.dispatch(g\_received\_pkt);

}

## Initialize SDK

Once the BLE Connection and Transmission callback is implemented, the SDK must be initialized with the respective platform. Platforms supported in this release include,

* python\_usb
* windows
* python\_ble
* iOS

**// File - gen4sdkWrapper.mm**

g\_mw.initialize\_sdk(g\_mw.ios);

Figure 5: Initialize SDK

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

# Getting Started With SDK

The SDK exports several classes that allow interaction with the watch. The Figure below shows the various applications that are available. See [***sdk/doc/doxygen/html/classwatch.html***](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 |

Figure 6: Application list

## L1 Commands

These represent Level 1, or L1 commands. 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
* Other
  + ped\_app: Pedometer application

## Basic SDK Operation of L1 Commands

Some basic operation with the SDK is explained.

The version of the software on the watch is read using the below method. For this please refer the function *PmApp* of the iOS sample. For more information on *get\_version*, see [sdk/doc/doxygen/html/classm2m2\_\_application.html](doc/doxygen/html/classm2m2__application.html) . The result of the version details will be stored in the pm\_app\_version structure.

common\_app\_version\_t pm\_app\_version;

g\_mw.pm\_app.get\_version(&pm\_app\_version);

Figure 7: Getting version using L1 Command

## L1 vs L2 commands

L1 commands provide primitive/fundamental access to the watch. Each L1 command is mapped to an equivalent M2M2 command. 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

g\_mw.adpd4000\_app.load\_cfg(adpd4000\_application::SENSOR\_ADPD4000\_DEVICE\_ID::ADPDCL\_DEVICE\_4000\_G);

g\_mw.adpd4000\_app.calibrate\_clock(6);

g\_mw.ppg\_app.load\_lcfg(ppg\_application::SENSOR\_PPG\_LCFG\_ID::PPG\_LCFG\_ID\_ADPD4000);

g\_mw.ppg\_app.ppg\_stream.subscribe(ppg\_stream\_callback);

g\_mw.ppg\_app.ppg\_stream.start();

Stop PPG via L1

g\_mw.ppg\_app.ppg\_stream.unsubscribe();

g\_mw.ppg\_app.ppg\_stream.stop();

The equivalent L2 commands are,

Start PPG via L2

g\_mw.start\_syncppg(ppg\_stream\_callback, syncppg\_stream\_callback);

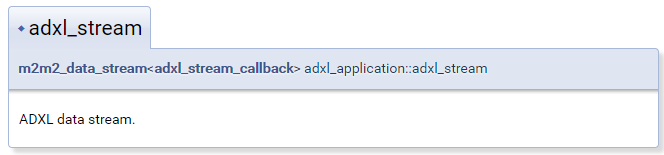
Stop PPG via L2

g\_mw.stop\_syncppg();

# Stream ADXL Data from the App using L1 Commands

A callback function is required to stream data from the watch. Each stream has an equivalent callback function. To understand streaming data from the watch, let us look at accelerometer, the ADXL362 based application.

Referencing [sdk /doc/doxygen/html/classadxl\_stream\_callback.html](doc/doxygen/html/classadxl__stream__callback.html), the callback is defined as:



Referencing the iOS sample, the ADXL callback is implemented as shown below.

class AdxlStreamCallback : public adxl\_stream\_callback {

public:

void call(std::vector<adxl\_stream\_cb\_data\_t> pkt, uint16\_t sequence\_num) {

std::cout << "Seqnum = " << sequence\_num << std::endl;

for (int i = 0; i < pkt.size(); i++)

{

adxl\_stream\_cb\_data\_t \*pAdxlItem = &pkt[i];

std::cout << "TS, x, y, z = "

<< pAdxlItem->timestamp << ", "

<< pAdxlItem->x << ", "

<< pAdxlItem->y << ", "

<< pAdxlItem->z

<< std::endl;

x\_value = pAdxlItem->x;

y\_value = pAdxlItem->y;

z\_value = pAdxlItem->z;

[[NSNotificationCenter defaultCenter]postNotificationName:@"UpdateADXL" object:nil];

}

};

};

The core of the streaming callback function is the *call* function. This function is common to all

streaming callback classes. In this case, a vector of type *adxl\_stream\_cb\_data\_t* is returned along with the sequence number of the incoming M2M2 packet. The sequence number allows the user to determine if any packets are missing. Each stream has a 16-bit sequence number, starting at a count of 0 and flipping back to 0 at the sequence number of 65,535. Please note that each call back contains a single piece of data. **Callback must not block to ensure smooth functioning of SDK.**

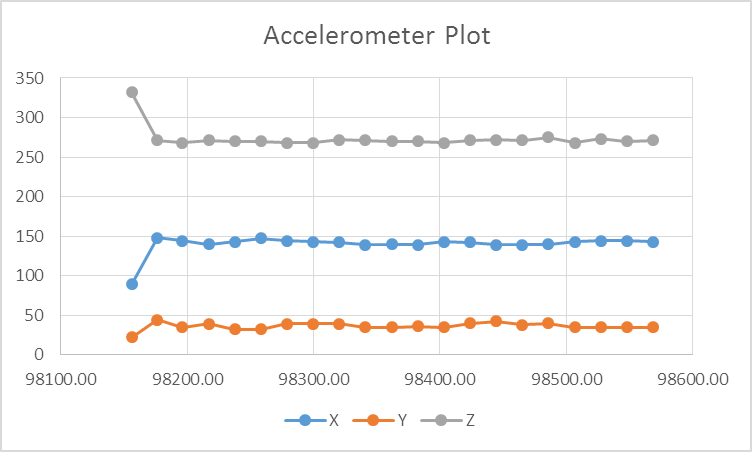
Let us take a look at *adxl\_stream\_cb\_data\_t*. Referencing [*sdk /doc/doxygen/html/structadxl\_\_stream\_\_cb\_\_data\_\_t.html*,](doc/doxygen/html/structadxl__stream__cb__data__t.html) we can see the data returned for streamed ADXL data.

|  |  |
| --- | --- |
| double | timestamp |
|  | System timestamp in milliseconds. |
|  | |
| int16\_t | x |
|  | X-axis accelerometer value. |
|  | |
| int16\_t | y |
|  | Y-axis accelerometer value. |
|  | |
| int16\_t | z |
|  | Z-axis accelerometer value. |

Here is an example of data streamed from the watch:

|  |  |  |  |
| --- | --- | --- | --- |
| **Timestamp (ms)** | **X** | **Y** | **Z** |
| 98156.30 | 89 | 22 | 332 |
| 98176.30 | 148 | 44 | 271 |
| 98196.30 | 144 | 35 | 268 |
| 98217.00 | 140 | 39 | 271 |
| 98237.70 | 143 | 32 | 270 |

Each data packet in a stream contains a timestamp. The timestamp, of type *double*, represents the watch time at which the sample was taken. The watch maintains a 32-bit timer with 32KHz accuracy. The X, Y and Z values are derived from the accelerometer.



In the iOS sample, adxl stream is started using L1 commands and its value is displayed on the dashboard.

Start ADXL via L1

g\_mw.adxl\_app.adxl\_stream.start();

g\_mw.adxl\_app.adxl\_stream.subscribe(adxl\_stream\_callback);

Stop ADXL via L1

g\_mw.adxl\_app.adxl\_stream.unsubscribe();

g\_mw.adxl\_app.adxl\_stream.stop();



Figure 8: ADXL Data on iOS Sample

# Stream ECG Data from app using L2 Commands

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.

The arguments of stream L2streamtest are explained below

Start ECG via L2

g\_mw.start\_ecg(ecg\_stream\_callback);

Stop ECG via L2

g\_mw.stop\_ecg();

In the iOS sample, ecg stream is started using L2 commands and its value is displayed on the dashboard.



Figure 9: ECG Data on iOS Sample