Document XD0202P, Revision E, 25 Mar 2022

Xsens DOT iOS SDK Documentation

Revision	Date	By	Changes
A	25 April 2020	XUF	Initial release
B	27 Aug 2020	XUF, JIAJ	Add SDK architecture Add suggested workflow Add sync class and example code Add recording class and example code Add details in real-time streaming example code Add heading reset methods and example code Add power saving example code
C	15 Dec 2020	XUF, JIAJ	Add an interface to check new firmware Add the workflow to start synchronization Add the workflow to start and stop real-time streaming Add the workflow for heading reset Add the workflow to start and stop recording Add the workflow to export recording data Add the new output rates for real-time streaming and recording Add new filter profile Add button callback function Support RSSI Add get sync status function Add stop sync function
D	16 July 2021	ERI, XUF, JIAJ	Update the diagrams with Xsens colour scheme Add the product ID Add the power options Add the free acceleration conversion Update the sync section Add firmware update via OTA Add Custom Mode 4 Remove the new firmware update notifications
E	25 Mar 2022	JIAJ	Add Mag field mapper Add Custom mode 5

Table of Contents

List of Tables

List of Figures

1 Introduction

The Xsens DOT iOS SDK is a software development kit for iOS applications on iPhones and iPads. iOS developers can use this SDK to build their applications to scan and connect the sensors, get data in real-time streaming or recording, as well as other functions.

This document mainly addresses SDK usage with example codes. Before getting started with the SDK, it is advised to read *Xsens DOT User Manual* first to understand the basic functions of the sensor.

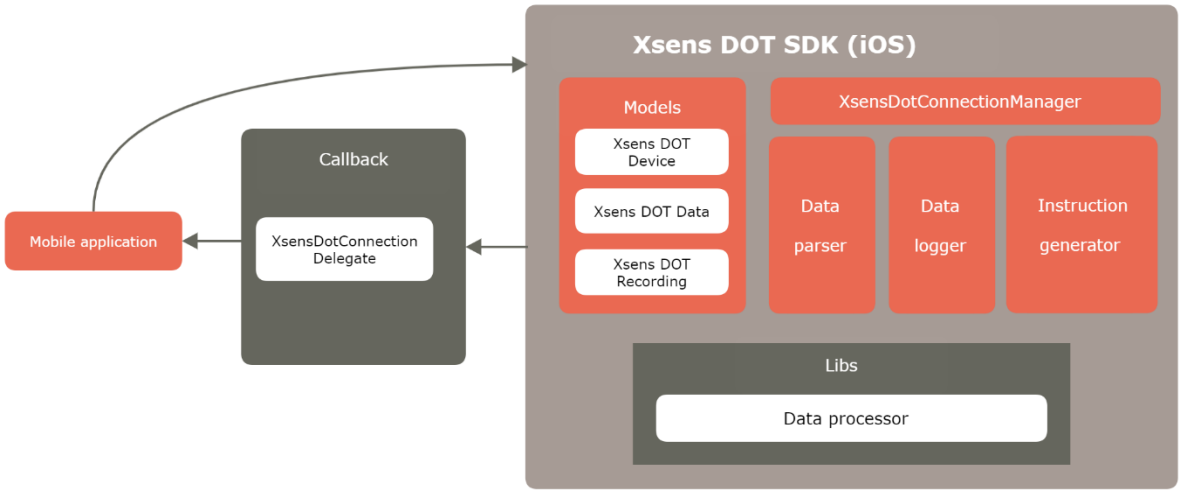

Figure 1: Xsens DOT Mobile SDK Architecture

The SDK provides classes for developers to facilitate easier integration into specific application. Figure 1 shows the SDK components and architecture. It contains 3 main models to manage the state of device, data payload types and data output. It also contains different classes[1] available for usage. The data processor library is integrated in SDK to process the data from firmware. Other libraries like sensor fusion and calibration libraries are running on Xsens DOT firmware.

[1] Not every class can be new or referenced

2 Getting Started with SDK

2.1 Platform Requirements

Table 1 shows the iOS version and Bluetooth requirements for the mobile devices.

Table 1: Software supported platforms

Platform requirements

• iOS 11.0 and above
• Bluetooth
  ○ Best performance with BLE 5.0, DLE[2] supported
  ○ Compatible with Bluetooth 4.2

2.2 Example code

Refer to this project on GitHub for the iOS example code of Xsens DOT SDK:

- https://github.com/xsens/xsens_dot_example_ios

2.3 SDK Changelogs

Refer to this BASE article for the iOS SDK changelogs.

2.4 Import SDK Framework

The required development tool is Xcode. The following steps describe how to import the SDK into your Xcode project.

1. Import the whole XsensDotSDK.framework package into the target project.
2. Make sure XsensDotSDK.framework is included in **`Target`** → **`General`** → **`Framework, Libraries, and Embedded Content`**, and the **`Embed property`** is **`'Embed & Sign'`**
3. Under **`Target`** → **`Build Settings`**, Enable Bitcode is **`NO`**
4. Add these to **`Info.plist`**
   - NSBluetoothAlwaysUsageDescription
   - NSBluetoothPeripheralUsageDescription

[2] Data Length Extension

3 Classes and methods

3.1 XsensDotConnectionManager

This class manages the BLE connection of Xsens DOT sensors.

Table 2: Methods in XsensDotConnectionManager

Method	Description
+ (void)scan	Scan the sensors
+ (void)stopScan	Stop scanning
+ (void)connect:(XsensDotDevice *)device	Connect sensors
+ (void)disconnect:(XsensDotDevice *)device	Disconnect sensors
+ (BOOL)managerStateIsPoweredOn	Check Bluetooth is enabled or not
+ (XsensDotManagerState)managerState	Get Bluetooth status
+ (void)setConnectionDelegate:(nullable id<XsensDotConnectionDelegate>)delegate	Set the connection delegate

3.2 XsensDotConnectionDelegate

The protocol of *XsensDotConnectionManager*. You can use these methods to get all the scanning and connection status.

Table 3: Methods in XsensDotConnectionDelegate

Method	Description
- (void)onManagerStateUpdate: (XsensDotManagerState)managerState	Return Xsens DOT BLE state
- (void)onScanCompleted	Return if the scanning is completed
- (void)onDiscoverDevice: (XsensDotDevice *_Nonnull)device	Return the discovered device
- (void)onDeviceConnectSucceeded: (XsensDotDevice *_Nonnull)device	Return if the connection is success
- (void)onDeviceConnectFailed: (XsensDotDevice *_Nonnull)device	Return if the connection fails
- (void)onDeviceDisconnected: (XsensDotDevice *_Nonnull)device	Return if the device is disconnected

3.3 XsensDotDevice

XsensDotDevice represents an Xsens DOT device object, including basic information and operations, data measurement and data logging.

3.3.1 Properties

Table 4: Properties of XsensDotDevice

Property	Description
uuid	Bluetooth device UUID
macAddress	Device mac address
RSSI	Bluetooth signal indication
Battery	Device battery object
firmwareVersion	Device firmware version
plotMeasureEnable	Measurement state of real-time streaming
plotMeasureMode	Measurement modes of real-time streaming
plotLogEnable	Data logging state

isSupportHeadingReset	A flag to know if this firmware supports heading reset function.
headingStatus	Heading reset status
^headingResetResult	Heading reset result block
timeoutXMinutes	Power saving time in advertisement mode – minute. Valid value: 0~30
timeoutXSeconds	Power saving time in advertisement mode – second. Valid value: 0~60
timeoutYMinutes	Power saving time in connection mode – minute. Valid value: 0~30
timeoutYSeconds	Power saving time in connection mode – minute. Valid value: 0~60
totalSpace	The total available internal storage space for recording. Unit is byte.
usedSpace	The used internal storage space of recording. Unit is byte.
recording	The recording object.
exportDataFormat	To set the export data format of recording files.
exportLogEnable	Set to enable/disable the file logging function when exporting recording data.
outputRate	Output rate. The unit is Hz: 1, 4, 10, 12, 15, 20, 30, 60 and 120 (120 Hz only for recording mode)
filterIndex	The filter profile index 0: General filter profile 1: Dynamic filter profile

3.3.2 Methods

Table 5: Methods in XsensDotDevice

Method	Description
- (void)setDidParsePlotDataBlock:(void (^ _Nullable)(XsensDotPlotData * _Nonnull plotData))block	Block of data measurement
- (void)setDeviceName:(NSString *)name	Set device tag name
- (void)startIdentifying	Identify sensor
- (void)powerOff	Power off sensor
- (BOOL)startHeadingReset	Start heading reset. "NO" means this feature is not supported with current the firmware.
- (BOOL)startHeadingRevert	Start heading revert. "NO" means this feature is not supported with current the firmware.
- (BOOL)getRecordingStatus	Get sensor's recording state. "NO" means this feature is not supported with current the firmware.
- (BOOL)startRecording:(UInt16) recordingTime	Start recording. "NO" means this feature is not supported with current the firmware.
- (BOOL)stopRecording	Stop recording. "NO" means this feature is not supported with current the firmware.
- (BOOL)getFlashInfo	Get information of recording flash. "NO" means this feature is not supported with current the firmware.
- (void)setFlashInfoDoneBlock:(void (^_Nullable)(XSFlashInfoStatus status))block	Get flash information done block
- (BOOL)getRecordingTime	Get recording time. "NO" means this feature is not supported with current the firmware.
- (BOOL)eraseData;	Request to erase all the recording data space. "NO" means this feature is not supported with current the firmware.
- (void)setEraseDataDoneBlock:(void (^_Nullable)(int success))block	Erase data done block
- (BOOL)getExportFileInfo	Get information of the export recording file. "NO" means this feature is not supported with current the firmware.

- (void)setExportFileInfoDone:(void (^_Nullable)(BOOL success))block	Export file information done block
- (BOOL)startExportFileData	Start export file data. "NO" means this feature is not supported with current the firmware.
- (BOOL)stopExportFileData	Stop export file data. "NO" means this feature is not supported with current the firmware.
- (void)setDidParseExportFileDataBlock:(void (^ _Nullable)(XsensDotPlotData * _Nonnull plotData))block	Data block of exported data
- (BOOL)isInitialized	The flag that sensor has been initialized after connection.
- (BOOL)isSynced	The flag of sensor synced.
- (void)setOutputRate:(int)outputRate filterIndex:(int)filterIndex	Set the output rate and filter index.
- (void)readRSSI:(void (^_Nullable)(NSNumber *signal))block	Read the RSSI of sensor
- (void)setDidButtonCallbackBlock:(void (^ _Nullable)(int timestamp))block	The button callback block.
- (BOOL)isProductV2	"YES" means the hardware is Xsens DOT v2.
- (BOOL)isProductV1	"YES" means the hardware is Xsens DOT v1.
- (BOOL)isUsbPowerOnEnabled	"YES" means the sensor will power on with USB plugin. This function is only available in v2.
- (BOOL)enableUsbPowerOn:(BOOL) isEnable	Enable the sensor to power on by USB plugin. This function is only available in v2.

3.4 XsensDotPlotData

XsensDotData contains all the measurement data. When set *setPlotMeasureEnable* to YES, the block will get the data from *setDidParsePlotDataBlock*.

Refer to *XsensDotPlotData.h* for more information.

3.5 XsensDotDefine

XsensDotDefine is a common define of SDK. It has notifications and other defines. Please import it in your Xcode project.

Table 6: Notifications of XsensDotDefine

Notification	Description
kXsensDotNotificationManagerStateDidUpdate	Bluetooth state update
kXsensDotNotificationDeviceConnectSucceeded	Sensor connection success notification
kXsensDotNotificationDeviceConnectFailed	Sensor connection failure notification
kXsensDotNotificationDeviceDidDisconnect	Sensor disconnection notification
kXsensDotNotificationDeviceBatteryDidUpdate	Battery level update
kXsensDotNotificationDeviceFirmwareVersionDidRead	Firmware version read success
kXsensDotNotificationDeviceNameDidRead	Tag name read success
kXsensDotNotificationDeviceMacAddressDidRead	Mac address read success
kXsensDotNotificationDeviceConnectionDidStart	Sensor starts connecting notification

Refer to *XsensDotDefine.h* for more information.

3.6 XsensDotUtils

The utils class contains the available conversion methods.

Table 7: Methods in XsensDotUtils

Method	Description
+ (void)quatToEuler:(double [_Nullable])eular WithW:(float)quatW withX:(float)quatX withY:(float)quatY withZ:(float)quatZ	Convert quaternion to Euler angles

3.7 XsensDotLog

Enable *XsensDotLog* when you want to get debug information. Note that you need to disable it in released apps.

Table 8: Methods in XsensDotLog

Method	Description
+ (void)setLogEnable:(BOOL)enable	Enable/disable debug mode
+ (BOOL)isLogEnable	Get debug mode state

3.8 XsensDotReconnectManager

When *setEnable* is set to **YES**, the sensor will automatically reconnect every second if the connection is lost.

Table 9: Methods in XsensDotReconnectManager

Method	Description
+ (void)setEnable:(BOOL)enable	Enable/disable reconnection
+ (BOOL)enable	Get current reconnection state

3.9 XsensDotSyncManager

This class is the synchronization manager. All sensors will be time-synced with each other to a common time base after synchronization. Refer to section 3.3.4 in *Xsens DOT User Manual* for more information. The root node in the iOS SDK is always the first sensor connected. It will take the sensors about 10 seconds to finish the sync period, so you can reconnect the sensors after that.

Table 10: Methods in XsensDotSyncManager

Method	Description
+ (void)startSync:(NSArray<XsensDotDevice *> *)devices result:(XsensDotSyncResultBolck) resultBlock	Start synchronization
+ (void)stopSync:(NSArray<XsensDotDevice *> *)devices	Stop synchronization

3.10 XsensDotRecording

This class has all the properties of recording, including flashInfoStatus, recordingStatus, recordingdata, recordingTime etc. *XsensDotDevice* has a property of recording, you can control it after *XsensDotDevice* connected.

Table 11: Properties in XsensDotRecording

Property	Description
flashInfoStatus	The status of flash information, refer to table?
recordingStatus	The status of recording, refer to table?
^updateRecordingStatus	Recording status update block
^updateExportingStatus	Export file data status update block
recordingDate	Recording start time UTC, the unit is second
recordingTime	The total recording time of a timed recording. For normal recording, the value is 0xFFFF.
remainingTime	The remaining time of a timed recording.
files	The recording file list
exportFileList	Export file list
^exportFileDone	Export file done block

3.11 XsensDotDevicePool

This class manages the reconnection behavior of sensors. When reconnection is enabled, you need to bind the sensor to activate the reconnection function after connecting a sensor. Unbind it after disconnecting the sensor, otherwise a reconnection will be initialized.

Table 12: Methods in XsensDotDevicePool

Method	Description
+ (BOOL)bindDevice:(XsensDotDevice *)device	Bind a sensor
+ (void)unbindDevice:(XsensDotDevice *)device	Unbind a sensor
+ (NSArray <XsensDotDevice *>*)allBoundDevices	All the bound devices

4 SDK Usage with Examples

4.1 Recommended workflow

The iOS SDK code flow is shown in Figure 2. This flow process can be used by iOS developers after importing SDK into iOS project.

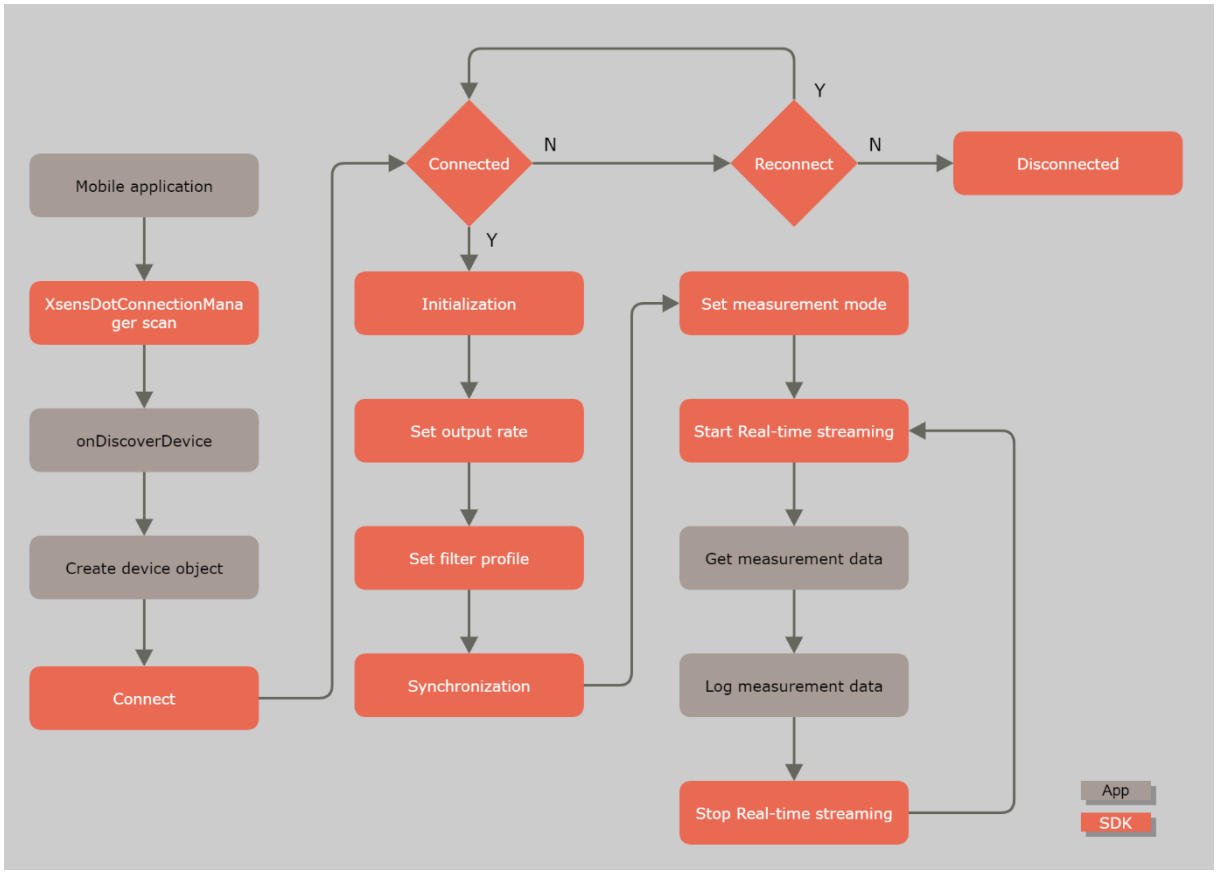

Figure 2: Xsens DOT iOS SDK Workflow

Begin with starting BLE scan with *XsensDotConnectionManager scan*. Developers can get the *XsensDotDevice* object using *onDiscoverDevice* method in *XsensDotConnectionDelegate*. *XsensDotDevice* object manages the all the behaviors of the sensor.

Use *XsensDotConnectionManager connect:device* to connect the sensors. The connection status is updated by *XsensDotConnectionDelegate*. If the connection process fails, the SDK will check if whether the reconnection feature is enabled. Reconnection will start automatically if enabled.

Each step is further explained in the following sections with example code.

4.2 Debugging Flag

This is a static function and can be used to enable/disable the debug messages. If it is set to true, the SDK will output debug message with this tag – 'XsensDotSDK'.

```
[XsensDotLog setLogEnable:YES];
```

4.3 Reconnection Setting

Follow these steps to configure the reconnection setting of Xsens DOT. If set to true and bound, the SDK will start to reconnect the sensor(s) automatically when the connection is lost.

4.3.1 Enable Reconnect Manager

Enable the reconnect manager in viewController:

```
[XsensDotReconnectManager setEnable:YES];
```

4.3.2 Bind the sensor

Add *bindDevice* after connecting and *unbindDevice* after disconnecting.

```
if(device.state != CBPeripheralStateConnected)
{
    [XsensDotConnectionManager connect:device];
    [XsensDotDevicePool bindDevice:device];
}
else
{
    [XsensDotConnectionManager disconnect:device];
    [XsensDotDevicePool unbindDevice:device];
}
```

4.4 BLE Scan

Before starting the scanning, you must ensure that Bluetooth is on and the iPhone is powered on and available.

```
if(![XsensDotConnectionManager managerStateIsPoweredOn])
{
    NSLog(@"Please enable bluetoooth first");
    return;
}
[XsensDotConnectionManager scan];

Set the delegate:
- (void)viewWillAppear:(BOOL)animated
{
    [super viewWillAppear:animated];
    [XsensDotConnectionManager setConnectionDelegate:self];
    xxx
}
```

The *XsensDotConnectionDelegate* has all status of scanning result.

4.5 Connect

Declare an *XsensDotDevice* object or *XsensDotDevice* array.

```
@property (strong, nonatomic) NSMutableArray *deviceList;
```

In *onDiscoverDevice:(XsensDotDevice *)device* of *XsensDotConnectionDelegate*, add all devices to *deviceList*.

```
- (void)onDiscoverDevice:(XsensDotDevice *)device
{
    NSInteger index = [self.deviceList indexOfObject:device];
    if(index == NSNotFound)
    {
        if(![self.deviceList containsObject:device])
        {
            [self.deviceList addObject:device];
        }
    }
}
```

Use *XsensDotConnectionManager* to connect one or multiple sensors.

```
XsensDotDevice *xsensDotDevice = self.deviceList[indexPath.row];
XsensDotConnectionManager connect:XsensDotDevice];
```

Refer to *XsensDotDevice.h* for more information.

4.6 Initialization

After the sensor connection, an initialization process will start automatically to enable BLE notifications and obtain basic sensor information, including the hardware and firmware version, MAC address, tag name, battery status, synchronization status, filter profile, output rate etc.

isInitialized method will tell user whether a sensor has been initialized after connection. Make sure *isInitialized* is YES before proceeding further, such as synchronization and measurement.

4.7 Filter profile

After the initialization is done, you can get or set the current filter profile for the measurement. Refer to section 3.2 in the User Manual for more information about filter profiles.

Get current filter profile that is applied in the measurement:

```
xsensDotDevice.filterIndex
```

The *filterIndex* is the index of the filter profiles.

Table 13: Filter profile index

Index	Filter profile	Description
0	General	This filter profile is the default setting. It assumes moderate dynamics and a homogeneous magnetic field. External magnetic distortion is considered relatively short.

1	Dynamic	This filter profile assumes fast and jerky motions that last for a short time. The dynamic filter uses the magnetometer for stabilization of the heading and assumes very short magnetic distortions. Typical applications are when sensors are applied on persons for sports such as sprinting.

Set a new filter profile:

```
- (void)setOutputRate:(int)outputRate filterIndex:(int)filterIndex;
```

4.8 Output rate

After the initialization is done, you can get or set the output rate for the measurement. Table 14 shows the available output rates during measurements.

Table 14: Output rates

Measurement	Output rates
Real-time streaming	1 Hz, 4 Hz, 10 Hz, 12 Hz, 15 Hz,20 Hz, 30 Hz, 60 Hz
Recording	1 Hz, 4 Hz, 10 Hz, 12 Hz, 15 Hz,20 Hz, 30 Hz, 60 Hz, 120 Hz

Get current output rate that is applied in the measurement:

```
xsensDotDevice.outputRate
```

Set a new output rate for the measurement:

```
- (void)setOutputRate:(int)outputRate filterIndex:(int)filterIndex;
```

4.9 Synchronization

All sensors will be time-synced with each other to a common time base after synchronization. Refer to section 3.3.2 in *Xsens DOT User Manual* for more information.

Set the output rate and filter profile before starting the synchronization. Since the sensor will enter measurement mode right after the sync succeeds so it's not possible to change it after sync. Figure 3 below shows the recommended workflow for synchronization.

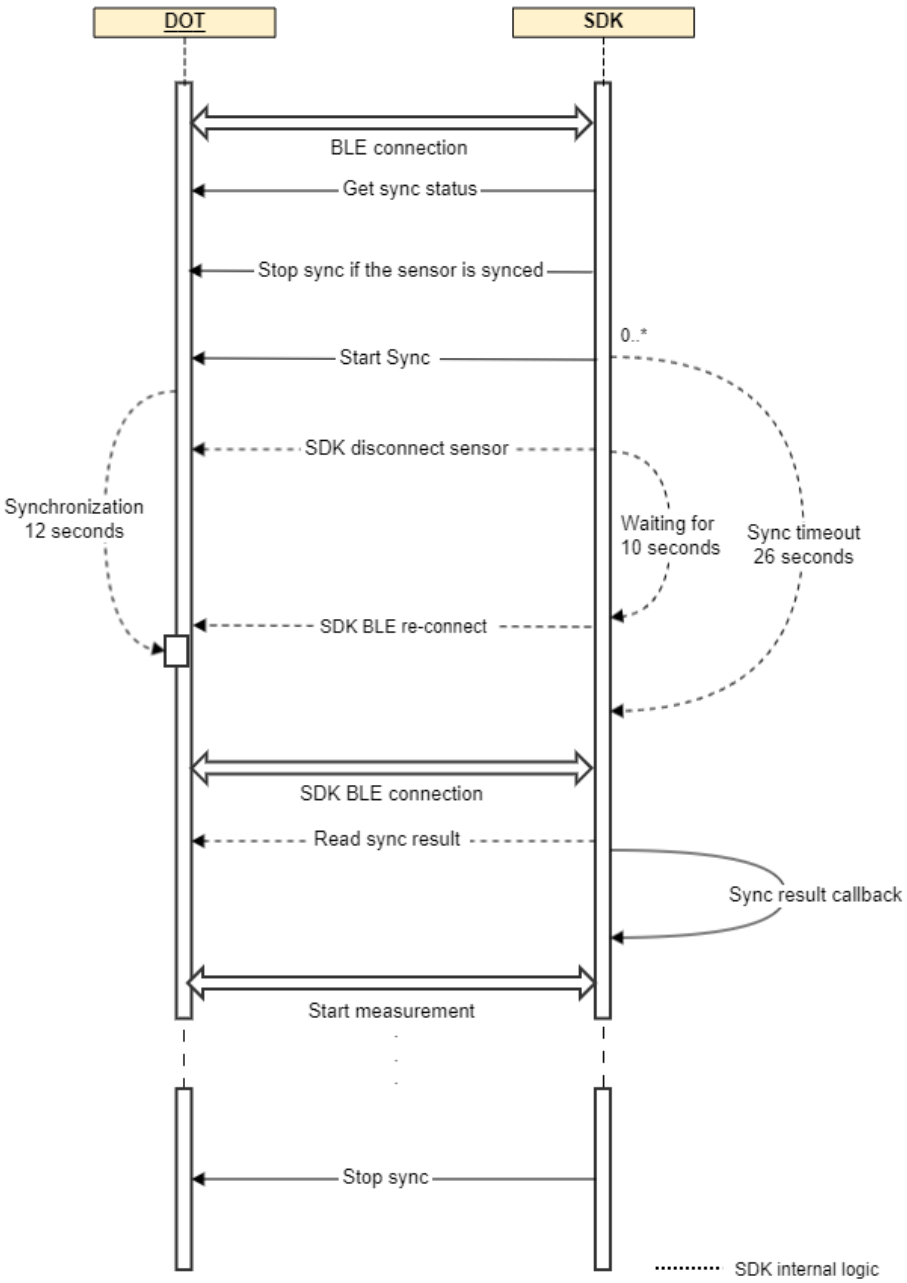

Figure 3: Synchronization workflow

4.9.1 Get sync status

Before starting the synchronization, check the synchronization status of the target sensors and make sure they are not synced. Stop the synced sensor before starting a new synchronization to prevent error status.

Get sync status with *isSynced* method.

4.9.2 Start sync

Use *startSync* method in *XsensDotSyncManager* to start the synchronization. The first param of *startSync* is a *XsensDotDevice* List and second param is a callback block. The first object in the List will be the root sensor while others will be the scanners.

SDK will disconnect the sensor after starting the synchronization. The synchronization of 5 sensors would take about 12 seconds while SDK will try to reconnect after 10 seconds.

4.9.3 Get sync results

After a successful reconnection, SDK will check the sync result to see if the sync is successful or not. The block param is an array object that includes the xsensDotDevice macAddress and sync result.

```
#import <XsensDotSdk/XsensDotSyncManager.h>
…

XsensDotSyncResultBolck block = ^(NSArray *array)
{

};
[XsensDotSyncManager startSync:self.xsensDotDeviceList result:block];
```

Once the sync succeeds, sensor will enter measurement mode. You can then choose to do real-time streaming or recording with the synced sensors.

If the sensor is not reconnected within 26 seconds, the sync is considered as failed. The sync is also failed if the result shows fail. In that case, SDK will stop those sensors that have been successfully synced. Refer to this BASE article for more tips about synchronization.

4.9.4 Stop sync

Stop sync is required after the measurement. Otherwise the sensor will stay in measurement mode and the battery will run out soon.

Use stopSync method in *XsensDotSyncManager* to stop synchronization. The param is the synced XsensDotDevice list.

```
[XsensDotSyncManager stopSync:boundDevices];
```

4.10 Real-time streaming

In real-time streaming, motion data is streamed and logged to the central device via a constant Bluetooth connection. You can set measurement mode, start/stop measurement and log the data to csv files with the SDK.

Figure 4 shows the workflow to start and stop real-time streaming.

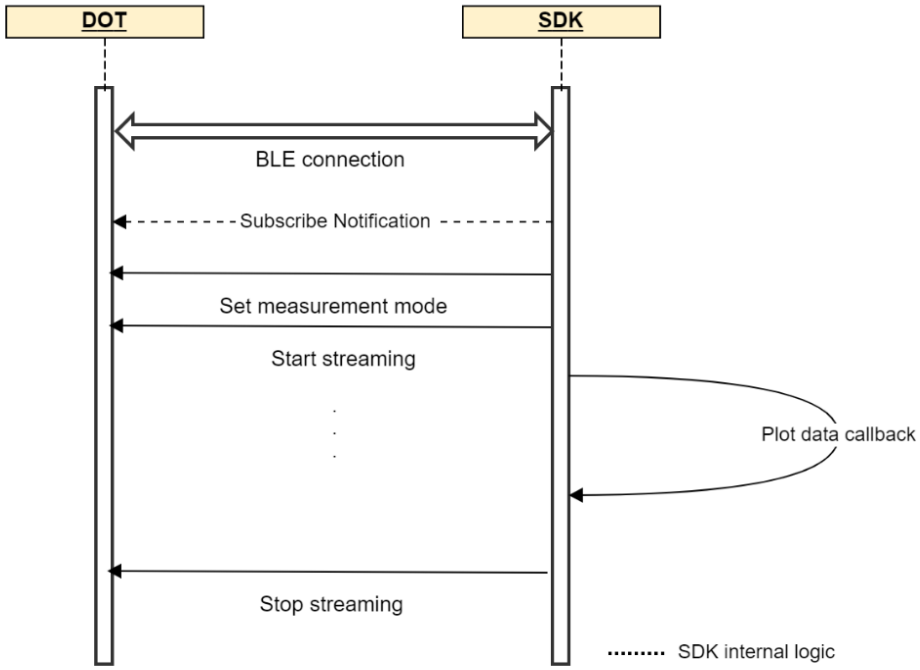

Figure 4: Workflow to start and stop real-time streaming

XsensDotDevice can report sensor data in real-time streaming and the callback is *setDidParsePlotDataBlock* block. To use this, notify the sensor to enter the measurement mode, then start the measurement by following the steps below.

4.10.1 Set measurement mode

You can get all the measurement modes of real-time streaming in this enum: *XSBleDevicePayloadMode*. 17 measurement modes are available for now. Refer to the Appendix for data outputs of different modes. Section 4.2 in *Xsens DOT User Manual* also gives detailed explanation about output values.

```
[xsensDotDevce setPlotMeasureMode:
XSBleDevicePayloadInertialHighFidelityWithMag];
```

4.10.2 Start measurement and set data block

```
[xsensDotDevce setPlotMeasureEnable:YES];

[_xsensDotDevice setDidParsePlotDataBlock:^(XsensDotPlotData *
_Nonnull plotData) {
```

```
double acc0 = plotData.acc0;
double acc1 = plotData.acc1;
double acc2 = plotData.acc2;
      …
        }];
```

Then you can get the data output according to different measurement modes.

4.10.3 Stop measurement

```
[xsensDotDevce setPlotMeasureEnable:NO];
```

Refer to *XsensDotPlotData.h* for more information.

4.10.4 Data Logging

Call *setPlotLogEnable* method in *XsensDotDevice* to enable or disable the data logging during real-time streaming.

```
[xsensDotDevce setPlotLogEnable:YES]
```

The logging data is saved in 'Logs' folder under *NSDocumentDirectory* as csv files.

4.10.5 High fidelity modes

In high fidelity mode, higher frequency (800 Hz) information is preserved with lower output data rate (60 Hz), even with transient data loss. There are 3 measurement modes containing high fidelity inertial data in the SDK:

- XSBleDevicePayloadInertialHighFidelityWithMag
- XSBleDevicePayloadHighFidelityNoMag
- XSBleDevicePayloadCustomMode4

To parse the high fidelity inertial data to to delta_q, delta_v or calibrated angular velocity and acceleration, you need to select the above measurement modes with high fidelity inertial data. After starting the measurement, you can get the values with read the properties of acc0-acc2, gyr0-gyr2, dQ0-dQ3, dV0-dV2 from *XsensDotData* object.

4.10.6 Data conversions

Data conversion functions are provided in Xsens DOT SDK. Developers can make use of these conversion functions to get the measurement quantities as required in their applications.

Convert quaternion to Euler angles

quatToEuler method is provided in XsensDotUtils class to convert quaternion values to Euler angles.

```
XsensDotPlotData *plotData;
[xsensDotDevice setDidParsePlotDataBlock:^(XsensDotPlotData * _Nonnull
plotData) {
        plotData = plotData;
}];
```

```
double eular[3];

[XsensDotUtils quatToEuler:eular WithW:plotData.quatW
withX:plotData.quatX withY:plotData.quatY withZ:plotData.quatZ];
```

Calculation of free acceleration

You can get the free acceleration from orientation and acceleration as mentioned in this BASE article.

In real-time streaming, *getCalFreeAcc* function is provided to help you omit the mathematical calculations.

As this function requires both orientation (in quaternion) and acceleration as input it can currently only be used with XSBleDevicePayloadCustomMode4. Custom mode 4 is the only mode that can output these two quantities at the same time.

```
Double *calFreeAcc = [plotData getCalFreeAcc];
//calFreeAcc[0] is free acceleration along the x-axis
//calFreeAcc[1] is free acceleration along the y-axis
//calFreeAcc[1] is free acceleration along the z-axis
```

The default gravity is 9.8127 m/s^2. You can set a custom gravity vector (for example 9.82 m/s^2) by defining its value in the following way:

```
double localGravity=9.82
[double *calFreeAcc = [plotData getCalFreeAcc:localGravity]
```

4.11 Heading reset

Heading reset function allows user to align heading outputs among all sensors and with the object they are connected to. Performing a heading reset will determine the orientation and free acceleration data with respect to a different earth-fixed local frame (L'), which defines the L' frame by setting the X-axis of L' frame while maintaining the Z-axis along the vertical. It computes L' such that Yaw becomes 0 deg.

The heading reset function must be executed during real-time streaming and with measurement mode including orientation output. The reset orientation is maintained between measurement start/stop and connection/disconnection but will be lost after a device reboot.

Figure 5 shows the workflow to do the heading reset.

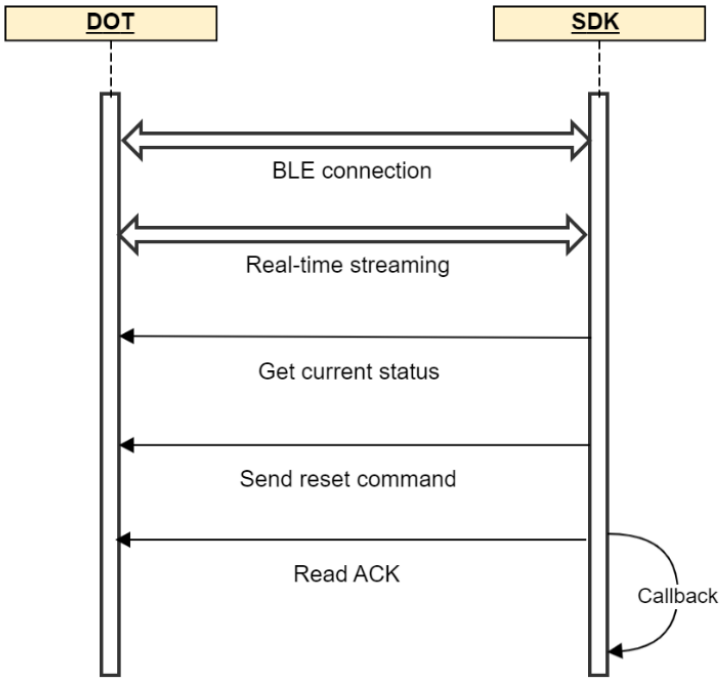

Figure 5: Workflow for heading reset

Follow the steps below to implement the heading reset function.

4.11.1 Check if heading reset is supported

Check if current firmware of the sensor support heading reset feature or not.

```
xsensDotDevice.isSupportHeadingReset
```

4.11.2 Heading reset status

Get the heading reset status of the sensor.

```
xsensDotDevice.headingStatus
```

Table 15: Heading status

Heading reset status	Value	Description
XSHeadingStatusXrmHeading	1	Heading has been reset
XSHeadingStatusXrmDefaultAlignment	7	Heading has been reverted to default status
XSHeadingStatusXrmNone	8	Default status

4.11.3 Reset heading

Align the sensor heading to 0 degree. Only reset the sensor which is reverted or in default status (status = 7 or 8).

```
[xsensDotDevice startHeadingReset]
```

4.11.4 Revert heading

Revert heading to the default value. Note that only revert the sensor which is reset (status = 1).

```
[xsensDotDevice startHeadingReset]
```

After reset the heading, a revert is required before conducting a new reset.

4.12 Recording

In recording mode, motion data is stored in the sensor internal storage and can be exported for post processing. Bluetooth connection is not required once the recording is started. With the SDK, you can start/stop recording, set timed recording and export recording data.

Figure 6 shows the recommended workflow to start and stop recording with iOS SDK.

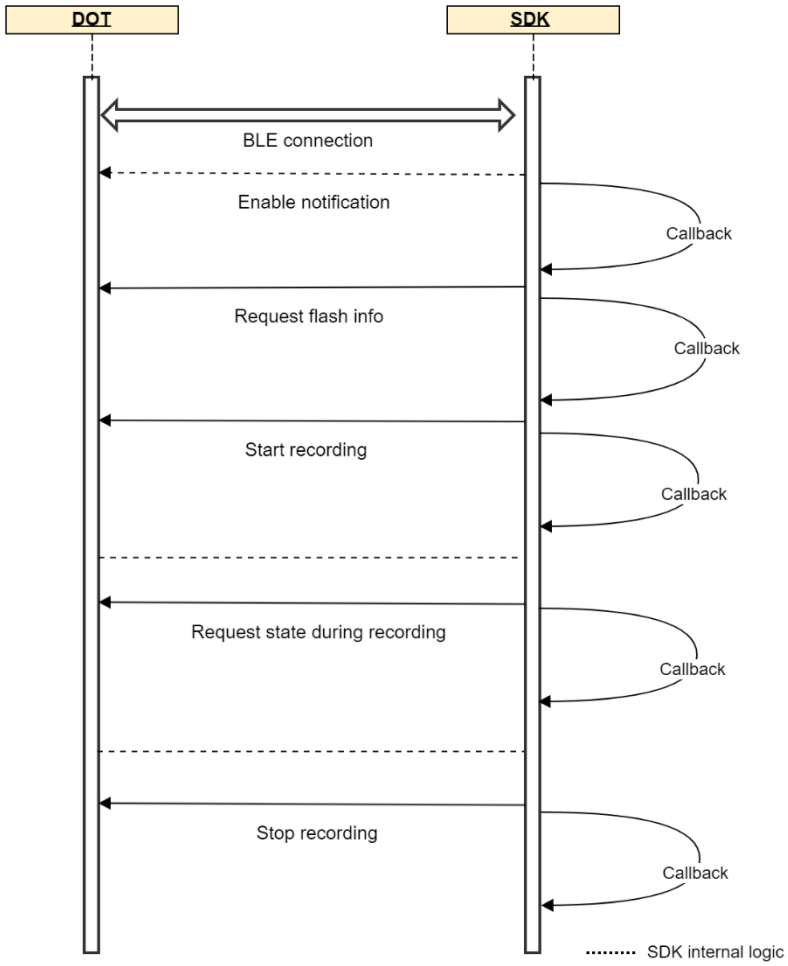

Figure 6: Workflow to start and stop recording

4.12.1 Get flash information

Flash information refers to recording flash size and its usage. The flash size that can be used for recording accounts for about 90% of the total size (16 MB for v1 sensor, 64MB for v2 sensor). So firstly, we need to get the available flash space and the remaining recording time before start recording.

xsensDotDevice.totalSpace and *xsensDotDevice.usedSpce* can only be initialized after calling *getFlashInfo*.

```
[xsensDotDevice getFlashInfo];
```

This can be used to update the flash status:

```
[xsensDotDevice setFlashInfoDoneBlock:^(XSFlashInfoStatus status) {
}];
```

Table 16: Flash status

Flash status	Description
XSFlashInfoIsReady	Recording flash space is ready for recording.
XSFlashInfoIsFull	Recording flash space is full (⩾ 90% the whole size).
XSFlashInfoIsUninitialized	Recording flash space is uninitialized or invalid.

Note that uninitialized or invalid flash means that the current recording flash space is not compatible with the flash structure supported in this firmware. Erase the flash space to reset the structure.

4.12.2 Start recording

You can start recording or set a timed recording with the parameter of *startRecording*.

Set the param to 0xFFFF to start recording; the recording will continue unless a stop recording command is received or sensor stops automatically. Set the param to other values to start a timed recording. The unit is second. Note that the maximum recording time of the sensor is 88 minutes.

```
[xsensDotDevice startRecording:0xFFFF];
```

4.12.3 Get recording status

Use *getRecordingStatus* to check the status of recording.

```
[xsensDotDevice getRecordingStatus];
```

After call this method, the *xsensDotDevice.recording.recordingStatus* will be updated with the recording status.

Table 17: Recording status

Recording status	Description
XSRecordingIsIdle	Idle status
XSRecordingIsRecording	Sensor is recording
XSRecordingIsRecordingStopped	Recording is stopped
XSRecordingIsErasing	Erasing recording data
XSRecordingIsFlashInfo	Sensor is getting flash information

In the meantime, *updateRecordingStatus* block will be updated automatically.

```
[xsensDotDevice.recording setUpdateRecordingStatus:^(XSRecordingStatus
status) {

}];
```

This block will also be updated after calling *startRecording* and *stopRecording*.

4.12.4 Stop recording

Use *stopRecording* to stop a normal recording or a timed recording.

```
[xsensDotDevice stopRecording];
```

Recording will also stop automatically in the following situations:

- power button is pressed over 1 second.
- time is up for timed recording.
- flash memory is over 90%.

4.12.5 Get recording time

If the *recordingStatus* is *XSRecordingIsRecording*, you can call *getRecordingTime* method to get *recordingDate*, *recordingTime* and *remainingTime*.

```
[xsensDotDevice getRecordingTime];
```

```
xsensDotDevice.recording.recordingDate
// start recording time 4 bytes unit is second.
xsensDotDevice.recording.recordingTime
// recording Time that you startRecording set 2 bytes unit is second.
xsensDotDevice.recording.remainingTime
// the remaining time after you start recording if you set
recordingTime is not 0XFFFF;
```

4.12.6 Erase flash

Erase all the recording data space, other flash space will not be affected.

```
[xsensDotDevice eraseData];
```

setEraseDataDoneBlock will be updated if the erase process is done.

```
[xsensDotDevice setEraseDataDoneBlock:^(int success) {}];
```

4.13 Recording data export

A stand-alone application – Xsens DOT Data Exporter is provided to export the recording data to PC via USB cable. You can download Windows or MacOS version in developers page.

Figure 7 shows the recommended workflow to start and stop recording with SDK.

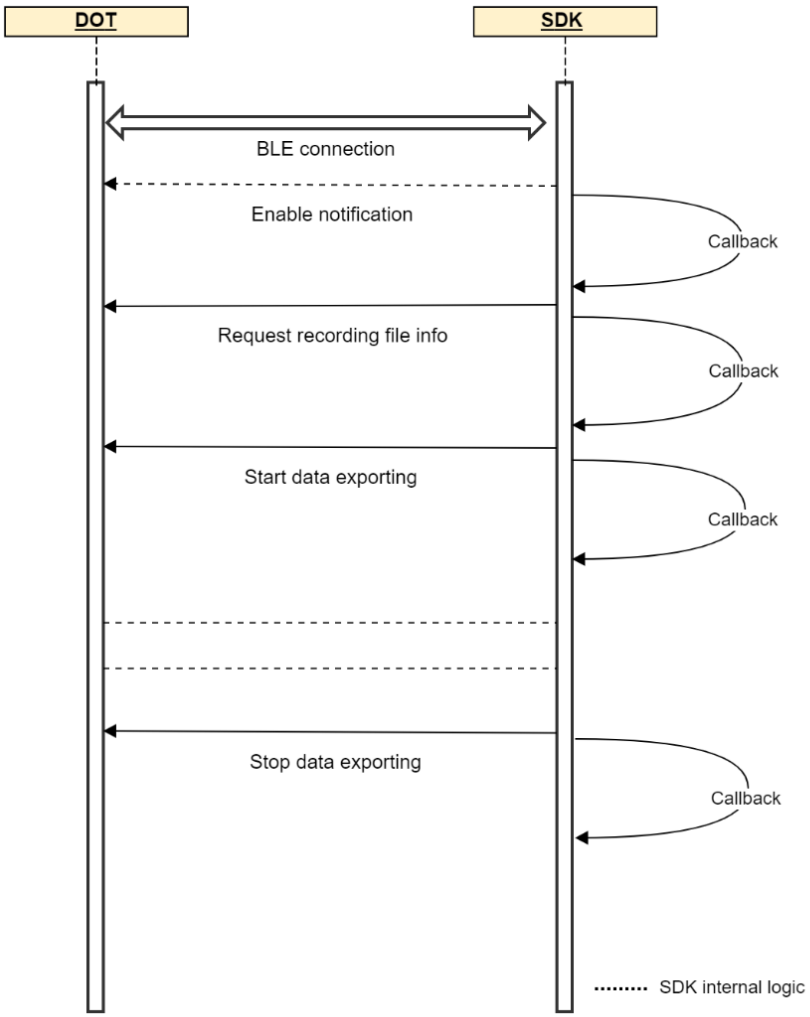

Figure 7: Workflow to export recording data

Follow the steps below to implement data export function in your application.

4.13.1 Set export data format

Refer to Table 18 for available data quantities when configuring the export data format. For physical meanings and other information of the data, please refer to chapter 4 in *Xsens DOT User Manual*.

Note that free acceleration is not provided in this firmware. Refer to this BASE article to calculate free acceleration from orientation (quaternion) and acceleration.

Table 18: Export data quantities

Data	Description
XSRecordingDataTimestamp	TimeStamp
XSRecordingDataQuaternion	Quaternion
XSRecordingDataEulerAngles	Euler angles
XSRecordingDataDq	dq
XSRecordingDataDv	dv
XSRecordingDataAcceleration	Calibrated acceleration
XSRecordingDataAngularVelocity	Calibrated angular velocity
XSRecordingDataMagneticField	Calibrated magnetic field
XSRecordingDataStatus	Status
XSRecordingDataClipCountAcc	clipCountAcc
XSRecordingDataClipCountGyr	clipCountGyro

If *exportDataFormat* is not set, the default value is:

- XSRecordingDataTimestamp
- XSRecordingDataEulerAngles
- XSRecordingDataAcceleration
- XSRecordingDataAngularVelocity

```
UInt8 bytes[4] = { XSRecordingDataTimestamp, XSRecordingDataQuaternion
, XSRecordingDataDq , XSRecordingDataDv };
NSData *exportData = [NSData dataWithBytes:defaultBytes
length:sizeof(bytes)];
xsensDotDevice.exportDataFormat = exportData;
```

4.13.2 Get export file information

You can use the *getExportFileInfo* method to get the start time UTC of the recording files. *xsensDotDevice.recording.files* will be initialized only after calling *xsensDotDevice getFlashInfo*. Since part of the file information is written in flash information.

```
[xsensDotDevice getExportFileInfo];
[xsensDotDevice setExportFileInfoDone:^(BOOL success) { }];
```

4.13.3 Set export file list

All the available recording files will be saved in *recording.files* array. Export file list is a *xsensDotDevice.recording.files* index array. You must set export file list via recording.exportFileList before start export file data, otherwise no files will be exported. The max index cannot exceed *xsensDotDevice.recording.files.count*.

```
NSArray *array = [[NSArray alloc] initWithObjects:@0, @1, @2, nil];
xsensDotDevice.recording.exportFileList = array;
or
NSArray *array = [[NSArray alloc] initWithObjects:@2, nil];
xsensDotDevice.recording.exportFileList = array;
```

4.13.4 Start export

Before starting to export thedata file, make sure you have set the export data format and get the export file information.

```
[xsensDotDevice startExportFileData];
```

Call *ExportFileDone* block after the data export is finished:

```
[xsensDotDevice.recording setExportFileDone:^(NSUInteger index, BOOL
result) { }];
```

If you want to use recording export file data, call parse file data block:

```
[xsensDotDevice setDidParseExportFileDataBlock:^(XsensDotPlotData *
_Nonnull plotData) { }];
```

4.13.5 Stop export

Stop export file data.

```
[xsensDotDevice stopExportFileData];
```

4.13.6 Update export status

UpdateExportingStatus block will be updated after calling *startExportFileData* and *stopExportFileData*.

```
[xsensDotDevice.recording setUpdateExportingStatus:^(XSExportStatus
status) { }];
```

Table 19: Export status

Export status	Description
XSExportIsExportingData	The flash is exporting recording data
XSExportIsStopExportingData	Data exporting is stopped

4.14 Firmware update

Continuous firmware releases from Xsens are scheduled for new features, improvements, and bug fixes. With Over-the-Air (OTA) firmware update function in Xsens DOT, you can easily update the sensors to latest firmware version.

With the OTA functions in the SDK, you can do the firmware update in your own application from Xsens DOT update server via OTA.

NOTE:

- Sensors can only upgrade or downgrade when **in charging status**.
- Network connection is required for OTA. Check the network connection before checking for firmware update.

All the firmware update related methods are in the *XsensDotOtaManager.h* header file. And the parameter is *XsensDotDevice* object. Make sure the device is initialized before calling the related methods. All the callback methods are in the *XsesnDotOtaManagerDelegate* class.

4.14.1 Set the delegate

Set the delegate in your ViewController class:

```
- (void)viewWillAppear:(BOOL)animated
{
    [super viewWillAppear:animated];
    [XsensDotReconnectManager setEnable:YES];
    ...
}
```

4.14.2 Check firmware update

Based on the current firmware version and release type, you can check if there is new firmware version available with *checkOtaUpdates*. This function is usually used when you just want to check if there is new firmware available.

```
[[XsensDotOtaManager defaultManager] checkOtaUpdates:device];
```

After calling *checkOtaUpdates*, *onOtaUpdates* method will be triggered. If the result = YES and version is not an empty string, it means there is a firmware that can be updated. The result would be "No" if there is no new firmware available.

```
(void)onOtaUpdates:(NSString *)address result:(BOOL)result
version:(NSString *)version releaseNotes:(NSString *)releaseNotes;
```

Use *checkOtaUpdatesAndDownload* method if you want to download the firmware file (.mfw) after checking and start the OTA process.

After calling *checkOtaUpdatesAndDownload* method, there will be two callbacks. One is the *onOtaUpdates*, which is the same as calling checkOtaUpdates. The other is *onOtaDownload*. If the available firmware has been downloaded, this method will be triggered:

```
(void)onOtaDownload:(NSString *)address version:(NSString *)version;
```

4.14.3 Check firmware downgrade

Firmware downgrade function is provided to downgrade the beta firmware versions to the last stable version. Stable firmware versions cannot downgrade to any previous versions.

Moreover, if a new stable version is available, beta versions cannot rollback to previous stable versions. You can only update the beta versions to the new stable version under this circumstance.

Similar to checking update, you can use *checkOtaRollback* method to check if the sensor can rollback.

```
[[XsensDotOtaManager defaultManager] checkOtaRollback:device];
```

After calling *checkOtaRollback*, *onOtaRollback* method will be triggered. If the result = YES and version is not an empty string, it means that there is a firmware that can be rollbacked. The result would be "No" if there is no firmware available.

```
(void)onOtaRollback:(NSString *)address result:(BOOL)result
version:(NSString *)version releaseNotes:(NSString *)releaseNotes;
```

Use *checkOtaRollbackAndDownload* method if you want to download the firmware file (.mfw) after checking and start the OTA process.

```
[[XsensDotOtaManager
defaultManager]checkOtaRollbackAndDownload:device];
```

After calling *checkOtaRollbackAndDownload* method, there will be two callbacks. One is the *onOtaRollback*, which is the same as calling *checkOtaRollback*. The other is *onOtaDownload*. If the available firmware has been downloaded, this method will be triggered:

```
(void)onOtaDownload:(NSString *)address version:(NSString *)version
```

4.14.4 Start OTA

You can start the OTA process once the target firmware file has been downloaded. Start the OTA by calling *startOta* method:

```
[[XsensDotOtaManager defaultManager] startOta:device];
```

During the OTA process, the firmware file will be transmitted to the sensor and updated. You can get the OTA status from these callback methods:

1. onOtaStart – The OTA has started successfully.
2. onOtaProgress – The OTA is still in progress.
3. onOtaEnd – The OTA has ended successfully and the update or downgrade is done.

```
(void)onOtaStart:(NSString *)address result:(BOOL)result
errorCode:(int)errorCode;
```

```
(void)onOtaProgress:(NSString *)address progress:(float)progress
errorCode:(int)errorCode;
```

```
(void)onOtaEnd:(NSString *)address result:(BOOL)result
errorCode:(int)errorCode;
```

The OTA will fail if any of the above stages fails. There are some common reasons for OTA failure:

1. Failed to send 'start OTA' and 'stop OTA' commands.
2. OTA file is not sent completely and is always retransmitting. This is usually due to the insufficient Bluetooth performance of the mobile device.
3. Sensor is out of charging status during OTA.
4. Sensor disconnects during OTA.

The *onOtaFileMismatch* callback will be called if the new firmware file does not match the current sensor. The OTA process will end.

```
(void)onOtaFileMismatch:(NSString *)address;
```

The *onOtaUncharged* callback will be called if the sensor is not in charging and the OTA process will end.

```
(void)onOtaUncharged:(NSString *)address;
```

4.14.5 Stop OTA

You can stop the OTA when it is still in progress:

```
[[XsensDotOtaManager defaultManager] stopOta:device];
```

After stopOta method, this callback will be triggered:

```
(void)onOtaEnd:(NSString *)address result:(BOOL)result
errorCode:(int)errorCode;
```

4.14.6 Clear the cache file

The downloaded firmware files will be saved in Library/Caches/${BundleId}/ota folder.You can delete them by using *clearCache* method.

```
[[XsensDotOtaManager defaultManager] clearCache]
```

4.15 Mag field mapper

@Elisabetta please add MFM description here:

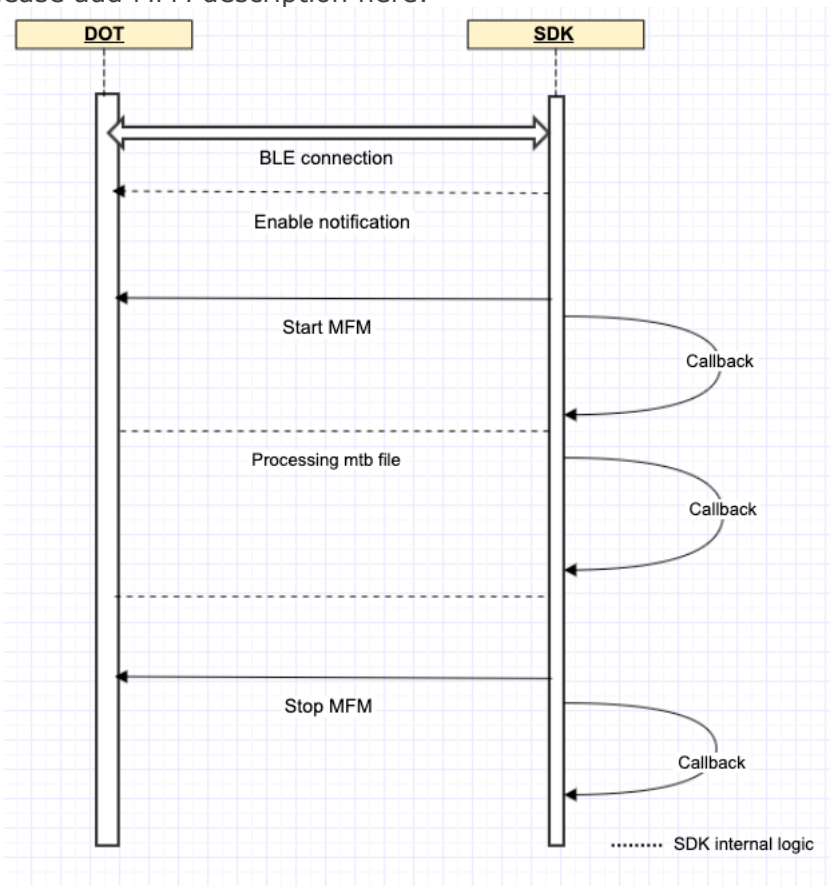

Figure 8: Workflow to MFM

4.15.1 Integrate MFM SDK framework

- Import the whole XsensDotSdkMfm.framework package into the target project.
- Make sure XsensDotSdkMfm.framework is included in **`Target → General → Framework, Libraries, and Embedded Content`**, and the **`Embed property`** is **`'Embed & Sign'`**

4.15.2 Implementation MFM

1. Set MFM delegate in viewController

```
[[XsensDotMFMManager defaultManager] setMfmDelegate:self];
```

2. Implement the XsensDotMFMDelegate interfaces

```
- (void)onMFMProgress:(int)progress address:(NSString *)address
{
  ...
}
```

```
- (void)onMFMCompleted:(XSDotMFMResultTpye)type address:(NSString
*)address
{
  ...
}
```

3. Start MFM
   When calling *startMFM()* method, the MFM process will be started, then the *onMFMProgress()* method will be triggered. Developer can listen the progress of MFM through this. When the progress reaches to 100, the MFM will be on processing state. After MFM process finished, the *onMFMCompleted()* will be triggered, developer can know the MFM state by the parameter type.

```
[[XsensDotMFMManager defaultManager] startMFM:self.mfmDevices];
```

4. Stop MFM
   If you want to stop current MFM process, you can call this *stopMFM()*. After call this method, the MFM process will be stopped and *onMFMCompleted()* will be triggered.

```
[[XsensDotMFMManager defaultManager] stopMFM:self.mfmDevices];
```

4.16 Other functions

4.16.1 Read RSSI

While scanning sensors, you can use the property RSSI to get the RSSI:

```
xsensDotDevice.RSSI
```

You can also read RSSI when sensor is connected by *readRSSI* method.

```
[xsensDotDevice readRSSI:^(NSNumber * _Nonnull signal) {
        }];
```

4.16.2 Identify

To identify or find your device, you can call the following function. The device will fast blink 8 times and then a short pause when you call this function.

```
[xsensDotDevice startIdentifying];
```

4.16.3 Power saving

In power-saving mode, sensors will turn off the signal pipeline and BLE connection and put the MCU in a sleep state to ensure minimum power consumption. The default time threshold to enter power saving mode is set to 10 min in advertisement mode and 30 min in connection mode. These values are saved in the non-volatile memory and can be adjusted in Xsens DOT app or SDK.

There is an example to set power saving time in advertisement and connection mode both to 30 minutes.

```
[xsensDotDevice setPowerSavingTimeout:30 xSecond:0 yMinutes:30
ySeconds:0];
```

4.16.4 Button callback

If there is a single click on the power button during connection, a notification will be sent with a timestamp when this single click is released. This function is called as "Button callback".

When the pressing time is 10~800ms, it is judged as a valid single click. The timestamp is from sensor's local clock and independent of synchronization.

```
[xsensDotDevice setDidButtonCallbackBlock:^(int timestamp) {
        }];
```

4.16.5 Power on options

This feature is to allow user to configure the Xsens DOT v2 sensor to be powered on by USB plugin or not. This setting is only available in v2 sensor.

By default, power on by USB is disabled. So, the sensor will be in charging status if connected with USB cable. You can call *enableUsbPowerOn()* to set enable this feature.

```
[xsensDotDevice enableUsbPowerOn:YES];
```

By enabling USB power on, the sensor will power on immediately after the USB plugin. With this feature, you can power on multiple sensors with the USB plugin at once.

5 Appendix

5.1 Real-time streaming modes

NOTE:
You can get other data quantities from the available data in each measurement mode. Refer to section 4.10.6 for the conversions that can be used.

5.1.1 Extended (Quaternion)

Table 20: Extended (Quaternion)

Mode name	Payload	Available data
XSBleDevicePayloadExtendedQuaternion	36 bytes	• SampleTimeFine • Orientation (Quaternions) • Free acceleration • Status

5.1.2 Complete (Quaternion)

Table 21: Complete (Quaternion)

Mode name	Payload	Available data
XSBleDevicePayloadCompleteQuaternion	32 bytes	• SampleTimeFine • Orientation (Quaternions) • Free acceleration

5.1.3 Orientation (Quaternion)

Table 22: Orientation (Quaternion)

Mode name	Payload	Available data
XSBleDevicePayloadOrientationQuaternion	20 bytes	• SampleTimeFine • Orientation (Quaternions)

5.1.4 Extended (Euler)

Table 23: Extended (Euler)

Mode name	Payload	Available data
XSBleDevicePayloadExtendedEuler	32 bytes	• SampleTimeFine • Orientation (Euler angles) • Free acceleration • Status

5.1.5 Complete (Euler)

Table 24: Complete (Euler)

Mode name	Payload	Available data

XSBleDevicePayloadCompleteEuler	28 bytes	• SampleTimeFine • Orientation (Euler angles) • Free acceleration

5.1.6 Orientation (Euler)

Table 25: Orientation (Euler)

Mode name	Payload	Available data
XSBleDevicePayloadOrientationEuler	16 bytes	• SampleTimeFine • Orientation (Euler angles)

5.1.7 Free acceleration

Table 26: Free acceleration

Mode name	Payload	Available data
XSBleDevicePayloadFreeAcceleration	16 bytes	• SampleTimeFine • Free acceleration

5.1.8 High fidelity (with mag)

Table 27: High fidelity (with mag)

Mode name	Payload	Available data
XSBleDevicePayloadInertialHighFidelityWithMag	35 bytes	• SampleTimeFine • dq • dv • Angular velocity • Acceleration • Magnetic field • Status

5.1.9 High fidelity

Table 28: High fidelity

Mode name	Payload	Available data
XSBleDevicePayloadHighFidelityNoMag	29 bytes	• SampleTimeFine • dq • dv • Angular velocity • Acceleration • Status

5.1.10 Delta quantities (with mag)

Table 29: Delta quantities (with mag)

Mode name	Payload	Available data
XSBleDevicePayloadDeltaQuantitiesWithMag	38 bytes	• SampleTimeFine • dq

		• dv • Magnetic field

5.1.11 Delta quantities

Table 30: Delta quantities

Mode name	Payload	Available data
XSBleDevicePayloadDeltaQuantitiesNoMag	32 bytes	• SampleTimeFine • dq • dv

5.1.12 Rate quantities (with mag)

Table 31: Rate quantities (with mag)

Mode name	Payload	Available data
XSBleDevicePayloadRateQuantitiesWithMag	34 bytes	• SampleTimeFine • Angular velocity • Acceleration • Magnetic Field

5.1.13 Rate quantities

Table 32: Rate quantities

Mode name	Payload	Available data
XSBleDevicePayloadRateQuantitiesNoMag	28 bytes	• SampleTimeFine • Angular velocity • Acceleration

5.1.14 Custom mode 1

Table 33: Custom mode 1

Mode name	Payload	Available data
XSBleDevicePayloadCustomMode1	40 bytes	• SampleTimeFine • Orientation (Euler angles) • Free acceleration • Angular velocity

5.1.15 Custom mode 2

Table 34: Custom mode 2

Mode name	Payload	Available data
XSBleDevicePayloadCustomMode2	34 bytes	• SampleTimeFine • Orientation (Euler angles) • Free acceleration • Magnetic field

5.1.16 Custom mode 3

Table 35: Custom mode 3

Mode name	Payload	Available data
XSBleDevicePayloadCustomMode3	32 bytes	• SampleTimeFine • Orientation (Quaternions) • Angular velocity

5.1.17 Custom mode 4

Table 36: Custom mode 4

Mode name	Payload	Available data
XSBleDevicePayloadCustomMode4	51 bytes	• SampleTimeFine • Orientation (Quaternions) • dq • dv • Angular velocity • Acceleration • Magnetic field • Status

5.1.18 Custom mode 5

Table 37: Custom mode 5

Mode name	Payload	Available data
XSBleDevicePayloadCustomMode5	44 bytes	• SampleTimeFine • Orientation (Quaternions) • Acceleration • Angular velocity