MSensor

MSensor 는 9 축 관성센서(가속도계,자이로스코프,지자기 센서)의 출력을 융합하여 방위각과 자세 정보를 BLE 4.2 를 이용하여 출력하는 센서입니다.

다양한곳에 부착하여 이용가능하도록 46mm* 38mm * 12mm 의 소형으로 제작되었으며, 휴대폰을 이용하여 3D 자세 출력을 사용할 수 있는 프로그램을 제공합니다.

차 례

제 1 장 MSensor 개요

제 2 장 HardWare

제 3 장 MSensor Monitor

제 4 장 프로토콜

제 5 장 GATT(Generic Attribute Profile)

제 6 장 MSensor SDK

제 7 장 Sample Application 사용법

제 8 장 Msensor Application 사용법

제 9 장 Bluetooth 4.0

제 10 장 소스분석

제 1 장 MSensor 개요

MSensor 는 관성센서(가속도계,자이로 스코프)와 지자기 센서를 이용하여 얻은 센서의 자세와 방위각을 BLE 4.2 로 출력하는 모듈이다.

MSensor 주요 기능

1. 센서

16bit 3 축 자이로스코프 : ± 2000 dps

16bit 3 축 가속도센서 : ± 16g

16bit 3 축 지자기 센서 : ± 4800 uT

2. 소프트웨어

자세 및 센서값을 100Hz 출력.

- 센서간 UUID 로 구별 가능
- Gyro,Magnet Calibration 가능
- 3D 모델링 지원
- 지원 OS : Android, Window 10 이상 (블루투스 필요)

3. 기타

- 2 개의 버튼 인터럽트
- 388mAh 리튬베터리 사용 (Micro -B 타입 케이블로 충전 가능)

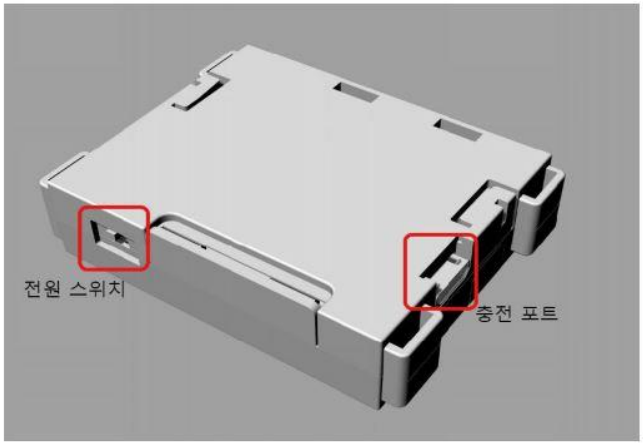
-풀 충전시 사용 가능시간

ㅁ 연속 동작시 : 40 시간

ㅁ 대기 상태시 : 1440 시간(2 달)

-저전력 Sleep 모드 지원.

제 2 장 HardWare



전원스위치 : 모듈 좌측에 있으며, 아래로 내렸을때가 ON.

충전 포트 : Micro B type 으로 충전이 가능함.

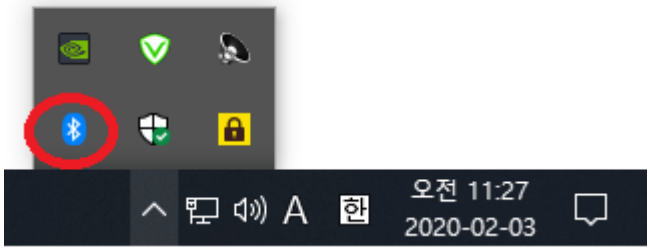
버튼 1,2 : 모듈 양쪽에 하나씩 있으며, LED 는 사용하지 않음.

제 3 장 MSensor Monitor

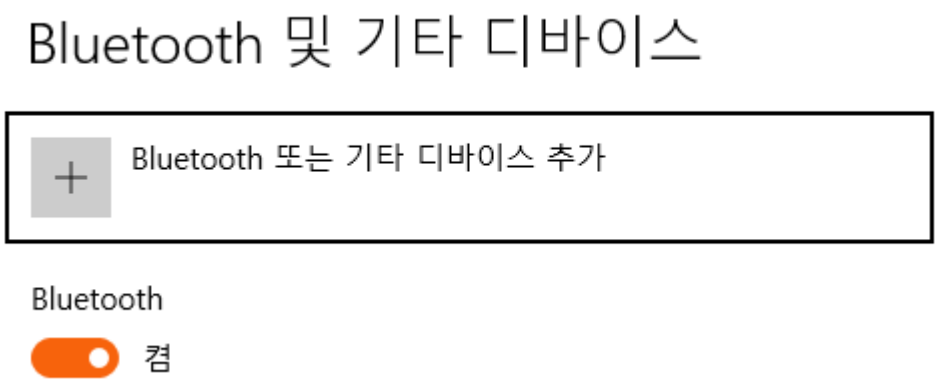
MSensor Monitor 는 PC 에서 MSensor 의 값을 확인하고, 여러 설정을 변경하기 위해 작성된 프로그램입니다. 이 프로그램은 Window10 이상에서만 동작하며, 블루투스 4.0 이상의 동글을 사용하거나, 자제적으로 탑제하고 있어야만 올바르게 동작 시킬 수 있습니다.

(이때 정상적으로 실행되지 않는다면, 설치한 폴더안의 "vc_redist_x64.exe 혹은 vc_redist_x86.exe"를 실행 시켜주십시오.)

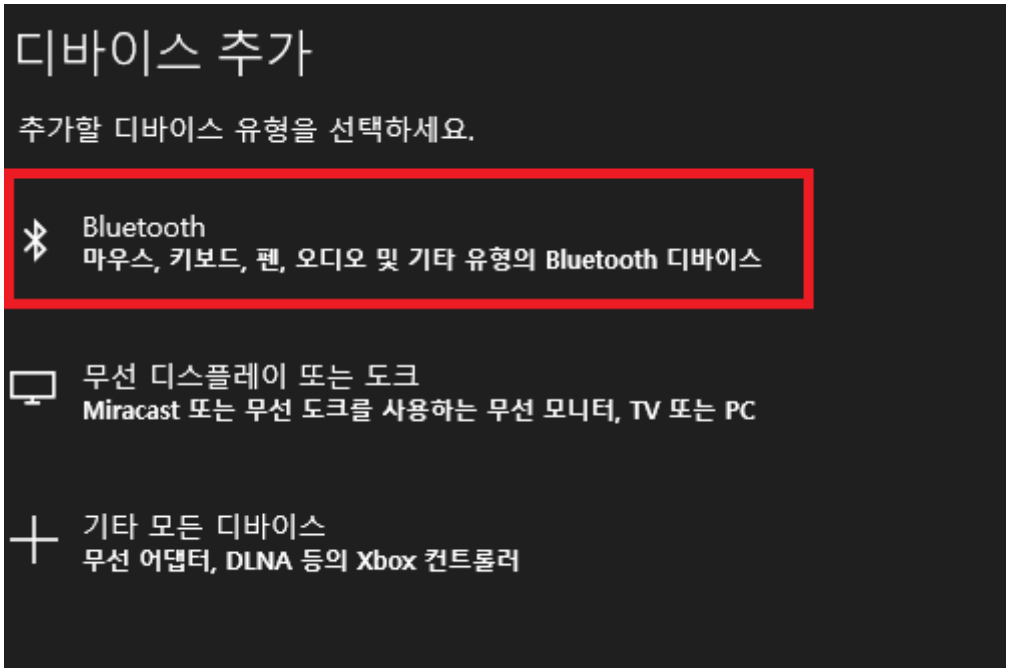
-1 절 시작하기.



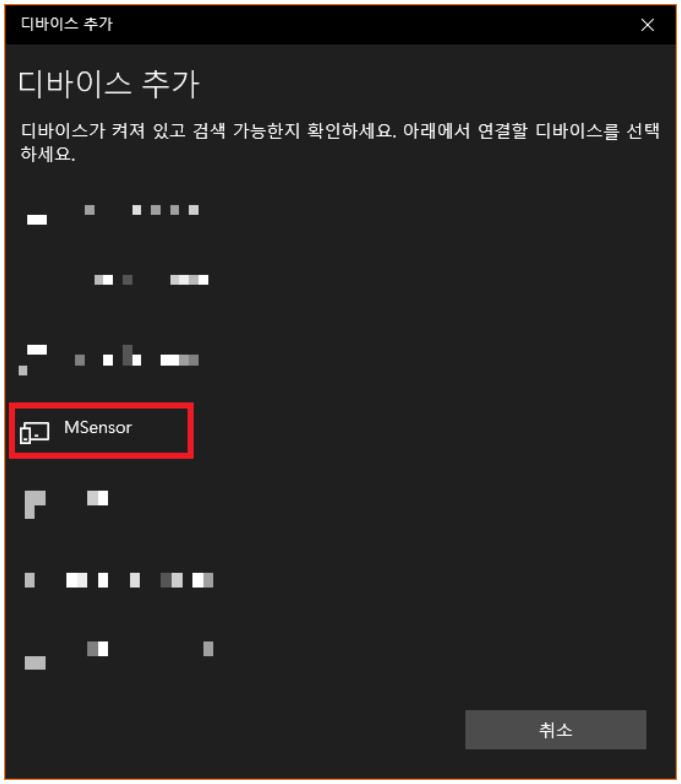
1 .혹은 자체 탑제된 블루투스 설정을 엽니다. 작업 표시줄에 표시된 블루투스 아이콘에서 오른쪽 버튼을 누른뒤, "블루투스 장치 추가"를 누릅니다.



2. 화면의 가장 상단의 "Bluetooth 또는 기타 디바이스 추가"를 누릅니다.



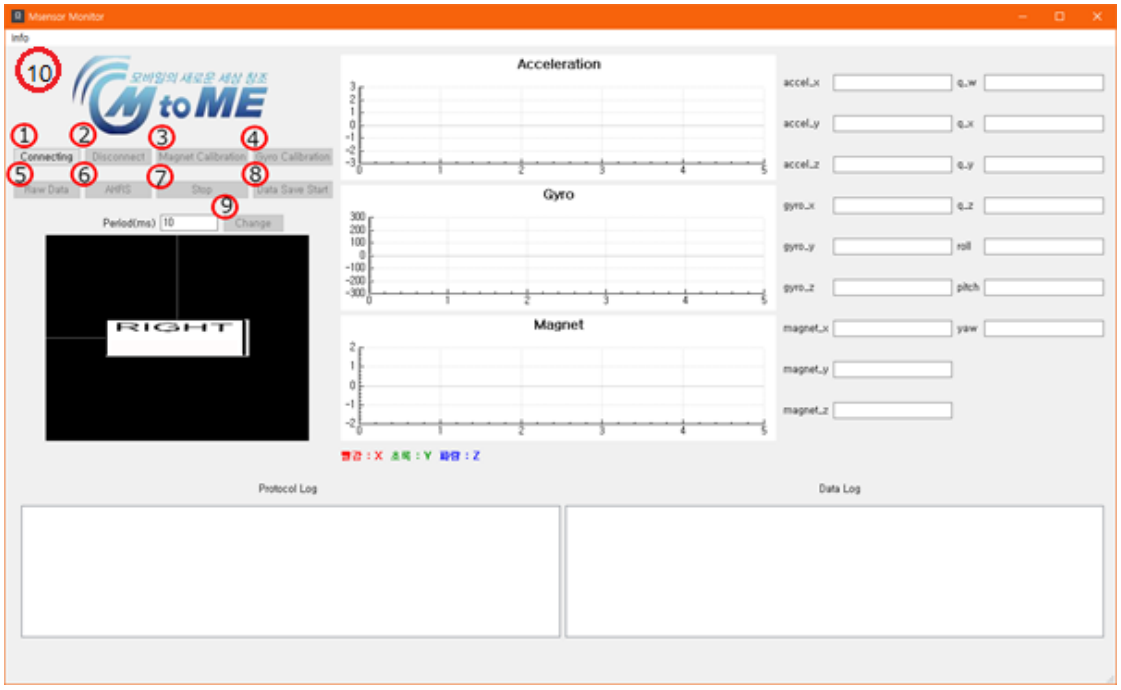
3. 이후 화면에서 가장 상단의 "Bluetooth"를 누릅니다.



4. 블루투스 목록에서 MSensor 를 찾아 클릭하여 연결합니다.

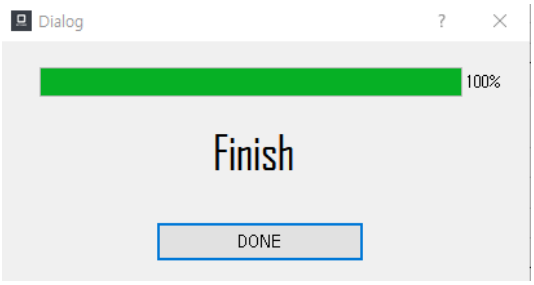
(정상적으로 연결되었다면 해당 화면이 보이게 됩니다.)

-2 절 프로그램 사용하기

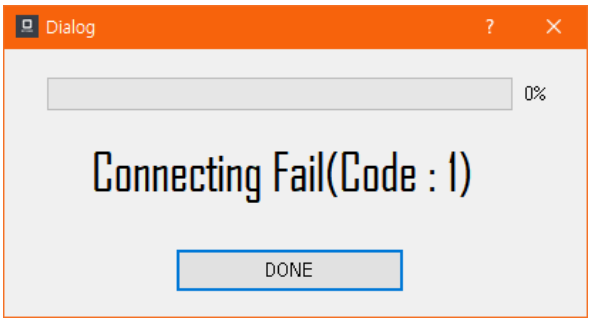


1. Connecting

1 절의 시작하기 부분을 정상적으로 진행했다면, 블루투스의 서비스와 특성을 읽어와서 프로그램을 사용할 수 있도록 세팅하는 버튼입니다.



위 사진처럼 화면이 떳을 때, Done 버튼을 눌르면, 창이 닫히며 프로그램을 시작하실 수 있습니다. 도중 x 를 눌러 창을 닫을 시 정상동작하지 않을 수 있습니다.



만약 1 절의 시작하기 부분을 제대로 하지 않았을 경우 다음 화면이 나타날 것입니다.

이때 에러코드 별 에러 내용은 다음과 같습니다.

Error Code 1 : 컴퓨터의 BLE 장치가 없거나, 블루투스 기능이 꺼져있거나 센서가 페어링 컴퓨터에 페어링되어있지 않을 때 발생합니다.

Error Code 2 : 센서의 전원이 꺼져있는 경우 발생합니다. 센서의 전원을 확인해 주세요.

Error Code 3 : MSensor 외, 다른 센서를 블루투스에 등록한 뒤 실행할 경우 발생합니다.

Error Code 4 : 블루투스 장치를 2 개이상 동시에 연결해둔 경우 발생합니다.

현재 프로그램은 1 개의 장치만 지원합니다.

2. Disconnect

현재 프로그램과 센서와 연결을 해제합니다.

3. Magnet Calibration

센서는 기본적으로 Calibration 이 된상태로 출고되나, 값이 너무 이상하면 해당 버튼을 통해

다시 Calibration 을 진행할 수 있습니다. 현재는 Calibration 값을 해당 방식으로만 변경이 가능합니다.

4. Gyro Calibration

Gyro 센서를 재조정 합니다.

5. Raw Data

센서에서 Raw Data(gyro,accel,magnet)을 출력합니다. BLE 특성상 한패킷당 20byte 이하로만 전송이 가능하여 Raw Data 와 쿼터니언 부분을 분리하였습니다. 해당버튼을 누르면 화면의 중간 부분 그래프에 값이 나오며, 화면 오른쪽에서 값을 확인 할 수 있습니다.

6. AHRS

센서에서 쿼터니언값을 출력합니다. Roll pitch yaw 값은 모듈에서 나온 쿼터니언을 이용하여 qt 라이브러리를 이용하여 계산하였으며, 해당 버튼을 누르면 버튼 아랫부분 3D 상자가 센서의 움직임에 따라 움직이며, 화면 오른쪽에서 실제 데이터값을 확인 할 수 있습니다.

7. Stop

Raw Data 혹은 AHRS 를 측정을 종료할 때 해당버튼을 눌러 측정을 종료합니다.

8. Data Save Start/Data Save Stop

Raw Data 혹은, 쿼터니언 측정 중 버튼이 활성화 되며, 해당 버튼을 누를시 데이터를

.txt 파일 형식으로 저장합니다. 이후 다시한번 버튼을 누르면 저장이 종료되며,

프로그램이 설치된 폴더에 "Data_log_번호"로 저장이 됩니다.

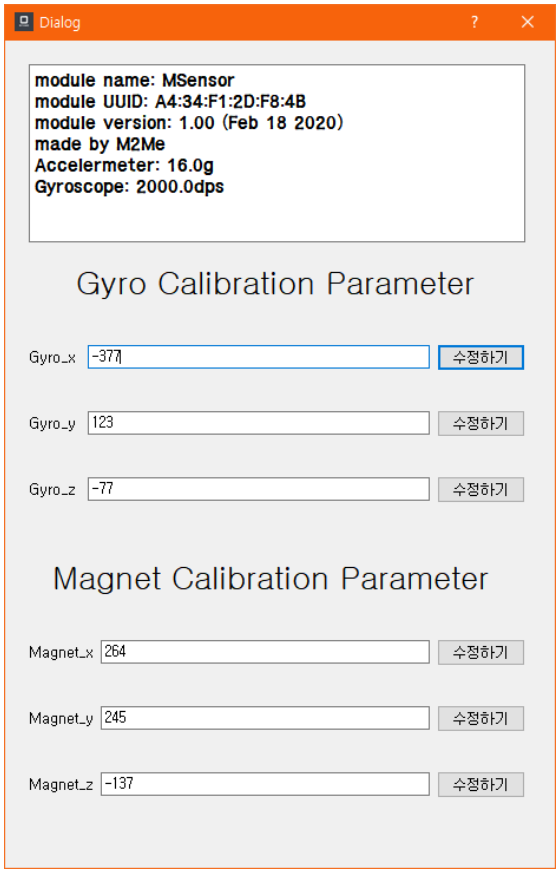
(번호는 폴더내 같은 파일이 있을 때 1 씩 증가합니다.)

9. Period Change

Period 를 변경합니다. 이때는 프로토콜로 변경하는 것이 아닌, 특성에 직접 접근하여 Period 를 변경합니다. (UUID: f000aa83-0451-4000-b000-000000000000)

Period 는 기본값으로 10ms 로 설정되어 있으며, AHRS 에서는 10ms~1000ms 까지 설정이 가능하며, Raw Data 상태에서는 10ms~2500ms 까지 설정이 가능합니다. 이후 측정 종료후 다시 측정시작을 누르면 10ms 로 변경됩니다.

10. Config



센서 정보를 확인 할 수 있으며, Calibration Parameter 를 수정할 수 있습니다.

제 4 장 프로토콜

MSensor 의 BLE4.2 를 이용하여 통신하는 방법에 대하여 기술한다. 이를통해

센서에서 데이터를 얻어오거나 설정을 변경 할 수 있다.

-전송 프로토콜

DATA1(1byte)	DATA2(1byte)	CMD(1byte)

DATA1,DATA2 : 통신에서 값을 요구하는 CMD 일 경우, 해당 위치에 값을 넣는다.

CMD : 어떤 동작을 할지에 대한 명령어를 넣는다.

- 프로토콜 실행 성공시

0	ACK(0X06)	CMD

-프로토콜 실행 실패시

Error Code	NAK(0X03)	CMD

Error Code 목록

1 : 현재 상태에서 CMD 수행불가 (AHRS 혹은 RAW 데이터 출력 상태에서는 출력 중지 외

다른 CMD 수행 불가)

2 : 등록되지 않은 CMD 를 입력했을 때

3 : 모듈에 등록되지 않은 타입으로 변경을 시도 했을 때

4 : 센서 타입을 지정하지 않은 상태에서 AHRS 혹은 RAW 데이터 출력을 시도할때

- CMD 목록

- 0x01 : 타입을 지정

0	TYPE	0x01

TYPE -> 0 : Quarterion 값 출력

TYPE -> 1 : RAW 데이터 (accel,gyro,magnet) 출력

-0x02 : 데이터 출력 시작

0	0	0x02

-0x03 : 데이터 출력 중지

0	0	0x03

-0x04 : Gyro Calibration 시작

0	0	0x04

Gyro Calibration 은 움직임이 없는곳에 가만히 약 1 초정도 가만히 두는 방식으로 진행되며,

해당값은 센서의 전원을 껏다 키면 사라지므로, 센서를 껏다가 켰을 때

초기 1 회 실행하는 것을 추천함.

-0x05 : Magnet Calibration 시작

0	0	0x05

Magnet Calibration 은 약 10 초 정도 진행되며, 진행되는 동안 센서를

다양한 축으로 회전시키면 된다. (예시 영상 : https://youtu.be/DLKqmFu_GD4?t=156)

-0x06 : gyro,magnet Calibration Parameter 출력

0	0	0x06

MSensor 는 다음과 같은 방식으로 센서값을 보정합니다.

$$Y = F(X-A)$$

Y = 보정된 센서값

F = 변환식

X = 센서 출력값 (Raw Data)

A = Calibration Parameter

해당 명령어는 현재 센서에 적용중인 Calibration Parameter 를 출력합니다.

명령어를 센서에 전달 시 Movement 서비스의 SensorData 특성으로 현재 설정된

Gyro x,y,z Magnet x,y,z 순서로 Calibration Parameter 를 전송합니다.

(이값은 Raw 데이터에 대한 Calibration 값입니다.)

-0x07: Gyro_x Calibration Parmeter 변경

Data_H	Data_L	0x07

Sensor 의 gyro 값중 x 축에 적용된 Calibration Parameter 를 Data 로 변경시킵니다.

Data 는 16bit 이고, 8byte 씩 분해되어 전송되므로 결합하는 과정이 필요합니다.

-0x08: Gyro_y Calibration Parmeter 변경

Data_H	Data_L	0x08

Sensor 의 gyro 값중 y 축에 적용된 Calibration Parameter 를 Data 로변경시킵니다.

-0x09: Gyro_z Calibration Parmeter 변경

Data_H	Data_L	0x09

Sensor 의 gyro 값중 z 축에 적용된 Calibration Parameter 를 Data 로변경시킵니다.

-0x0A: Magnet_x Calibration Parmeter 변경

Data_H	Data_L	0x0A

Sensor 의 Magnet 값중 x 축에 적용된 Calibration Parameter 를 변경시킵니다.

-0x0B: Magnet_y Calibration Parmeter 변경

Data_H	Data_L	0x0B

Sensor 의 Magnet 값중 y 축에 적용된 Calibration Parameter 를 변경시킵니다.

-0x0C: Magnet_z Calibration Parmeter 변경

Data_H	Data_L	0x0C

Sensor 의 Magnet 값중 z 축에 적용된 Calibration Parameter 를 변경시킵니다.

제 5 장 GATT(Generic Attribute Profile)

-Profile 목록

-Generic Attribute(UUID : 00001801-0000-1000-8000-00805f9b34fb)

ㅇ BLE 에서 정의된 서비스로 장치에 대한 일반적인 설명이 들어가는 Service

ㅁ Characteristic 목록

- Device Name
- Apperance
- Peripheral Preferred Connection Parameter

-Device Information(UUID : 0000180a-0000-1000-8000-00805f9b34fb)

ㅇ모듈에 대한 정보가 들어있는 Service

ㅁ Characteristic 목록

-System ID

-Model Number

-Serial Number

-FirmWare Version information

-HardWare Revision information

-SoftWare Revision information

-Manufacturer name information

-IEEE 11083-20601 Requlatory Certification Data List

-PnP ID

-Battery Service(UUID : 0000180f-0000-1000-8000-00805f9b34fb)

ㅇ베터리에 대한 정보가 있는 서비스

ㅁ Characteristic 목록

-Batterty Level

-Movement Service(UUID : f000aa80-0451-4000-b000-000000000000)

ㅇ센서에서 데이터가 나오는 서비스

ㅁ Characteristic 목록

-SensorData (UUID : f000aa81-0451-4000-b000-000000000000)

-SensorConfig(UUID : f000aa82-0451-4000-b000-000000000000)

-SensorPeriod(UUID : f000aa83-0451-4000-b000-000000000000)

-Terminal Service(UUID : f000aa70-0451-4000-b000-000000000000)

ㅇ BLE 를 통해서 명령을 주고받는 서비스.

ㅁ Characteristic 목록

-Response Data (UUID : f000aa71-0451-4000-b000-000000000000)

-Receive Data (UUID : f000aa72-0451-4000-b000-000000000000)

-Reserved

-SimpleKey Service (UUID : 0000ffe0-0000-1000-8000-00805f9b34fb)

ㅇ 모듈의 KEY 상태를 파악하는 서비스 (버튼 인터럽트)

ㅁ Characteristic 목록

-Key press state (UUID : 0000ffe1-0000-1000-8000-00805f9b34fb)

주요 Characteristic 설명

- Movement Service 중 SensorData (UUID : f000aa81-0451-4000-b000-000000000000)

: 센서 데이터가 출력되는 부분으로 타입에 따라 9 축 데이터(각속도,가속도,지자기) 혹은 자세값(Euler angle, Quaternion) 을 다음과 같은 형태로 출력한다.

Properties 는 READ 와 Notify 로 설정되어 있다.

X_H	X_L	Y_H	Y_L	Z_H	Z_L	X_H	X_L	Y_H	Y_L	Z_H	Z_L	X_H	X_L	Y_H	Y_L	Z_H	Z_L

타입 1 : 9 축 데이터 출력

빨강 : GyroScope 값 파랑 : Accelerometer 값 초록 : Magnetmeter 값

W_H	W_L	X_H	X_L	Y_H	Y_L	Z_H	Z_L	0	0	0	0	0	0	0	0	0	0

타입 2 : Quaternion , Euler angle 데이터 출력

빨강 : Quaternion 값

-Terminal Service 중

Response Data (UUID : f000aa71-0451-4000-b000-000000000000),

Receive Data (UUID : f000aa72-0451-4000-b000-000000000000)중

Response Data 는 센서에서 받은 프로토콜에 대한 응답을 출력해 주며

Receive Data 에 프로토콜을 작성하면 센서로 전달된다.

Response Data 의 Properties 는 Read 와 notify 로 설정되어 있으며,

Receive Data 의 Properties 는 Read 와 Write 로 설정되어 있다.

- SimpleKey Service 중 Key press state (UUID : 0000ffe1-0000-1000-8000-00805f9b34fb)

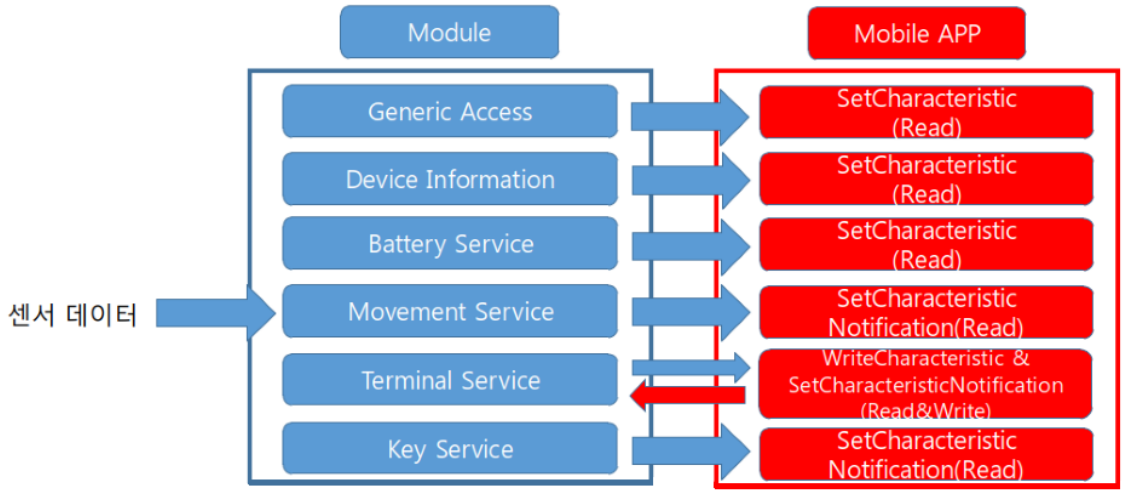
: Key press state 는 현재 버튼 상태에 대한 정보를 나타내며, 상태에 따른 값은 다음과 같다.

0 : 아무것도 안눌린 상태

1 : 2 번 스위치가 눌림.

2: 1 번 스위치가 눌림.

3: 2 개의 스위치가 모두 눌림.


Module
Mobile APP
Generic Access
Device Information
Battery Service
Movement Service
Terminal Service
Key Service
센서 데이터
SetCharacteristic (Read)
SetCharacteristic (Read)
SetCharacteristic (Read)
SetCharacteristic Notification(Read)
WriteCharacteristic & SetCharacteristicNotification (Read&Write)
SetCharacteristic Notification(Read)

제 6 장 MSensor SDK

-제 1 절 개요

MSensor 의 제어 어플리케이션을 쉽게 제작할 수 있도록 MSensor Monitor 코드를 제공합니다.

MSensor 의 SDK 는 PC 에서는 Window10 이상에서만 지원합니다. MSensor Monitor 코드는 QT(4.10.2)에서 작성되었으며, 해당 문서에서는 BLE 를 이용하여 MSensor 에 연결하고 통신하는 부분에 대하여 설명할 것입니다.

-예제 프로그램 실행

다운받은 폴더안의 MSensor_Monitor.pro 파일을 실행하여 qt 를 실행시킨뒤, 디버깅을 진행하여 실행시킵니다.

-제 2 절 프로그램 코드 설명

프로그램을 새로 만든다고 한다면, characteristicinfo,deviceinfo,serviceinfo 코드는 블루투스를 이용하여 센서와 동글 혹은 pc 와 연결하여 정보를 읽어오는 코드로 별다른 수정없이 그대로 사용하시면 됩니다.

Data_Convert.cpp : 센서에서 읽어온 데이터를 단위에 맞게 변환하는 공식이 들어있습니다.

Geometryengine,miwidget,objectgl.cpp : 3D 큐브를 그리기 위한 클래스 입니다.

Protocol.cpp : 현재 센서에 사용되는 프로토콜을 쉽게 사용하기 위한 클래스입니다.

qCustomplot.cpp : 그래프를 그리기 위한 클래스 입니다.

My_messagebox.cpp : message 박스를 쉽게 만들기 위한 클래스입니다.

Mainwindow.cpp : 눌린 버튼에 따른 기능을 수행하는 클래스입니다.

Main.cpp : 초기에 메인 윈도우를 띄어줍니다.

Device.cpp : 장치를 관리하는 클래스입니다.(사실상 메인)

-제 3 절 주요 코드 예제 및 설명

Device.cpp 에서 연결부터 데이터를 센서로 수신하여 센서로 전달하고, pc 화면에 보여주기 위해 mainWindow 로 전송하는것까지 전부 진행하므로, 프로그램에서 가장 중요한 부분이니다.

연결을 진행할 때, Device.cpp 에서 진행하는 과정은 다음과 같습니다.

1. Device 검색 (BLE 에 MSensor 가 연결되어 있는지 확인)
2. MSensor 가 있는 것을 확인하면 연결을 시도합니다.
3. 페어링에 성공한다면, 센서의 GATT SERVER 에서 먼저 서비스목록을 읽어옵니다.

   현재 모듈에서 사용하는 서비스의 UUID 에 해당하는 찾고 서비스를 저장합니다.
4. 찾은 서비스에 접근하여 Characteristic 을 읽어온 뒤 필요한 특성을 저장합니다.
5. 과정 중 에러가 있었는지 판별하며 없었다면 PC 프로그램을 활성화 시키고 연결 과정을 끝냅니다.

Device 클래스의 함수로 적으면,

StartDiscovery ->addDevice ->scanServices ->deviceconnected ->connectToService-> serviceDetailsDiscovered ->deviceScanFinished

순서로 진행됩니다.

현재 코드에서는 연결과정을 하면서 DataService 에서, DataRecive_character, DataPeriod_character, Recive_character, Terminal_character 를 저장하여 프로그램에서 센서와 데이데이 주고받을 때 사용합니다. 또한 이때 Terminal 서비스와 Characteristic 서비스에서 데이터가 변화하면 SIGNAL 이 발생하는데, 이것을 SLOT 에 연결하는 작업도 같이 진행합니다.

-Terminal Service

```
connect(Terminal_Service,
SIGNAL(characteristicChanged(QLowEnergyCharacteristic,QByteArray)),
                    this,
SLOT(ReceiveValue(QLowEnergyCharacteristic,QByteArray)));

connect(Terminal_Service,
SIGNAL(characteristicWritten(QLowEnergyCharacteristic,QByteArray)),
                    this,
SLOT(Written_Check(QLowEnergyCharacteristic,QByteArray)));
```

-Data_Service

```
connect(Data_Service,
SIGNAL(characteristicChanged(QLowEnergyCharacteristic,QByteArray)),
                    this,
SLOT(DataReceiveValue(QLowEnergyCharacteristic,QByteArray)));

connect(Data_Service,
SIGNAL(characteristicWritten(QLowEnergyCharacteristic,QByteArray)),
                    this,
SLOT(Written_Check(QLowEnergyCharacteristic,QByteArray))
```

센서에 데이터를 전송할때는 Protocol 클래스의 send_protocol 를 이용하여 전송할 수 있으며 프로토콜은 제 4 장을 참고하여 QByteArray 를 이용하여 입력해 주면 됩니다.

Ex) 데이터를 쿼터니안값이 나오도록 변경

```
Protocol::send_protocol(QByteArray::fromHex("000201"))
```

데이터 출력 시작

```
Protocol::send_protocol(QByteArray::fromHex("000002"))
```

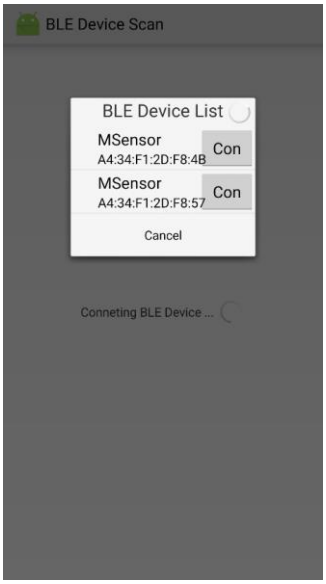
받은 데이터는 Protocol 클래스의 Interpret protocol,Interpret data 함수를 이용하여

프로토콜과 데이터를 해석할 수 있으며 이후 나온 데이터를 이용하면 됩니다.

제 7 장 Sample Application 사용법

- https://www.m2me.co.kr:4000/MSensor/apk/MSensorSample.apk

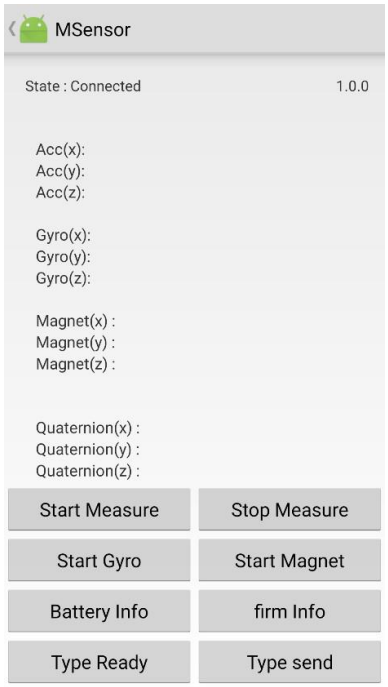
Sample Application apk 다운로드 주소입니다.



1. Msensor 와 연결하기.

어플리케이션을 실행하면 오른쪽 사진과 같이 검색된

MSensor 중 연결을 원하는 Msensor 의 'Con'버튼을 누릅니다.



'Con'버튼을 누르면 해당 센서와 연결을 시도하고 연결이 완료되면 왼쪽 사진과 같이 화면 아래쪽의 버튼들이 나타나게 됩니다.

2. 각 버튼 설명

-Start Measure : 설정된 타입으로 측정을 시작합니다.

-Stop Measure : 측정을 종료합니다.

-Start Gyro : Gyro Calibration 을 시작합니다.

-Start Magnet : Magnet Calibration 을 시작합니다. 방법은 PC 프로그램과 같습니다.

-Battery Info : 현재 모듈의 베터리 잔량을 표기합니다.

-Firm Info : 현재 모듈 펌웨어 버전을 나타냅니다.

-Type Ready : Type 을 변경합니다.

*타입은 두가지가 있으며 다음과 같습니다.

- 1 번 타입은 RAW 데이터 (gyro,accel,magnetic 값을 출력합니다)
- 2 번 타입은 Quaternion 데이터를 출력합니다.

-Type send : 설정된 타입을 모듈로 전송합니다.

* 측정 시작버튼을 누르기전에 Type 을 먼저 설정해주지 않으면 측정시작이 되지 않습니다.

제 8 장 Msensor Application 사용법

- https://www.m2me.co.kr:4000/MSensor/apk/MSensor.apk

다운로드 링크

Msensor Application 은 휴대폰의 BLE 를 이용해 MSensor 를 모니터링 할 수 있는 프로그램입니다.

1. Connect 버튼입니다. 휴대폰에 연결할때는 휴대폰의 블루투스 기능을 이용하여 연결하지 않고

해당 버튼을 누르고, 검색된 MSensor 를 선택하여 연결해야합니다.

2. Calibration 을 할 수 있는 버튼입니다.

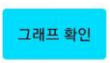
Gyro Calibration 은 평평한 바닥에 약 3 초 정도 가만히 놓아두고 버튼을 누르면 되며, 도중 센서에 힘이 가해져서는 안됩니다.

Magnet Calibration 은 캘리브레이션 버튼을 누른뒤, 약 10 초간 진행되며 그동안 센서를 모듈을 3 축으로 회전시키거나, 8 자를 그리며 센서를 흔들어 주면 됩니다.
(참고 영상 :https://www.youtube.com/watch?v=y80R2nVlzuQ)

3. 메뉴를 표시하는 공간이며, 크게 3 가지가 있습니다.

1. Measure 는 Calibration 을 진행하는 화면입니다.

2. RECORD 에서 3D 시작을 누르면, 센서에서는 쿼터니언값이 출력되기 시작하며 유니티를 통해 3D 큐브로 센서에서 나오는 값을 확인 할 수 있습니다.



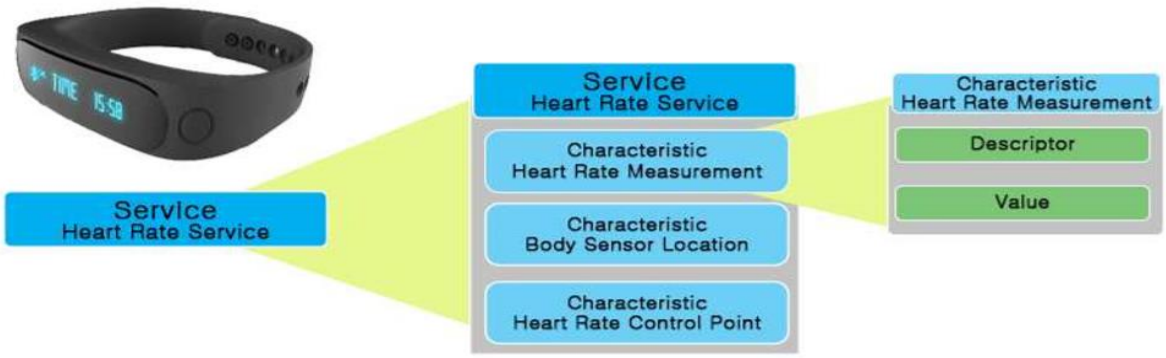
3. GRAPH 에서 그래프 확인을 누르면, 센서에서는 RAW 9 축 데이터가 출력되며 해당값을 그래프로 찍어줍니다.

제 9 장 Bluetooth 4.0

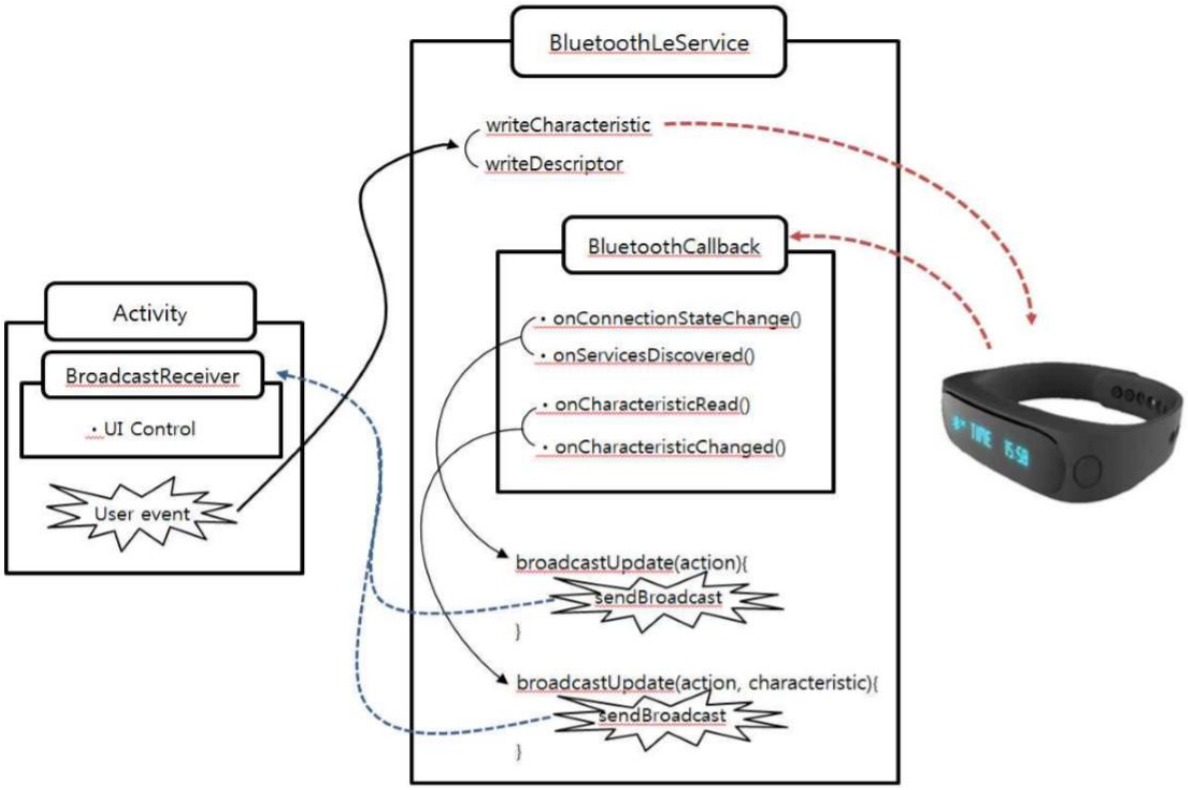
BLE 란 Bluetooth Low Energy 의 약자로 Bluetooth4.0 을 말한다. Bluetooth3.0 은 전송 속 도를 높이는데 초점을 맞췄고 Bluetooth4.0 은 전력소비를 줄이는데 초점을 맞춰 이전 기술 에 비해 전력소모량을 최대 90%까지 줄였다. 때문에 소형 배터리로 수년 이상 사용이 가 능하며 보안, 홈 엔터테인먼트, 헬스케어 및 각종 피트니스 제품들에서 사용되고 있다. (Android 4.3 이후 버전에서 사용 가능)

주요 용어 및 설명

GATT (Generic Attribute profile) : BLE 기기간 데이터(Service 와 Characteristic) 전 송 방법을 정의. Attribute Protocol (ATT) : 데이터 유닛을 전송하는 방법의 정의. 각각의 속성은 128bit 의 UUID 를 가지며 ATT 에 의해 부여된 속성(Attribute)은 서비스(Service)와 특성 (Characteristic)을 결정. 다른 기기에 'attribute' 라는 데이터를 노출시킨다. Service : 특성(Characteristic)들의 집합. 한 개의 Service 마다 고유한 UUID 제공. Characteristic : 특성을 설명하는 여러개의 Descriptor 와 하나의 값을 포함. Descriptor : 특성의 값을 설명.



위의 그림은 'Heart Rate Serivce'를 제공하는 블루투스 기기가 있다고 가정했을때의 프로필이다. (한 기기에 여러개의 Service가 있을 수 있다)



위의 그림은 BLE 장치와의 통신을 간략하게 나타낸 그림이다. Activity는 Service를 통해 BLE 장치와 통신을 하게된다.

제 10 장 소스분석

블루투스장치를 검색하고 통신하려면 manifest에 퍼미션을 추가해줘야 한다.

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

BLE를 지원하는 장치만 동작하도록 하려면 아래의 퍼미션을 manifest에 추가해준다.

```
<uses-feature android:name="android.hardware.bluetooth_le" android:required="true"/>
```

BLE 지원 여부를 다음코드로 알아낼 수 있다.

```
if (!getPackageManager().hasSystemFeature(PackageManager.FEATURE_BLUETOOTH_LE)){
    Toast.makeText(this, R.string.ble_not_supported, Toast.LENGTH_SHORT).show();
    finish();
}
```

BluetoothAdapter를 얻는다.

```
final BluetoothManager bluetoothManager =
        (BluetoothManager) getSystemService(Context.BLUETOOTH_SERVICE);
mBluetoothAdapter = bluetoothManager.getAdapter();
```

만약 블루투스가 활성화 되어있지 않으면 블루투스 활성화 다이얼로그를 띄운다.

```
if (mBluetoothAdapter == null || !mBluetoothAdapter.isEnabled()) {
    Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
    startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
}
```

위의 과정은 BLE장치와 연결하기 위한 준비과정이었다면 이제는 BLE 장치를 검색하여 GATT 서버에 연결하는 과정이다.

startLeScan() 메소드를 호출하여 BLE 장치를 검색한다. 장치가 검색되면 LeScanCallback 이 호출된다.

```
private void scanLeDevice(final boolean enable) {
    if (enable) {
        mHandler.postDelayed(new Runnable() {
            @Override
            public void run() {
                mScanning = false;
                mBluetoothAdapter.stopLeScan(mLeScanCallback);
            }
        }, SCAN_PERIOD);

        mScanning = true;
        mBluetoothAdapter.startLeScan(mLeScanCallback);
    } else {
        mScanning = false;
        mBluetoothAdapter.stopLeScan(mLeScanCallback);
    }
}
```

검색된 BLE기기의 GATT 서버에 연결한다.

```
mBluetoothGatt = device.connectGatt(this, false, mGattCallback);
```

GATT 서버에 연결되면 Callback이 호출되면서 broadcastUpdate()메소를 호출한다.

```
private final BluetoothGattCallback mGattCallback = new BluetoothGattCallback(){
    @Override
    public void onConnectionStateChange(BluetoothGatt gatt, int status, int newState) {
        String intentAction;
        if (newState == BluetoothProfile.STATE_CONNECTED) {
            intentAction = ACTION_GATT_CONNECTED;
            mConnectionState = STATE_CONNECTED;
            broadcastUpdate(intentAction);
        } else if (newState == BluetoothProfile.STATE_DISCONNECTED) {
            intentAction = ACTION_GATT_DISCONNECTED;
            mConnectionState = STATE_DISCONNECTED;
            broadcastUpdate(intentAction);
        }
    }

    @Override
    public void onServicesDiscovered(BluetoothGatt gatt, int status) {
        if (status == BluetoothGatt.GATT_SUCCESS) {
            broadcastUpdate(ACTION_GATT_SERVICES_DISCOVERED);
        }
    }

    @Override
    public void onCharacteristicRead(BluetoothGatt gatt,
            BluetoothGattCharacteristic characteristic, int status) {
        if (status == BluetoothGatt.GATT_SUCCESS) {
            broadcastUpdate(ACTION_DATA_AVAILABLE, characteristic);
        }
    }
}
```

broadcastUpdate()메소드에서는 sendBroadcast를 해 Activity에 필요한 이벤트를 전달한다.

broadcastUpdate(final String action) 메소드는 BLE 장치와의 Connection 상태나 새로 발견된 Service를 sendBroadcast를하고, broadcastUpdate(final String action, final BluetoothGattCharacteristic characteristic) 메소드는 BLE 장치의 특성활동을 action에 넣어 sendBroadcast한다.

```
private void broadcastUpdate(final String action) {
    final Intent intent = new Intent(action);
    sendBroadcast(intent);
}

private void broadcastUpdate(final String action,
                             final BluetoothGattCharacteristic characteristic) {
    final Intent intent = new Intent(action);

    if (UUID_HEART_RATE_MEASUREMENT.equals(characteristic.getUuid())) {
        int flag = characteristic.getProperties();
        int format = -1;
        if ((flag & 0x01) != 0) {
            format = BluetoothGattCharacteristic.FORMAT_UINT16;
        } else {
            format = BluetoothGattCharacteristic.FORMAT_UINT8;
        }
```

```
        final int heartRate = characteristic.getIntValue(format, 1);
        intent.putExtra(EXTRA_DATA, String.valueOf(heartRate));
    } else {
        final byte[] data = characteristic.getValue();
        if (data != null && data.length > 0) {
            final StringBuilder stringBuilder = new StringBuilder(data.length);
            for(byte byteChar : data)
                stringBuilder.append(String.format("%02X ", byteChar));
            intent.putExtra(EXTRA_DATA, new String(data) + "\n" +
                    stringBuilder.toString());
        }
    }
    sendBroadcast(intent);
}
```

위에서 보내진 event들은 Activity의 BroadcastReceiver가 받아서 각event에 맞는 적절한 액션을 취하게 된다.

```
private final BroadcastReceiver mGattUpdateReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        final String action = intent.getAction();
        if (BluetoothLeService.ACTION_GATT_CONNECTED.equals(action)) {
            mConnected = true;
            updateConnectionState(R.string.connected);
            invalidateOptionsMenu();
        } else if (BluetoothLeService.ACTION_GATT_DISCONNECTED.equals(action)) {
            mConnected = false;
            updateConnectionState(R.string.disconnected);
            invalidateOptionsMenu();
            clearUI();
        }elseif(BluetoothLeService.ACTION_GATT_SERVICES_DISCOVERED.equals(action))
{
            displayGattServices(mBluetoothLeService.getSupportedGattServices());
        } else if (BluetoothLeService.ACTION_DATA_AVAILABLE.equals(action)) {
            displayData(intent.getStringExtra(BluetoothLeService.EXTRA_DATA));
        }
    }
};
```

BroadcastReceiver에서 넘겨받은 BLE Attribute를 읽어서 처리하는 코드이다.

```
private void displayGattServices(List<BluetoothGattService> gattServices) {
    if (gattServices == null) return;
    String uuid = null;
    String unknownServiceString = getResources().getString(R.string.unknown_service);
    String unknownCharaString = getResources().getString(R.string.unknown_characteristic);
    ArrayList<HashMap<String, String>> gattServiceData =
            new ArrayList<HashMap<String, String>>();
    ArrayList<ArrayList<HashMap<String, String>>> gattCharacteristicData
            = new ArrayList<ArrayList<HashMap<String, String>>>();
    mGattCharacteristics = new ArrayList<ArrayList<BluetoothGattCharacteristic>>();
```

```
    for (BluetoothGattService gattService : gattServices) {
        HashMap<String, String> currentServiceData = new HashMap<String, String>();
        uuid = gattService.getUuid().toString();
        currentServiceData.put(LIST_NAME, SampleGattAttributes.
                            lookup(uuid, unknownServiceString));
        currentServiceData.put(LIST_UUID, uuid);
        gattServiceData.add(currentServiceData);

        ArrayList<HashMap<String, String>> gattCharacteristicGroupData =
                    new ArrayList<HashMap<String, String>>();
        List<BluetoothGattCharacteristic> gattCharacteristics =
                    gattService.getCharacteristics();
        ArrayList<BluetoothGattCharacteristic> charas =
                    new ArrayList<BluetoothGattCharacteristic>();
        for (BluetoothGattCharacteristic gattCharacteristic : gattCharacteristics) {
                charas.add(gattCharacteristic);
            HashMap<String, String> currentCharaData = new HashMap<String, String>();
            uuid = gattCharacteristic.getUuid().toString();
            currentCharaData.put(LIST_NAME, SampleGattAttributes.lookup(uuid,
                                unknownCharaString));
            currentCharaData.put(LIST_UUID, uuid);
            gattCharacteristicGroupData.add(currentCharaData);
        }
        mGattCharacteristics.add(charas);
        gattCharacteristicData.add(gattCharacteristicGroupData);
    }
}
```

다음은 GATT로부터 특성 변경에 대한 통지를 받기위해 장치 알림을 설정하는 코드이다.

```
public void setCharacteristicNotification(BluetoothGattCharacteristic characteristic, boolean
enabled) {
    if (mBluetoothAdapter == null || mBluetoothGatt == null) {
        return;
    }
    mBluetoothGatt.setCharacteristicNotification(characteristic, enabled);

    BluetoothGattDescriptor descriptor = characteristic.getDescriptor(UUID.fromString(
                        SampleGattAttributes.CLIENT_CHARACTERISTIC_CONFIG));
    descriptor.setValue(BluetoothGattDescriptor.ENABLE_NOTIFICATION_VALUE);

    mBluetoothGatt.writeDescriptor(descriptor);
}
```

GATT로부터 통지를 수신(BluetoothGattCallback의 onCharacteristicChanged()호출됨)한다. Activity로 sendBroadcast하여 Activity의 BroadcastReceiver가 해당 통지를 받아 필요한 처리를 하면 된다.

```
@Override
public void onCharacteristicChanged(BluetoothGatt gatt,
BluetoothGattCharacteristic characteristic) {
    broadcastUpdate(ACTION_DATA_AVAILABLE, characteristic);
}
```

앱을 종료할땐 반드시 close()를 호출한다.

```
public void close() {
    if (mBluetoothGatt == null) {
        return;
    }
    mBluetoothGatt.close();
    mBluetoothGatt = null;
}
```