
Chapter 243. 블루투스 프로그래밍

1. 블루투스

안드로이드 플랫폼에는 블루투스 연결에 대한 지원이 포함되어 있어 손쉽게 다른 블루투스 장치와의 무선 데이터 교환이 가능하다. 블루투스 연결을 사용하기 위해 어플리케이션은 안드로이드의 블루투스 API를 사용하여야 하며 블루투스 API를 통해 안드로이드 응용 프로그램은 다음의 네 가지 작업을 수행할 수 있다.

- 블루투스 설정
- 페어링 되었거나 페어링 되지 않은 주변 블루투스 장치 검색
- 다른 블루투스 장치와의 연결
- 장치 간 데이터 전송

다른 블루투스 장치와 데이터를 주고받기 위해서는 일반적으로 위의 네 가지 작업을 순서대로 시행하여야 한다. 먼저 안드로이드 폰의 블루투스 장치를 사용 가능하도록 설정하고 주변의 블루투스 장치를 검색하여 검색된 장치의 정보를 등록하여야 한다. 검색된 장치의 등록 절차를 페어링이라고 하며 장치의 전원을 끄더라도 페어링 정보는 남아 있으므로 페어링이 이루어진 장치는 이후 다시 페어링하지 않아도 된다. 블루투스에서는 페어링 된 장치와의 연결만이 가능하며 연결된 이후 데이터 송수신이 가능하다.

먼저 블루투스 테스트를 위한 프로젝트를 생성해 보자. 프로젝트의 이름은 'BluetoothTest'로, 도메인은 'gyeongyong.dongueui.edu'로 설정한다. 다른 옵션들은 디폴트값을 사용하였다.

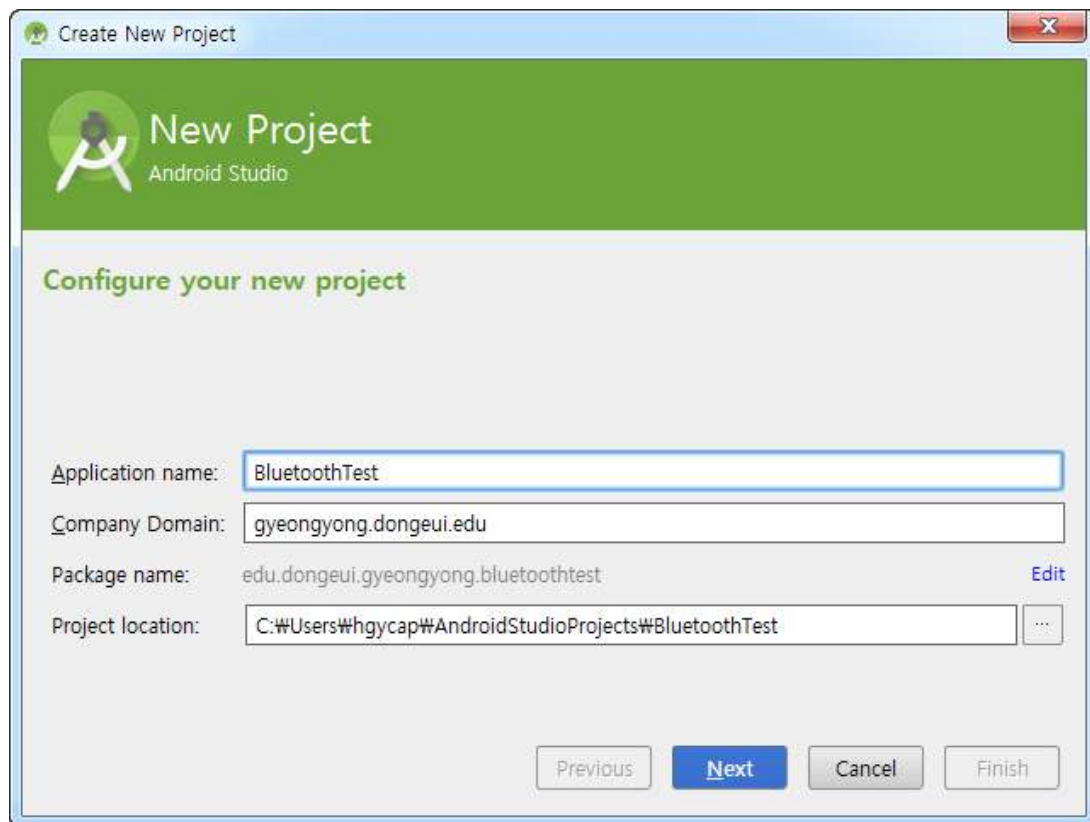


그림 1. 블루투스 테스트를 위한 프로젝트 생성

2. 블루투스 퍼미션

안드로이드 어플리케이션에서 블루투스를 사용하기 위해서는 BLUETOOTH 퍼미션(permission)을 선언해 주어야 한다. 퍼미션은 특정 기능을 사용할 수 있도록 하는 '승인' 또는 특정 기능을 사용할 수 있는 '권한'을 의미하며, 어플리케이션은 운영체제로부터 퍼미션을 얻어야만 특정 기능을 사용할 수 있다. BLUETOOTH 퍼미션은 블루투스 통신을 위한 연결 요청, 연결 수락, 데이터 전송 등을 위해 필요하다. 어플리케이션에서 주변 장치를 검색하거나 블루투스 설정을 바꿀 수 있도록 하려면 BLUETOOTH_ADMIN 퍼미션 역시 필요하며 BLUETOOTH_ADMIN 퍼미션을 얻기 위해서는 BLUETOOTH 퍼미션 역시 필요하다. 아두이노와의 통신을 위해서는 블루투스의 사용 여부를 바꿀 수 있어야 하므로 두 가지 퍼미션을 모두 선언해 준어야 한다. 퍼미션은 'AndroidManifest.xml' 파일에서 선언하며 AndroidManifest.xml 파일은 프로젝트의 'manifests' 디렉터리 아래에서 찾을 수 있다.

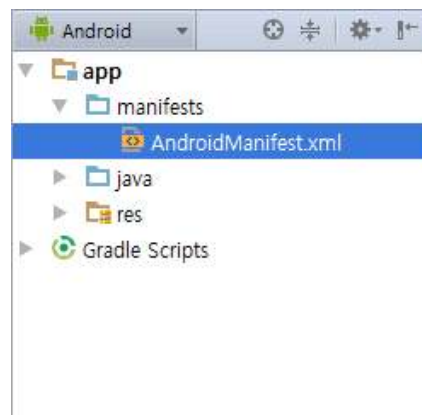


그림 2. AndroidManifest.xml 파일 위치

AndroidManifest.xml 파일은 모든 어플리케이션이 반드시 가지고 있어야 하는 파일로 어플리케이션에 대한 필수적인 정보를 안드로이드 플랫폼에 알려주기 위해 사용된다. 레이아웃을 정의하는 activity_main.xml 파일이 XML 형식으로 작성되는 것과 마찬가지로 AndroidManifest.xml 파일도 XML 형식으로 계층적으로 구성된다. AndroidManifest.xml의 최상위 요소 'manifest' 아래에 블루투스 사용을 위해 uses-permission 항목을 추가하자.

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

3. 블루투스 설정

어플리케이션이 블루투스로 통신을 수행하기 위해서는 장치가 블루투스를 지원하여야 하며, 블루투스를 지원하는 경우에는 블루투스 기능을 사용할 수 있도록 설정되어 있는지 확인하여야 한다. 최근 출시된 안드로이드 운영체제를 사용하는 스마트폰은 모두 블루투스를 지원하므로 블루투스 지원 여부를 걱정할 필요는 없다. 따라서 블루투스 장치의 상태를 검사하여 비활성 (disabled) 상태이면 이를 활성화 (enable) 상태로 어플리케이션 내에서 전환하는 방법에 대해 알아보자. 블루투스 설정은 BluetoothAdapter를 통해 두 단계로 이루어진다.

(1) 블루투스 어댑터 얻기

모든 블루투스 관련 작업은 BluetoothAdapter를 통해 이루어진다. BluetoothAdapter를 얻기 위해서는 정적 함수인 getDefaultAdapter를 사용하면 되고 getDefaultAdapter 함수는 장치의 블루투스 어댑터를 반환한다. 장치가 블루투스를 지원하지 않는 경우에는 getDefaultAdapter 함수는 null을 반환한다.

```
BluetoothAdapter mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
if(mBluetoothAdapter == null){
    // 장치가 블루투스를 지원하지 않는 경우
}
else {
    // 장치가 블루투스를 지원하는 경우
}
```

(2) 블루투스 활성화

다음은 블루투스 장치가 활성 상태인지 검사하여야 한다. 활성 상태를 검사하기 위해서는 블루투스 어댑터 객체의 isEnabled 함수를 사용할 수 있으며 활성 상태인 경우 true를, 비활성 상태인 경우 false를 반환한다. 블루투스가 비활성 상태인 경우 블루투스를 활성 상태로 변경하기 위해서는 BluetoothAdapter의 ACTION_REQUEST_ENABLE 인텐트로 startActivityForResult 함수를 호출하면 된다. 이 때 REQUEST_ENABLE_BT는 사용자 정의 상수로 블루투스 활성 상태의 변경 결과를 어플리케이션으로 알려줄 때 식별자로 사용되며 0보다 큰 값으로 정의하여야 한다.

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

startActivityForResult 함수가 호출되면 블루투스를 활성 상태로 변경하기 위해 사용자의 동의를 구하는 다이얼로그가 출력된다. 코드 1은 MainActivity 클래스의 멤버 함수로 블루투스

어댑터를 얻고, 블루투스 어댑터를 이용하여 블루투스의 활성 상태를 검사한 후, 비활성 상태이면 활성화를 위한 다이얼로그를 출력하는 함수의 예이다. 이 때 MainActivity 클래스의 멤버함수로 mBluetoothAdapter 변수가 선언되어 있어야 하며, REQUEST_ENABLE_BT 상수 역시 정의되어 있어야 한다.

```
static final int REQUEST_ENABLE_BT = 10;
BluetoothAdapter mBluetoothAdapter;
```

작성한 함수는 MainActivity 클래스의 onCreate 함수에서 호출하면 된다.

코드 1. 스마트폰의 블루투스 지원 여부 검사

```
static final int REQUEST_ENABLE_BT = 10;
BluetoothAdapter mBluetoothAdapter;

void checkBluetooth(){
    mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
    if(mBluetoothAdapter == null){
        // 장치가 블루투스를 지원하지 않는 경우
        finish(); // 어플리케이션 종료
    }
    else {
        // 장치가 블루투스를 지원하는 경우
        if (!mBluetoothAdapter.isEnabled()) {
            // 블루투스를 지원하지만 비활성 상태인 경우
            // 블루투스를 활성 상태로 바꾸기 위해 사용자 동의 요청
            Intent enableBtIntent =
                new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
            startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
        }
        else {
            // 블루투스를 지원하며 활성 상태인 경우
            // 페어링 된 기기 목록을 보여주고 연결할 장치를 선택
```

```
        selectDevice();
    }
}
```

startActivityResult 함수가 호출되면 그림 3과 같은 다이얼로그가 나타난다. “예”를 선택하면 시스템의 블루투스 장치를 활성화시키고 “아니오”를 선택하면 비활성화 상태를 유지한다.

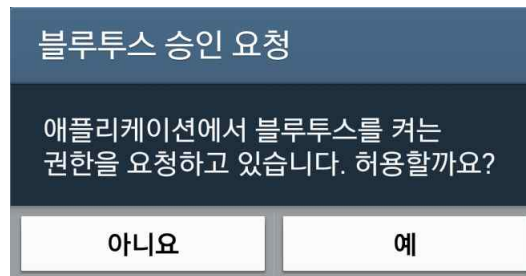


그림 3. 블루투스 활성화 다이얼로그

사용자의 선택 결과는 onActivityResult 콜백 함수에서 확인할 수 있다. 블루투스가 활성화 상태로 변경된 경우 RESULT_OK를, 오류나 사용자의 “아니오” 선택으로 블루투스가 비활성 상태로 남아있는 경우 RESULT_CANCELED 값이 onActivityResult 콜백 함수의 인자로 전달된다. ‘Code → Override Methods...’ 메뉴를 선택하면 오버라이드(override) 가능한 함수 목록이 나타난다. 이 중 onActivityResult를 선택하고 ‘OK’를 누르면 함수가 자동으로 생성된다.

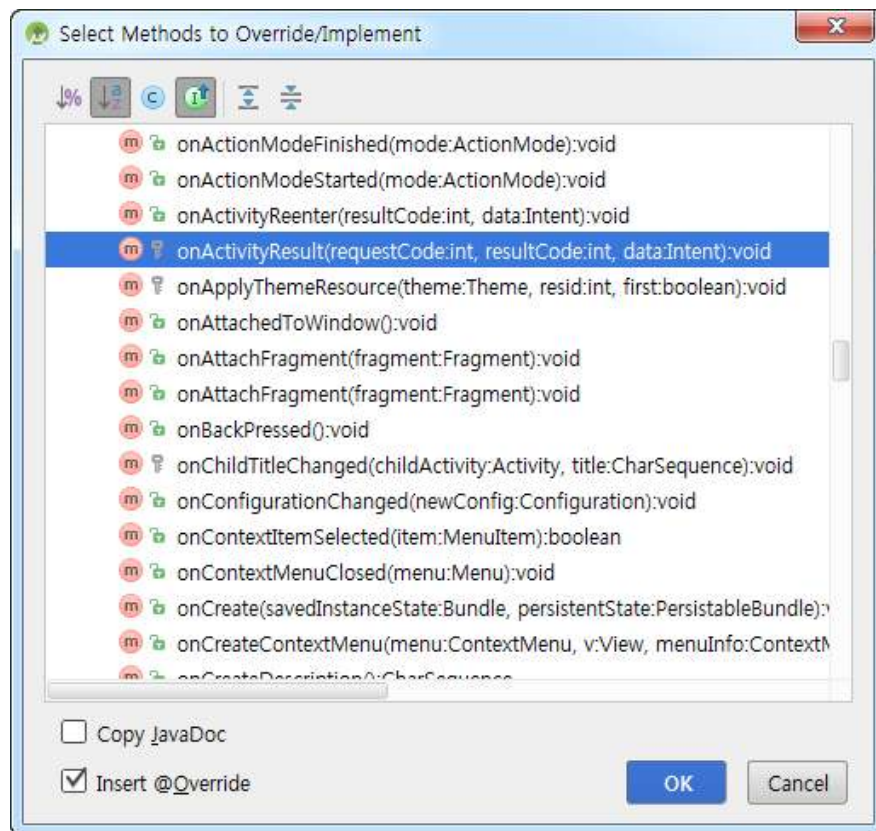


그림 4. 오버라이드 가능한 함수 목록

함수의 인자 중 requestCode에는 블루투스 활성화를 요구하기 위해 사용한 상수 값인 REQUEST_ENABLE_BT가, resultCode에는 활성 상태 변경 결과인 RESULT_OK 또는 RESULT_CANCELED 중 하나의 값이 전달된다. 코드 2는 블루투스가 승인 요청에서 '아니오'를 선택하여 블루투스가 비활성 상태인 경우 프로그램을 종료하도록 하는 코드이다.

코드 2. 블루투스가 비활성 상태인 경우 처리

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    switch(requestCode){
        case REQUEST_ENABLE_BT:
            if(resultCode == RESULT_OK){
                // 블루투스가 활성 상태로 변경됨
                selectDevice();
            }
            else if(resultCode == RESULT_CANCELED){
```

```
// 블루투스가 비활성 상태임
finish(); // 어플리케이션 종료
}
break;
}
super.onActivityResult(requestCode, resultCode, data);
}
```

4. 블루투스 장치 찾기

연결하고자 하는 블루투스 장치를 찾는 방법은 발견(discovery) 과정을 통해 주변의 블루투스 장치를 검색하고 검색된 장치와 페어링 과정을 거쳐 가능하다. 발견 과정은 발견 가능하도록(discoverable) 설정된 블루투스 장치를 검색하여 장치의 이름, 클래스, MAC 주소 등을 얻어 오는 과정을 말한다. 발견된 장치와 처음으로 연결을 시도하면 보안을 위해 핀(PIN)을 입력하도록 하고 있으며, 한 번 페어링이 된 기기는 스마트폰에서 관리되고 있으므로 페어링을 해제하기 전까지 핀을 다시 입력할 필요는 없다. 페어링이 이루어진 이후에는 실제로 데이터를 주고받기 위한 연결이 진행될 수 있다. 핀은 Passcode, Pairing Code, Passkey 등으로도 불린다.

이 장에서는 어플리케이션을 가능한 간단하게 구성하기 위해 발견 과정과 페어링 과정을 구현하지 않으며 연결하고자 하는 장치는 이미 페어링이 이루어져 있다고 가정한다. 페어링이 이루어지지 않은 새로운 장치의 경우에는 안드로이드 폰의 환경설정을 통해 페어링을 먼저 수행하여야 한다. 안드로이드 폰에서 '환경설정 → 블루투스'를 선택해 보자. 스마트폰은 자동으로 사용할 수 있는 블루투스를 찾아서 목록을 보여준다. 목록을 보여주지 않는다면 아래쪽의 '찾기' 버튼을 누르면 블루투스 장치를 검색할 수 있다.

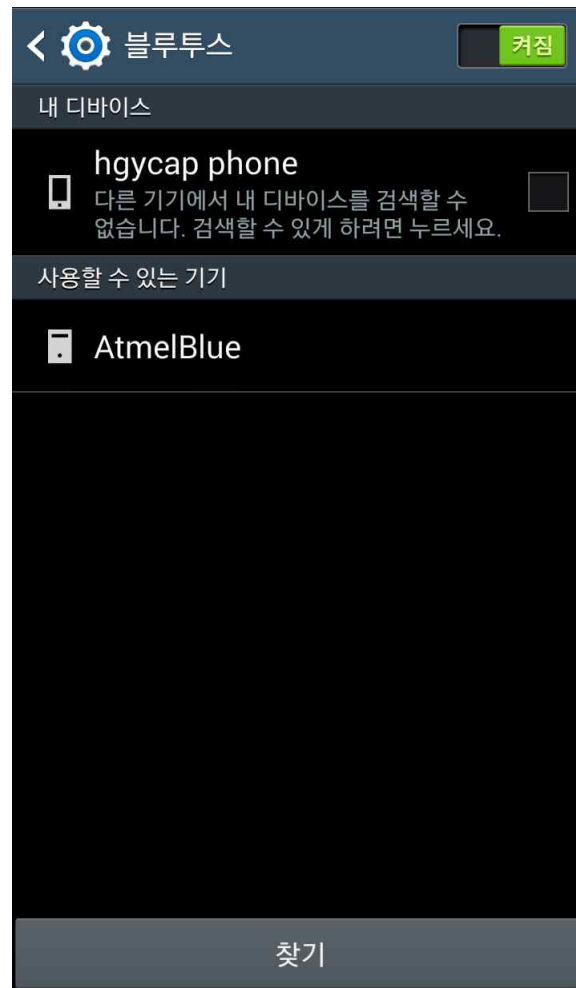
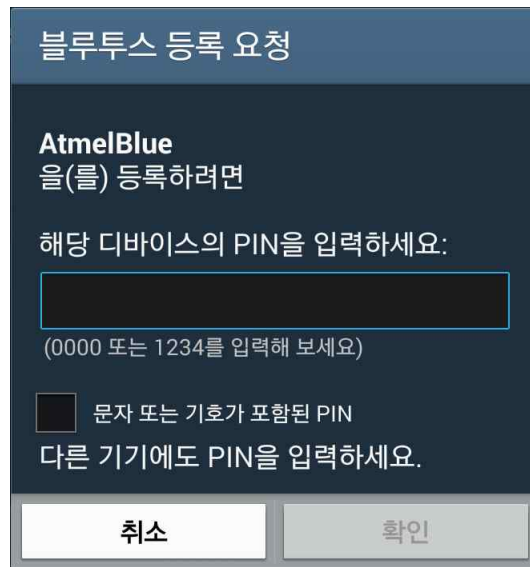


그림 5. 블루투스 설정 화면

‘사용할 수 있는 기기’에서 블루투스 모듈을 선택하면 핀을 입력하는 창이 출력된다.



블루투스 등록 요청

AtmelBlue
을(를) 등록하려면

해당 디바이스의 PIN을 입력하세요:

(0000 또는 1234를 입력해 보세요)

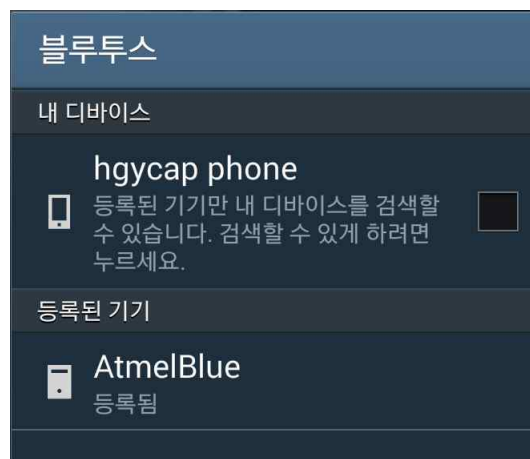
☐ 문자 또는 기호가 포함된 PIN

다른 기기에도 PIN을 입력하세요.

취소 확인

그림 6. 페어링을 위한 핀 입력

블루투스 모듈에 설정된 핀을 입력하면 페어링 과정은 끝나고 블루투스 모듈은 '등록된 기기'에 표시된다.



블루투스

내 디바이스

hgycap phone
등록된 기기만 내 디바이스를 검색할 수 있습니다. 검색할 수 있게 하려면 누르세요.

등록된 기기

AtmelBlue
등록됨

그림 7. 블루투스 모듈 페어링 완료

블루투스 모듈과의 페어링이 완료되었으므로 이제 실제 블루투스 모듈과의 연결 방법을 알아보자.

5. 연결할 장치 선택

연결하고자 하는 블루투스 장치는 페어링이 이루어져 있다고 가정하였으므로 장치를 선택하기 위해서는 먼저 페어링 된 장치들의 목록을 가져와야 한다. 페어링 된 장치 목록은 블루투스 어댑터의 `getBondedDevices` 함수를 사용하여 알아낼 수 있다.

```
Set<BluetoothDevice> pairedDevices = mBluetoothAdapter.getBondedDevices();
if (pairedDevices.size() > 0) {
    // 페어링 된 장치가 있는 경우
}
else {
    // 페어링 된 장치가 없는 경우
}
```

코드 3은 `AlertDialog`를 이용하여 페어링 된 장치의 목록을 보여주고 이 중 연결할 장치를 선택하는 `selectDevice` 함수를 구현한 예이다. 이 때 페어링이 되고 등록되어 연결할 수 있는 장치를 나타내는 `BluetoothDevice` 클래스들의 모음을 나타내기 위해 `BluetoothDevice` 클래스의 집합(`Set`) 형식의 멤버 변수인 `mDevices`와 페어링 된 장치의 수를 나타내는 `mPairedDeviceCount` 역시 선언해 주어야 한다.

```
int mPairedDeviceCount = 0;
Set<BluetoothDevice> mDevices;
```

페어링 된 장치가 없거나 연결할 장치를 선택하지 않은 경우에는 어플리케이션을 종료하도록 하였으며 뒤로 가기 버튼은 사용할 수 없도록 하였다. `selectDevice` 함수는 `checkBluetooth` 함수에서 현재 블루투스가 활성 상태인 경우와 `onActivityResult` 함수에서 사용자가 비활성 상태에서 활성 상태로 전환한 두 경우에 모두 호출해 주어야 한다.

코드 3

```
void selectDevice(){
    mDevices = mBluetoothAdapter.getBondedDevices();
```

```
mPairedDeviceCount = mDevices.size();

if(mPairedDeviceCount == 0){
    // 페어링 된 장치가 없는 경우
    finish();    // 어플리케이션 종료
}

AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setTitle("블루투스 장치 선택");

// 페어링 된 블루투스 장치의 이름 목록 작성
List<String> listItems = new ArrayList<String>();
for (BluetoothDevice device : mDevices) {
    listItems.add(device.getName());
}
listItems.add("취소");    // 취소 항목 추가

final CharSequence[] items =
    listItems.toArray(new CharSequence[listItems.size()]);

builder.setItems(items, new DialogInterface.OnClickListener(){
    public void onClick(DialogInterface dialog, int item){
        if(item == mPairedDeviceCount){
            // 연결할 장치를 선택하지 않고 '취소'를 누른 경우
            finish();
        }
        else{
            // 연결할 장치를 선택한 경우
            // 선택한 장치와 연결을 시도함
            connectToSelectedDevice(items[item].toString());
        }
    }
});

builder.setCancelable(false);    // 뒤로 가기 버튼 사용 금지
AlertDialog alert = builder.create();
```

```

    alert.show();
}

```

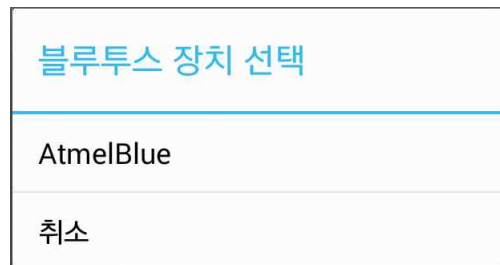


그림 8. 페어링된 블루투스 장치 목록 및 장치 선택

6. 장치 연결

연결하고자 하는 장치를 페어링 된 목록에서 선택하였으므로 이제 선택한 장치와 실제 통신이 가능하도록 연결하여야 한다. 두 장치가 연결되었다는 것은 동일한 RFCOMM 채널에 연결된 블루투스 소켓을 가지고 있으며 소켓을 통해 데이터 송수신이 가능하다는 의미이다.

다이얼로그에서 블루투스 장치의 이름을 선택하면 먼저 선택한 이름에 해당하는 BluetoothDevice 객체를 페어링 된 기기 목록에서 얻어온다. getBondedDevices 함수가 반환하는 페어링 된 기기 목록은 Set 형식이며 Set 형식에서는 n 번째 원소를 얻어오는 방법이 없으므로 주어진 이름과 비교하여 찾아야 한다. 코드 4는 블루투스 장치의 이름이 주어졌을 때 해당하는 블루투스 장치 객체를 페어링 된 장치 목록에서 찾아내는 코드의 예이다.

코드 4. 페어링된 블루투스 장치를 이름으로 찾기

```

BluetoothDevice getDeviceFromBondedList(String name){
    BluetoothDevice selectedDevice = null;

    for (BluetoothDevice device : mDevices) {
        if(name.equals(device.getName())){
            selectedDevice = device;
            break;
        }
    }
}

```

```

    }
    return selectedDevice;
}

```

BluetoothDevice 클래스로 표현되는 객체는 원격 블루투스 기기를 나타내며 createRfcommSocketToServiceRecord 함수를 사용하여 원격 블루투스 장치와 통신할 수 있는 소켓을 생성할 수 있다. createRfcommSocketToServiceRecord 함수의 매개변수로 주어지는 UUID(Universally Unique Identifier)는 SPP(Serial Port Profile)에 사용되는 UUID인 "00001101-0000-1000-8000-00805F9B34FB"를 사용하면 된다. 소켓이 생성되면 소켓의 connect 함수를 호출함으로써 두 기기의 연결은 완료된다. 실제 데이터 송수신을 위해서는 소켓으로부터 입출력 스트림을 얻고 입출력 스트림을 이용하여 이루어진다. 원격 블루투스 장치와 연결을 위해서는 연결 대상이 되는 블루투스 장치를 나타내는 멤버변수인 mRemoteDevice, 원격 장치와 연결에 사용할 소켓을 나타내는 멤버변수인 mSocket이 선언되어 있어야 한다. 또한 실제 데이터 송수신을 위한 입력 및 출력 스트림도 멤버변수로 선언되어 있어야 한다.

```

BluetoothDevice mRemoteDevice;
BluetoothSocket mSocket = null;
OutputStream mOutputStream = null;
InputStream mInputStream = null;

```

코드 5는 원격 장치와 연결하는 과정을 나타낸 예이다.

코드 5. 원격 블루투스 장치와의 연결

```

void connectToSelectedDevice(String selectedDeviceName){
    mRemoteDevice = getDeviceFromBondedList(selectedDeviceName);
    UUID uuid = UUID.fromString("00001101-0000-1000-8000-00805f9b34fb");

    try{

```

```

// 소켓 생성
mSocket = mRemoteDevice.createRfcommSocketToServiceRecord(uuid);
// RFCOMM 채널을 통한 연결
mSocket.connect();

// 데이터 송수신을 위한 스트림 얻기
mOutputStream = mSocket.getOutputStream();
mInputStream = mSocket.getInputStream();

// 데이터 수신 준비
beginListenForData();
}catch(Exception e){
    // 블루투스 연결 중 오류 발생
    finish();    // 어플리케이션 종료
}
}

```

더 이상 블루투스 연결이 필요하지 않은 경우에는 입출력 스트림과 소켓을 닫아주어야 한다. 'Code → Override Methods...' 메뉴에서 onDestroy 함수를 선택한다. onDestroy 함수는 어플리케이션이 종료될 때 호출되는 함수로 이 함수에서 스트림과 소켓을 닫아준다. 더불어 데이터 수신을 위해 별도의 쓰레드를 사용하므로 프로그램 종료 시에는 쓰레드 역시 종료시켜 준다. mWorkerThread 변수는 어플리케이션의 동작과 별도로 블루투스를 통한 데이터 수신을 감시하는 쓰레드를 나타낸다.

```
Thread mWorkerThread = null;
```

코드 6. 블루투스 소켓 닫기 및 데이터 수신 쓰레드 종료

```

protected void onDestroy() {
    try{
        mWorkerThread.interrupt(); // 데이터 수신 쓰레드 종료
        mInputStream.close();
    }
}

```

```

        mOutputStream.close();
        mSocket.close();
    }catch(Exception e){}

    super.onDestroy();
}

```

7. 데이터 송수신

소켓을 열고 데이터 송수신을 위한 준비는 끝났으므로 실제 데이터 송수신 방법을 살펴보자. 데이터 송수신을 구현하는 과정에서 주의해야 할 점 중 한 가지는 데이터 송수신을 별도의 스레드로 구현해야 한다는 점이다. 이는 입력 스트림에서 데이터를 읽는 read 함수와 출력 스트림으로 데이터를 쓰는 write 함수는 데이터 송수신이 완료될 때까지 반환하지 않는 블로킹(blocking) 호출을 사용하기 때문이다. 일반적으로 write 함수는 데이터를 출력한 후 바로 반환하여 블록 되지 않지만 연결된 원격 블루투스 장치가 전송된 데이터를 읽지 않아 버퍼가 가득 찬 상태라면 흐름 제어를 위해 블록 될 수 있다. read 함수는 스트림에 읽을 수 있는 데이터가 준비될 때까지 블록 된다. 먼저 데이터 전송 함수를 만들어 보자. 전송 함수에서는 문자열의 끝을 표현하기 위해 '\n'을 사용한다.

```

String mStrDelimiter = "\n";
char mCharDelimiter = '\n';

```

코드 7은 문자열을 전송하는 함수의 예로 별도의 스레드로 작성하지는 않았다.

코드 7. 데이터 송신 함수

```

void sendData(String msg){
    msg += mStrDelimiter; // 문자열 종료 표시
    try{
        mOutputStream.write(msg.getBytes()); // 문자열 전송
    }
}

```



```

    }catch(Exception e){
        // 문자열 전송 도중 오류가 발생한 경우
        finish();    // 어플리케이션 종료
    }
}

```

코드 8은 메시지 수신 함수의 예로 메시지 수신을 위한 쓰레드를 생성하여 수신 메시지를 계속 검사한다. 메시지 수신 쓰레드에서 수신된 문자 중 문자열 종료 표시가 발견되면 새로운 쓰레드를 생성하여 수신된 문자열에 대한 처리를 진행하도록 한다. 이 때 수신된 문자열을 저장하고 저장된 위치를 기억하기 위해 멤버 변수를 정의하여야 한다.

```

byte[] readBuffer;
int readBufferPosition;

```

코드 8은 메시지 수신 및 처리 함수의 예이다.

코드 8. 데이터 수신 준비 및 처리

```

void beginListenForData()
{
    final Handler handler = new Handler();

    readBuffer = new byte[1024];    // 수신 버퍼
    readBufferPosition = 0;        // 버퍼 내 수신 문자 저장 위치

    // 문자열 수신 쓰레드
    mWorkerThread = new Thread(new Runnable(){
        public void run(){
            while(!Thread.currentThread().isInterrupted()){
                try {
                    int bytesAvailable = mInputStream.available();    // 수신 데이터 확인
                    if(bytesAvailable > 0){                            // 데이터가 수신된 경우

```

```
byte[] packetBytes = new byte[bytesAvailable];
mInputStream.read(packetBytes);
for(int i = 0; i < bytesAvailable; i++){
    byte b = packetBytes[i];
    if(b == mDelimiter){
        byte[] encodedBytes = new byte[readBufferPosition];
        System.arraycopy(readBuffer, 0,
            encodedBytes, 0, encodedBytes.length);
        final String data = new String(encodedBytes, "US-ASCII");
        readBufferPosition = 0;

        handler.post(new Runnable(){
            public void run(){
                // 수신된 문자열 데이터에 대한 처리 작업
            }
        });
    }
    else{
        readBuffer[readBufferPosition++] = b;
    }
}
}
catch (IOException ex){
    // 데이터 수신 중 오류 발생
    finish();
}
}
});

mWorkerThread.start();
}
```

문자열을 송수신하는 방법까지 알아보았다. 실제 송수신된 문자열에 대한 의미는 어플리케이션에 따라 달라지므로 어플리케이션에 맞게 처리하여야 한다.

8. 데이터 송수신 실험

이제 아두이노와 데이터 송수신을 해보자. 먼저 아두이노의 디지털 2번과 3번 핀에 TX와 RX를 각각 연결한다.

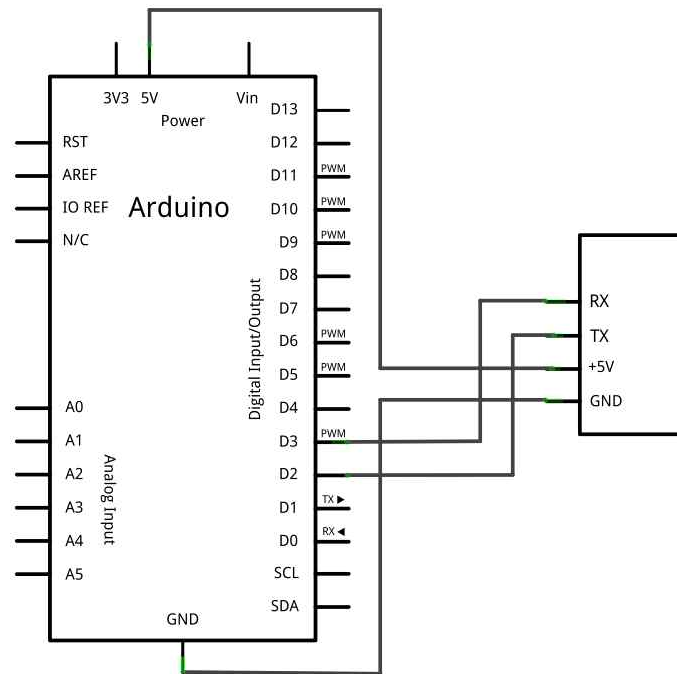


그림 9. 아두이노와 블루투스 모듈 연결

아두이노에는 코드 9를 업로드 한다. 코드 9는 블루투스를 통해 문자열을 전송 받고 문자열 종료 표시가 발견된 수신된 문자열을 다시 블루투스를 통해 전송하는 예이다. 송신과 수신에서 모두 문자열의 종료 표시로 개행문자인 ‘\n’을 사용하고 있다.

코드 9

```
#include <SoftwareSerial.h>

SoftwareSerial BTSerial(2, 3);    // SoftwareSerial(RX, TX)

byte buffer[1024];                // 데이터 수신 버퍼
```

```
int bufferPosition;           // 버퍼에 기록할 위치

void setup()
{
    BTSerial.begin(9600);      // 블루투스 모듈 초기화
    Serial.begin(9600);        // 시리얼 모니터 초기화
    bufferPosition = 0;
}

void loop()
{
    if (BTSerial.available()){ // 블루투스로 데이터 수신
        byte data = BTSerial.read();
        Serial.write(data);    // 수신된 데이터 시리얼 모니터로 출력
        buffer[bufferPosition++] = data;
        if(data == '\n'){      // 문자열 종료 표시
            buffer[bufferPosition] = '\0';

            // 스마트폰으로 전송할 문자열을 시리얼 모니터에 출력
            Serial.print("Echo Back : ");
            Serial.write(buffer, bufferPosition);

            // 스마트폰으로 문자열 전송
            BTSerial.write(buffer, bufferPosition);

            bufferPosition = 0;
        }
    }
}
```

어플리케이션에는 전송할 문자열 입력을 위한 텍스트 박스와 수신된 문자열을 표시할 텍스트 박스 그리고 전송 버튼을 넣는다. 화면 배치는 relative 레이아웃을 사용하며 문자열 입력 텍스트 박스를 좌상단에, 전송 버튼은 문자열 입력 박스의 오른쪽에, 그리고 출력 텍스트 박스는 아래쪽에 화면에 딱 차게 표시한다. 코드 10은 그림 10의 화면 배치를 위한 'activity_main.xml' 파일이다.

코드 10. activity_main.xml

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/sendText"
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true"
        android:layout_alignBottom="@+id/sendButton"
        android:layout_toLeftOf="@+id/sendButton" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="전송"
        android:id="@+id/sendButton"
        android:layout_alignParentTop="true"
        android:layout_alignParentRight="true" />

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/receiveText"
        android:layout_below="@+id/sendButton"
```

```
android:layout_alignParentLeft="true"
android:layout_alignRight="@+id/sendButton"
android:layout_alignParentBottom="true"
android:gravity="top" />
</RelativeLayout>
```

코드 10에 의한 화면 배치는 그림 10과 같이 위쪽에 문자열 입력을 위한 텍스트 박스와 전송 버튼이 위치하고 아래쪽에는 수신된 문자열을 표시할 텍스트 박스가 위치한다.



그림 10. 화면 배치

문자열이 수신되면 데이터 수신 쓰레드에서 수신된 문자열을 텍스트박스에 나타내고 이는 코드 8에서 처리한다. 데이터 송신 버튼이 눌러진 경우 처리는 MainActivity 클래스가 OnClickListener를 구현하여 처리한다.

어플리케이션의 실행하고 문자열을 입력한 뒤 'Send' 버튼을 누르면 아두이노로 문자열이 전송되어 그림 11과 같이 시리얼 모니터에 수신된 문자열이 나타나고, 아두이노는 다시 스마트폰으로 동일한 내용을 전송하여 어플리케이션 상에 그림 12와 같이 나타나게 된다.



그림 11. 어플리케이션 실행 화면 - 아두이노

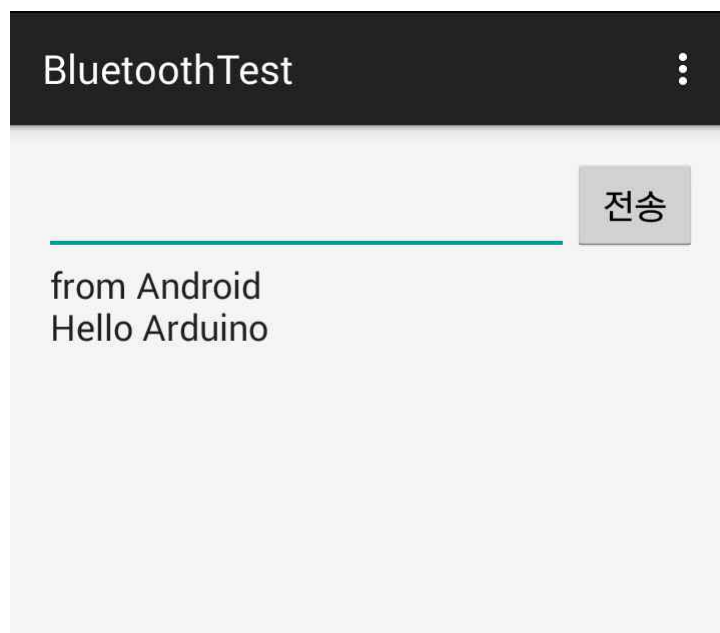


그림 12. 어플리케이션 실행 화면 - 스마트폰

프로그램의 전체 코드는 코드 11과 같다.

코드 11. 블루투스 테스트 코드

```
package edu.dongui.gyeongyong.bluetoothtest;

import android.app.AlertDialog;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.content.DialogInterface;
import android.content.Intent;
import android.os.Handler;
import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.Set;
import java.util.UUID;

public class MainActivity
    extends ActionBarActivity implements View.OnClickListener{

    static final int REQUEST_ENABLE_BT = 10;
    BluetoothAdapter mBluetoothAdapter;
    int mPairedDeviceCount = 0;
    Set<BluetoothDevice> mDevices;
    BluetoothDevice mRemoteDevice;
    BluetoothSocket mSocket = null;
    OutputStream mOutputStream = null;
```



```
InputStream mInputStream = null;

Thread mWorkerThread = null;
String mStrDelimiter = "\n";
char mCharDelimiter = '\n';
byte[] readBuffer;
int readBufferPosition;

EditText mEditReceive, mEditSend;

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    switch(requestCode){
        case REQUEST_ENABLE_BT:
            if(resultCode == RESULT_OK){
                // 블루투스가 활성화 상태로 변경됨
                selectDevice();
            }
            else if(resultCode == RESULT_CANCELED){
                // 블루투스가 비활성 상태임
                finish(); // 어플리케이션 종료
            }
            break;
    }
    super.onActivityResult(requestCode, resultCode, data);
}

void checkBluetooth(){
    mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
    if(mBluetoothAdapter == null){
        // 장치가 블루투스를 지원하지 않는 경우
        finish(); // 어플리케이션 종료
    }
    else {
        // 장치가 블루투스를 지원하는 경우
        if (!mBluetoothAdapter.isEnabled()) {
```

```
// 블루투스를 지원하지만 비활성 상태인 경우
// 블루투스를 활성 상태로 바꾸기 위해 사용자 동의 요청
Intent enableBtIntent =
    new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
}
else {
    // 블루투스를 지원하며 활성 상태인 경우
    // 페어링 된 기기 목록을 보여주고 연결할 장치를 선택
    selectDevice();
}
}
}

void selectDevice(){
    mDevices = mBluetoothAdapter.getBondedDevices();
    mPairedDeviceCount = mDevices.size();

    if(mPairedDeviceCount == 0){
        // 페어링 된 장치가 없는 경우
        finish();    // 어플리케이션 종료
    }

    AlertDialog.Builder builder = new AlertDialog.Builder(this);
    builder.setTitle("블루투스 장치 선택");

    // 페어링 된 블루투스 장치의 이름 목록 작성
    List<String> listItems = new ArrayList<String>();
    for (BluetoothDevice device : mDevices) {
        listItems.add(device.getName());
    }
    listItems.add("취소");    // 취소 항목 추가

    final CharSequence[] items =
        listItems.toArray(new CharSequence[listItems.size()]);
```

```
builder.setItems(items, new DialogInterface.OnClickListener(){
    public void onClick(DialogInterface dialog, int item){
        if(item == mPairedDeviceCount){
            // 연결할 장치를 선택하지 않고 '취소'를 누른 경우
            finish();
        }
        else{
            // 연결할 장치를 선택한 경우
            // 선택한 장치와 연결을 시도함
            connectToSelectedDevice(items[item].toString());
        }
    }
});

builder.setCancelable(false); // 뒤로 가기 버튼 사용 금지
AlertDialog alert = builder.create();
alert.show();
}

void beginListenForData() {
    final Handler handler = new Handler();

    readBuffer = new byte[1024]; // 수신 버퍼
    readBufferPosition = 0; // 버퍼 내 수신 문자 저장 위치

    // 문자열 수신 쓰레드
    mWorkerThread = new Thread(new Runnable(){
        public void run(){
            while(!Thread.currentThread().isInterrupted()){
                try {
                    int bytesAvailable = mInputStream.available(); // 수신 데이터 확인
                    if(bytesAvailable > 0){ // 데이터가 수신된 경우
                        byte[] packetBytes = new byte[bytesAvailable];
                        mInputStream.read(packetBytes);
                        for(int i = 0; i < bytesAvailable; i++){
                            byte b = packetBytes[i];
```

```
        if(b == mCharDelimiter){
            byte[] encodedBytes = new byte[readBufferPosition];
            System.arraycopy(readBuffer, 0,
                encodedBytes, 0, encodedBytes.length);
            final String data = new String(encodedBytes, "US-ASCII");
            readBufferPosition = 0;

            handler.post(new Runnable(){
                public void run(){
                // 수신된 문자열 데이터에 대한 처리 작업
                mEditReceive.setText(mEditReceive.getText().toString()
                    + data + mStrDelimiter);
                }
            });
        }
        else{
            readBuffer[readBufferPosition++] = b;
        }
    }
}
catch (IOException ex){
    // 데이터 수신 중 오류 발생
    finish();
}
}
});

mWorkerThread.start();
}

void sendData(String msg){
    msg += mStrDelimiter; // 문자열 종료 표시
    try{
        mOutputStream.write(msg.getBytes());
        // 문자열 전송
    }
```

```
    }catch(Exception e){
        // 문자열 전송 도중 오류가 발생한 경우
        finish();    // 어플리케이션 종료
    }
}

BluetoothDevice getDeviceFromBondedList(String name){
    BluetoothDevice selectedDevice = null;

    for (BluetoothDevice device : mDevices) {
        if(name.equals(device.getName())){
            selectedDevice = device;
            break;
        }
    }

    return selectedDevice;
}

@Override
protected void onDestroy() {
    try{
        mWorkerThread.interrupt(); // 데이터 수신 쓰레드 종료
        mInputStream.close();
        mOutputStream.close();
        mSocket.close();
    }catch(Exception e){}

    super.onDestroy();
}

void connectToSelectedDevice(String selectedDeviceName){
    mRemoteDevice = getDeviceFromBondedList(selectedDeviceName);
    UUID uuid = UUID.fromString("00001101-0000-1000-8000-00805f9b34fb");

    try{
```