**http://hyoin1223.tistory.com/entry/%EC%95%88%EB%93%9C%EB%A1%9C%EC%9D%B4%EB%93%9C-%EB%B8%94%EB%A3%A8%ED%88%AC%EC%8A%A4-%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%B0%8D**

**Chapter 61. 안드로이드 블루투스 프로그래밍**

**1.블루투스**

안드로이드 플랫폼은 블루투스 연결에 대한 지원이 포함되어 있어 손쉽게 다른 블루투스 장치와의 무선 데이터 교환이 가능함

블루투스 연결을 사용하기 위해 응용 프로그램은 안드로이드의 블루투스 API를 사용하여야 하며

블루투스 API 는 android.bluetooth 패키지에 포함되어 있다.

블루투스 API 를 이용 다음 4가지 작업을 수행 가능

   - 블루투스 설정.

   - 페어링 되었거나 페어링 되지 않은 주변 블루투스 장치 검색.

   - 다른 블루투스 장치와의 연결.

   - 장치 간 데이터 전송.

다른 블루투스 장치와 데이터 송수신을 위해서는 일반적으로 위의 네가지 작업을 순서대로 시행해야 한다.

먼저 안드로이드 블루투스 장치를 사용 가능하도록 설정,

주변의 블루투스 장치를 검색, 검색되니 장치의 정보를 등록.

**페어링**:  검색된 장치의 등록 절차.

          블루투스에서는 페어링 된 장치와의 연결만이 가능하며 연결된 이후 데이터 송수신이 가능하다.

----------------------------------------------------------------------------------------------------

**2. 블루투스 퍼미션.**

안드로이드 App 에서 블루투스 사용하기 위해서는 BLUETOOTH 퍼미션을 선언해 주어야함.

BLUETOOTH PERMISSION : 블루투스 통신을 위한 연결 요청, 연결 수락, 데이터 전송 등을 위해 필요함.

APP에서 주변 장치를 검색하거나 블루투스 설정을 바꿀 수 있도록 하려면 BLUETOOTH 퍼미션 역시 필요함.

**▶ AndroidManifest.xml 에 선언함.**

|  |  |  |
| --- | --- | --- |
| 1  2 | <uses-permission android:name="android.permission.BLUETOOTH"/>  <uses-permission android:name="android.permission.BLUETOOTH\_ADMIN"/> | [cs](http://colorscripter.com/info#e) |

----------------------------------------------------------------------------------------------------

**3. 블루투스 설정.**

App이 블루투스로 통신을 수행하기 위해서는 블루투스 지원하여야 하며,

블루투스를 지원하는 경우 블루투스 기능을 사용할 수 있도록 설정되어 있는지 확인해야 함.

최신 스마트폰은 거의 모두 블루투스를 지원하므로, 블루투스의 장치의 상태를 검사하여

비활성화(Disabled) 상태이면, 이를 활성(Enable) 상태로 App 내에서 전환하는 방법에 대해 알아보자.

블루투스 설정은 BluetoothAdapter를 통해 두 단계로 이루어 진다.

**(1) 블루투스 어댑터 얻기**

- 블루투스 관련 작업은 BluetoothAdapter를 통해 이루어 진다.

  BluetoothAdapter를 얻기 위해서는 정적 함수인 getDefaultAdapter를 사용하면 되고,

  getDefaultAdapter 함수는 장치의 블루투스 어댑터를 반환한다.

  장치가 블루투스를 지원하지 않는 경우에는 getDefaultAdapter 함수는 null을 반환함.

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9 | BluetoothAdapter mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();  if(mBluetoothadapter == null) {     //장치가 블루투스를 지원하지 않는 경우.  }    else {     // 장치가 블루투스를 지원하는 경우.  }    [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

**(2) 블루투스 활성화**

 - 블루투스 장치가 활성 상태인지 검사하여야 한다. 활성 상태를 검사하기 위해서는 블루투스 어댑터 객체의

   isEnabled 함수를 사용할 수 있으며  활성 상태인 경우 true를, 비활설 상태인 경우 false를 반환 한다.

   블루투스 비활성상태에서 활성 상태로 변경 : BluetoothAdapter의 ACTION\_REQUEST\_ENABLE 인텐트로

   startActivityForResult 함수를 호출하면 된다.

   이때 REQUEST\_ENABLE\_BT 는 사용자 정의 상수로 블루투스 활성 상태의 변경 결과를 App 으로 알려줄 때

   식별자로 사용되며 0 보다 큰 값으로 정의 해야 한다.

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5 | if(!mBluetoothAdapter.isEnabled()) {     Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION\_REQUEST\_ENABLE);     startActivityForResult(enabledBtIntent, REQUEST\_ENABLE\_BT);  }    [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

startActivityForResult 함수가 호출되면 블루투스를 활성화 상태로 변경하기 위해 사용자의 동의를 구하는

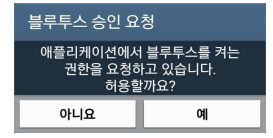
다이얼로그가 출력 된다.

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | void checkBlueTooth() {      mBluetoothAdapter = BlueAdapter.getDefaultAdapter();     if(mBluetoothAdapter == nulll) {        // 장치가 블루투스 지원하지 않는 경우        finish();   // 어플리케이션 종료     }       else {       // 장치가 블루투스 지원하는 경우      if(!mBluetoothAdapter.isEnabled()) {      // 블루투스를 지원하지만 비활성 상태인 경우      // 블루투스를 활성 상태로 바꾸기 위해 사용자 동의 요첨      Intent enable BtIntent = new Intent(BluetoothAdapter.ACTION\_REQUEST\_ENABLE);      startActivityForResult(enableBtIntent, REQUEST\_ENABLE\_BT);     }       else {     // 블루투스를 지원하며 활성 상태인 경우     // 페어링된 기기 목록을 보여주고 연결할 장치를 선택.      }    }  }  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

startActivityForResult 함수가 호출되면 아래와 같은 다이얼로그가 나타난다.

**예**     : 시스템의 블루투스 장치를 활성화 시키고

**아니오**:  비활성화 상태를 유지함



- 사용자의 선택 결과는 onActivityResult 콜백 함수에서 확인할 수 있다.

  RESULT\_OK       : 블루투스 활성화 상태로 변경된 경우.

  RESULT\_CANCELED : 오류나 사용자의 "아니오" 선택으로 블루투스가 비활성 상태로 남아 있는 경우.

                    RESULT\_CANCELED 값이 onActivityReulst 콜백 함수의 인자로 전달됨.

- Alt + Shift + S  →  Source → Override / Implement Methods 메뉴를 오버라이드(override)나 구현(implement)

  가능한  함수의 목록이 나타 난다.

  이중 onActivityResult를 선택하고 OK 를 누르면 함수가 자동으로 생성됨

함수의 인자중 requestCode에는 블루투스 활성화를 요구하기 위해 사용한 REQUEST\_ENABLE\_BT가,

resultCode에는 활성 상태 변경 결과인 RESULT\_OK 또는 RESULT\_CANCELED 중 하나의 값이 전달 됨.

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | protected void onActivityResult(int requestCode, int resultCode, Intent data) {    switch(requestCode) {     case REQUEST\_ENABLE\_BT:       if(resultCode == RESULT\_OK) {        // 블루투스가 활성 상태로 변경됨       }         else if(resultCode == RESULT\_CANCELDE) {        // 블루투스가 비활성 상태임        finish();  //  어플리케이션 종료          }       break;     }     super.onActivityResult(requestCode, resultCode, data);   }  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

**4. 블루투스 장치 찾기.**

연결하고자 하는 블루투스 장치를 찾는 방법은 발견(discovery) 과정을 통해 주변의 불루투스 장치를 검색하고,

검색된 장치와 페어링 과정을 거쳐 가능함.

발견 과정은  발견 가능하도록(discoverable) 설정된 블루툿 장치를 검색하여 장치의 이름, 클래스, MAC 주소등을 얻어 오는

과정으로 발견된 장치를 선택함으로써 연결을 시작할 수 있다.

발견된 장치와 처음 연결을 시도하념 사용자에게는 키를 입력하도록 요구하며 페어링 된 장치는 기기에 등록 되므로

이후 다시 발견 과정을 거치지 않고 바로 연결 할 수 있다.

페어링과 연결은 그 의미가 다른 점에 유의.

  - 페어링 : 두 블루투스 장치가 서로의 존재를 알고 있고 인증에 사용할 키를 공유하고 있어 암호화된 연결이 가능하다는 의미

  - 연결 :  두 블루투스 장치가 RFCOM (Radio Frequncy COMMunication) 채널을 공유하고 있어 데이터 송수신이 가능하다는 의미

안드로이드의 블루투스 API 에서는 연결이 이루어지기 이전에 반드시 페어링이 이루어져 있어야 한다.

**5. 연결할 장치 선택**

연결하고자 하는 블루투스 장치는 페어링이 이루어져 있다고 가정하였으므로 장치를 선택하기 위해서는 먼저

페이링 된 장치의 목록을 얻어와야 하며, 페어링된 장치 목록은 블루투스 어댑터의 getBondedDevices 함수를

사용하여 알아낼 수 있다.

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9 | set<BluetoothDevice> pairedDevices = mBluetoothAdapter.getBondedDevices();  if(pairedDevices.size() > 0) {   // 페어링 된 장치가 있는 경우.  }    else {    // 페어링 된 장치가 없는 경우.  }    [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

아래 코드는 AlertDialog를 이용하여 페어링 된 장치의 목록을 보여주고 이 중 연결할 장치를 선택하는

selectDevice 함수를 보여줌.

페어링된 장치가 없거나 연결할 장치를 선택하지 않은 경우에는 어플리케이션을 종료하도록 함.

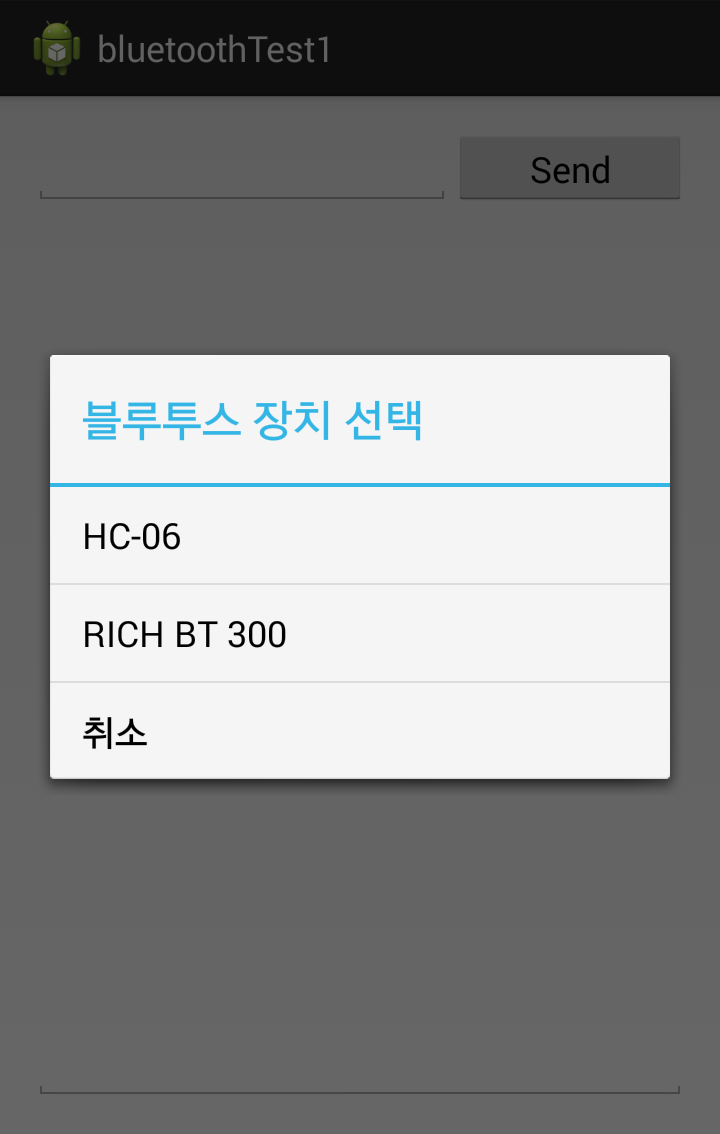
뒤로가기 버튼은 사용하지 못하도록 함.

selectDevice 함수는 checkBluetooth 함수에서 현재 블루투스가 활성 상태인 경우 onActivityResult 함수에서

사용자가 비활성 상태에서 활성 상태로 전환한 두 경우에 모두 호출해 주어야 함.

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41 | void selectDevice() {     mDevices = mBluetoothAdapter.getBondedDevices();     mPairedDeviceCount = mDevices.size();       if(mPairedDeviceCount == 0 ) {     //  페어링 된 장치가 없는 경우     finish();    // 어플리케이션 종료    }      AlertDialog.Buildier 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: mDevices) {   }   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     connectToSelectedDevices(items[item].toString());     }    }   });       builder.setCancelable(false);    // 뒤로 가기 버튼 사용 금지   AlertDialog alert = builder.create();   alert.show();  }  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

**※ 페어링된 블루투스 장치 목록 및 장치 선택**



**6. 장치 연결**

연결하고자 하는 장치를 페어링 된 목록에서 선택하였으므로 이제 선택한 장치와 실제 통신이 가능하도록 연결 해야 함.

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.

아래 코드는 블루투스 장치의 이름이 주어졌을때 해당하는 블루투스 장치 객체를 페어링 된 장치 목록에서 찾아내는 코드의 예이다.

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | BluetoothDevice getDeviceFromBondedList(String name) {     BluetoothDevice selectedDevice = null;       for(BluetoothDevice device : mDevices) {       if(name.equals(device.getName())) {        selectedDevice = devices;          break;          }       }       return selectDevice;     }  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

※ UUID : Universally Unique IDEntifier

※ SPP  : Serial Port Profile

**※ Bluetooth UUID**

**출처)**[**http://beomnaegol.tistory.com/entry/Bluetooth-UUID**](http://beomnaegol.tistory.com/entry/Bluetooth-UUID)

|  |
| --- |
| **- 블루투스의 UUID 란?**      범용 고유 번호(Universally Unique IDentifiers) 라고 불리며, 128 비트의 숫자들을 조합한다.      말 그대로 범용적으로 사용할 수 있는 고유의 ID를 사용하기 위해 생성되며, 그렇기 때문에 128 비트의 HEX 조합은      unique 하여야 한다. Bluetooth 에서는 device에서 제공하는 service를 검색하여 각 service 마다 UUID 를      부여하는 등 많은 부분에서 사용된다.    **- UUID 의 구성 요소**       UUID = (time\_low) - (time\_mid) - (time\_high\_and\_version) - (clock\_seq\_hi\_and\_reserved) -              (clock\_seq\_low\_node)       ex) 000011001-0000-1000-8000-00805F9B34FB           time\_low : 타임 스탬프(시간표시) 의 최하위 32 비트.      - 블루투스 device 끼리 통신을 하기 위해서 소켓 생성시 UUID 가 필요한데  이는 해당 프로토콜의 의미한다.      안드로이드 플랫폼의 단말기끼리는 어떤 UUID를 사용하여도 되지만 특정 프로토콜의 device에 접근하기 위해서는 각각의      프로토콜 UUID 를 사용하여만 한다. |

**※ Android 와 Bluetooth 모듈 SPP 통신**

**출처)**[**http://blog.naver.com/PostView.nhn?blogId=free2824&logNo=60108857440&parentCategoryNo=&cate**](http://blog.naver.com/PostView.nhn?blogId=free2824&logNo=60108857440&parentCategoryNo=&cate)

|  |
| --- |
| - **블루투스의 SPP 란?**      Serial Port Profile 이므로 블루투스로 연결해서 시리얼 포트처럼 사용할 수 있는 모듈이다.      외부기기에서 블루투스로 데이터를 전송. |

BluetoothDevice 클래스로 표현되는 객체는 원격 블루투스 기기르 나타내며,

CreateRfcommSocketToServiceRecord 함수를 사용하여 원격 블루투스 장치와 통신할 수 있는 소켓을 생성할 수 있다.

CreateRfcommSocketToServiceRecord 함수의 매개변수로 주어지는 UUID 는 SPP 에 사용되는 UUID 인

"000011001-0000-1000-8000-00805F9B34FB" 를 사용하면 된다.

소켓이 생성되면 소켓의 connect 함수를 호출함으로써 두기기의 연결은 완료된다.

실제 데이터 송수신을 위해서는 소켓으로부터 입출력 스트림을 얻고 입출력 스트림을 이용하여 이루어 짐.

아래 코드는 원격 장치와 연결하는 과정을 나타낸 예

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | void connectToSelectdDevice(String selectedDeviceName) {    mRemoteDevice = getDeviceFromBondedList(selectdDeviceName);    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 열기     mOutputStream = mSocket.getOutputStream();     mInputStream = mSocket.getInputStream();       // 데이터 수신 준비     beginListenForData();     }catch(Exception e) {      // 블루투스 연결 중 오류 발생      finish();   // 어플 종료     }  }  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

블루투스 연결이 필요하지 않은 경우는 입출력 스트림 소켓을 닫아주어야 한다.

onDestory 함수는 어플리케이션이 종료될 때 호출되는 함수로 이 함수에서 스트림과 소켓을 닫아준다.

더불어 데이터 수신을 위해 별도의 쓰ㅜ레드를 사용하므로 프로그램 종료 시에는 쓰레드 역시 종료 시켜준다.

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | protected void onDestory() {     try {        mWorkerThread.interrupt();   // 데이터 수신 쓰레드 종료        mInputStream.close();        mOutputStream.close();        mSocket.close();      } catch(Exception e) { }         super.onDestory();   }  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

**7. 데이터 송수신.**

소켓을 열고 데이터 송수신을 위한 준비는 끝났으므로 실제 데이터 송수신 방법을 살펴보자.

데이터 송수신 구현 과정에서 주의 할점 데이터 송수신을 별도의 쓰레드로 구현해야 한다는 점.

이는 입력 스트림에서 데이터를 읽는 read 함수와 출력 스트림으로 데이터를 쓰는 write함수는 데이터 송수신이 완료 될때까지 반환하지 않는 블록킹(blocking) 호출츨 사용하기 때문.

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.

데이터 전송 함수. 문자열 전송하는 함수의 예로 별도의 쓰레드로 작성하지 않았음.

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | void sendData(String msg) {     msg += mDelimiter;    // 문자열 종료 표시       try {       mOutputStream.write(msg.getBytes());    // 문자열 전송     } catch(Exception e) {       // 문자열 전송 도중 오류가 발생한 경우.       finish();    // APP 종료     }  }    [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

아래 코드는 메시지 수신 함수의 예로 수신을 위한 쓰레드 생성해서 수신 메시지를 계속 검사함.

메시지 수신 쓰레드에서 수신된 문자 중 문자열 종료 표시가 발견 되면 새로운 쓰레드를 생성하여

수신된 문자열에 대한 처리를 진행함.

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47 | void beginListenForData()  {     final Handle handler = new Handler();       readBuffer = new byte[1024] ;  //  수신 버퍼     readBufferPositon = 0;        // 버퍼 내 수신 문자 저장 위치       // 문자열 수신 쓰레드     mWorkerThread = new Thread(new Runnable() {        public void run() {          while(!Thread.currentThread().isInterrrupted()){             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();  }  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

**8. 데이터 송수신 실험.**

아두이노와 데이터 송수신.

먼저 아두이노의 디지털 2번과 3번 핀에 TX 와 RX 를 각각 연결한다.

아래 코드는 블루투스를 통해 문자열을 전송 받고 문자열 종료 표시가 발견된 경우 수신된 문자열을 다시 전송하는 예

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | #include <SoftwareSerial.h>      SoftwareSerial BTSerial(2, 3); // SoftwareSerial(RX, TX)    byte buffer[1024]; // 데이터를 수신 받을 버퍼    int bufferPosition; // 버퍼에 데이타를 저장할 때 기록할 위치    void setup(){    BTSerial.begin(9600);    Serial.begin(9600);    bufferPosition = 0; // 버퍼 위치 초기화  }    void loop(){    if (BTSerial.available()){ // 블루투스로 데이터 수신      byte data = BTSerial.read(); // 수신 받은 데이터 저장      Serial.write(data); // 수신된 데이터 시리얼 모니터로 출력      buffer[bufferPosition++] = data; // 수신 받은 데이터를 버퍼에 저장        if(data == '\n'){ // 문자열 종료 표시        buffer[bufferPosition] = '\0';          // 스마트폰으로 문자열 전송        BTSerial.write(buffer, bufferPosition);        bufferPosition = 0;      }    }  }  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

-  어플리케이션에는 전송할 문자열 입력을 위한 텍스트 박스와 수신된 문자열을 표시할 텍스트 박수

   그리고 전송 버튼을 넣는다.

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47 | <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:paddingBottom="@dimen/activity\_vertical\_margin"      android:paddingLeft="@dimen/activity\_horizontal\_margin"      android:paddingRight="@dimen/activity\_horizontal\_margin"      android:paddingTop="@dimen/activity\_vertical\_margin"      tools:context="com.acontech.bluetoothtest.MainActivity" >        <EditText          android:id="@+id/sendString"          android:layout\_width="wrap\_content"          android:layout\_height="wrap\_content"          android:layout\_alignParentLeft="true"          android:layout\_alignParentTop="true"          android:ems="10">          <requestFocus /> **// xml로 포커스 주는 방법**          </EditText>        <Button          android:id="@+id/sendButton"          android:layout\_width="match\_parent"          android:layout\_height="wrap\_content"          android:layout\_above="@+id/receiveString"          android:layout\_toRightOf="@+id/sendString"          android:text="Send"/>        <EditText          android:id="@+id/receiveString"          android:layout\_width="match\_parent"          android:layout\_height="match\_parent"          android:layout\_alignLeft="@+id/sendString"          android:layout\_alignRight="@+id/sendButton"          android:layout\_below="@+id/sendString"          android:layout\_marginTop="26dp"          android:ems="10"          android:focusable="false"          android:gravity="top"          android:inputType="textMultiLine"          />    </RelativeLayout>  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

**출처) 레이아웃 정리**

[**http://blog.naver.com/rkdwnsdud555/220284445103**](http://blog.naver.com/rkdwnsdud555/220284445103)

**1. RelativeLayout : 상대적 레이아웃. 다른 View들과의 상대적인 위치에 따라서 배치가 되는 레이아웃 이다.**

**2.  <requestFocus />  : xml 로 포커스를 주는 방법.**

**※ <requestFocus /> 이 여러개인 경우?**

**xml 은 순차적으로 읽기 때문에, 맨 마지막에 포커스가 간다.**

**3. android:layout\_alignParentLeft="true" : 부모 컨테이너의 왼쪽 부분에 view 를 배치 하겠다.**

**4. android:layout\_alignParentTop="true" : 부모 컨테이너 즉 상대 레이아웃의 윗 부분에 view의 윗부분을 배치 하겠다.**

**5. android:ems="10" : 너비(width)를 픽셀 단위 대신 em 단위로 설정할 수 있다. (1ems = 12pt)**

**6. android:focusable="false"  : 키보드 포커스를 받을 수 있는지 지정**

**7. android:layout\_gravity :  화면에서 콘트롤의 위치**

**android:gravity : 콘트롤 내의 텍스트의 위치를 정함.**

**8. android:inputType="textMultiLine" : 여러 줄을 입력할 수 있는 창.**

1. 문자열 입력을 위한 텍스트 박스

2. 전송 버튼

3. 아래쪽에는 수신된 문자열을 표시할 텍스트 박스.

문자열이 수신되면 데이터 수신 쓰레드에서 수신된 문자열을 텍스트박스에 나타내면 된다.

데이터 송신 버튼이 눌러진 경우 처리는 버튼에 onClickListener 를 추가하여 처리한다.

리스너 추가는 앱의 onCreate 함수에서 수행되며 이때 텍스트박스와 버튼에 대한 변수 할당도 이루어 진다.

어플리케이션의 실행하고 문자열을 입력한 뒤 'Send' 버큰을 누르면 아두이노로 문자열이 전송되어,

시리얼 모니터에 수시니된 문자열이 나타나고, 아두이노는 다시 스마트폰으로 동일한 내용을 전송하여

어플리케이션 상에 나타나게 된다.

[출처] <http://cafe.naver.com/arduinostory>