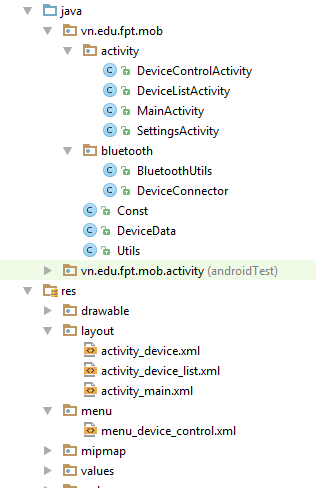
**Xử lý trên Android**

Để tiến hành xử lý giao tiếp Bluetooth giữa Android và Arduino cần một số lớp cho mục đích quản lý kết nối và lưu trữ dữ liệu truyền đi giữa hai thiết bị.



**Cấu trúc source code app Android**

Do Arduino xử lý và truyền dữ liệu theo từng byte với giá trị chuỗi dạng Hex nên cần phải có phương thức toHex(String) chuyển đổi dữ liệu chuỗi String để thiết bị Arduino để xử lý và phương thức printHex(String) chuyển từ dữ liệu mảng byte[] với các phần tử dạng chuỗi Hex sang dạng chuỗi String với mục đích hiển thị nội dung thông thường lên màn hình.

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 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 | /\*\*  \* Display hex string to normal string  \*/  **public** **static** String **printHex**(String hex) {  StringBuilder sb = **new** StringBuilder();  **int** len = hex.length();  **try** {  **for** (**int** i = **0**; i < len; i += **2**) {  sb.append("0x").append(hex.substring(i, i + **2**)).append(" ");  }  } **catch** (NumberFormatException e) {  log("printHex NumberFormatException: " + e.getMessage());  } **catch** (StringIndexOutOfBoundsException e) {  log("printHex StringIndexOutOfBoundsException: " + e.getMessage());  }  **return** sb.toString();  }  /\*\*  \* Convert ASCII-byte hex to string array  \* @param hex - String  \* @return - an bytes array  \*/  **public** **static** **byte**[] **toHex**(String hex) {  **int** len = hex.length();  **byte**[] result = **new** **byte**[len];  **try** {  **int** index = **0**;  **for** (**int** i = **0**; i < len; i += **2**) {  result[index] = (**byte**) Integer.parseInt(hex.substring(i, i + **2**), **16**);  index++;  }  } **catch** (NumberFormatException e) {  log("toHex NumberFormatException: " + e.getMessage());  } **catch** (StringIndexOutOfBoundsException e) {  log("toHex StringIndexOutOfBoundsException: " + e.getMessage());  }  **return** result;  } | |

Chúng ta có lớp DeviceData (thường gọi là model) chứa dữ liệu của thiết bị Bluetooth khi kết nối vào thiết bị Android với các thuộc tính và contructor như sau

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 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  48  49  50  51  52  53  54  55  56  57 | **private** String name = "";  **private** String address = "";  **private** **int** bondState = BluetoothDevice.BOND\_NONE;  **private** ArrayList<ParcelUuid> uuids = **null**;  **private** **final** **int** deviceClass;  **private** **final** **int** majorDeviceClass;  **public** **DeviceData**(BluetoothDevice device, String emptyName) {  name = device.getName();  address = device.getAddress();  bondState = device.getBondState();  **if** (name == **null** || name.isEmpty()) name = emptyName;  deviceClass = device.getBluetoothClass().getDeviceClass();  majorDeviceClass = device.getBluetoothClass().getMajorDeviceClass();  uuids = BluetoothUtils.getDeviceUuids(device);  }  **public** String **getName**() {  **return** name;  }  **public** **void** **setName**(String name) {  **this**.name = name;  }  **public** String **getAddress**() {  **return** address;  }  **public** **void** **setAddress**(String address) {  **this**.address = address;  }  **public** **int** **getBondState**() {  **return** bondState;  }  **public** **void** **setBondState**(**int** bondState) {  **this**.bondState = bondState;  }  **public** ArrayList<ParcelUuid> **getUuids**() {  **return** uuids;  }  **public** **void** **setUuids**(ArrayList<ParcelUuid> uuids) {  **this**.uuids = uuids;  }  **public** **int** **getDeviceClass**() {  **return** deviceClass;  }  **public** **int** **getMajorDeviceClass**() {  **return** majorDeviceClass;  } | |

Để thực hiện việc tắt/mở chức năng Bluetooth ngay trong ứng dụng, chúng ta cần thực hiện như sau:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 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  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179 | **private** **static** **final** String TAG = "BluetoothUtils";  **private** **static** **final** **boolean** D = **true**;  **private** **static** **final** Map<String, String> uuidsDescriptions = **new** HashMap<String, String>();  **static** {  uuidsDescriptions.put("0001", "SDP");  uuidsDescriptions.put("0002", "UDP");  uuidsDescriptions.put("0003", "RFCOMM");  uuidsDescriptions.put("0004", "TCP");  uuidsDescriptions.put("0005", "TCS-BIN");  uuidsDescriptions.put("0006", "TCS-AT");  uuidsDescriptions.put("0007", "ATT");  uuidsDescriptions.put("0008", "OBEX");  uuidsDescriptions.put("0009", "IP");  uuidsDescriptions.put("000A", "FTP");  uuidsDescriptions.put("000C", "HTTP");  uuidsDescriptions.put("000E", "WSP");  uuidsDescriptions.put("000F", "BNEP");  uuidsDescriptions.put("0010", "UPNP");  uuidsDescriptions.put("0011", "HIDP");  uuidsDescriptions.put("0012", "HardcopyControlChannel");  uuidsDescriptions.put("0014", "HardcopyDataChannel");  uuidsDescriptions.put("0016", "HardcopyNotification");  uuidsDescriptions.put("0017", "AVCTP");  uuidsDescriptions.put("0019", "AVDTP");  uuidsDescriptions.put("001B", "CMTP");  uuidsDescriptions.put("001E", "MCAPControlChannel");  uuidsDescriptions.put("001F", "MCAPDataChannel");  uuidsDescriptions.put("0100", "L2CAP");  uuidsDescriptions.put("1000", "ServiceDiscoveryServerService");  uuidsDescriptions.put("1001", "BrowseGroupDescriptorService");  uuidsDescriptions.put("1002", "PublicBrowseGroupService");  uuidsDescriptions.put("1101", "SerialPortService");  uuidsDescriptions.put("1102", "LANAccessUsingPPPService");  uuidsDescriptions.put("1103", "DialupNetworkingService");  uuidsDescriptions.put("1104", "IrMCSyncService");  uuidsDescriptions.put("1105", "OBEXObjectPushService");  uuidsDescriptions.put("1106", "OBEXFileTransferService");  uuidsDescriptions.put("1107", "IrMCSyncCommandService");  uuidsDescriptions.put("1108", "HeadsetService");  uuidsDescriptions.put("1109", "CordlessTelephonyService");  uuidsDescriptions.put("110A", "AudioSourceService");  uuidsDescriptions.put("110B", "AudioSinkService");  uuidsDescriptions.put("110C", "AVRemoteControlTargetService");  uuidsDescriptions.put("110D", "AdvancedAudioDistributionService");  uuidsDescriptions.put("110E", "AVRemoteControlService");  uuidsDescriptions.put("110F", "VideoConferencingService");  uuidsDescriptions.put("1110", "IntercomService");  uuidsDescriptions.put("1111", "FaxService");  uuidsDescriptions.put("1112", "HeadsetAudioGatewayService");  uuidsDescriptions.put("1113", "WAPService");  uuidsDescriptions.put("1114", "WAPClientService");  uuidsDescriptions.put("1115", "PANUService");  uuidsDescriptions.put("1116", "NAPService");  uuidsDescriptions.put("1117", "GNService");  uuidsDescriptions.put("1118", "DirectPrintingService");  uuidsDescriptions.put("1119", "ReferencePrintingService");  uuidsDescriptions.put("111A", "ImagingService");  uuidsDescriptions.put("111B", "ImagingResponderService");  uuidsDescriptions.put("111C", "ImagingAutomaticArchiveService");  uuidsDescriptions.put("111D", "ImagingReferenceObjectsService");  uuidsDescriptions.put("111E", "HandsfreeService");  uuidsDescriptions.put("111F", "HandsfreeAudioGatewayService");  uuidsDescriptions.put("1120", "DirectPrintingReferenceObjectsService");  uuidsDescriptions.put("1121", "ReflectedUIService");  uuidsDescriptions.put("1122", "BasicPringingService");  uuidsDescriptions.put("1123", "PrintingStatusService");  uuidsDescriptions.put("1124", "HumanInterfaceDeviceService");  uuidsDescriptions.put("1125", "HardcopyCableReplacementService");  uuidsDescriptions.put("1126", "HCRPrintService");  uuidsDescriptions.put("1127", "HCRScanService");  uuidsDescriptions.put("1128", "CommonISDNAccessService");  uuidsDescriptions.put("1129", "VideoConferencingGWService");  uuidsDescriptions.put("112A", "UDIMTService");  uuidsDescriptions.put("112B", "UDITAService");  uuidsDescriptions.put("112C", "AudioVideoService");  uuidsDescriptions.put("112D", "SIMAccessService");  uuidsDescriptions.put("112E", "Phonebook Access - PCE");  uuidsDescriptions.put("112F", "Phonebook Access - PSE");  uuidsDescriptions.put("1130", "Phonebook Access");  uuidsDescriptions.put("1131", "Headset - HS");  uuidsDescriptions.put("1132", "Message Access Server");  uuidsDescriptions.put("1133", "Message Notification Server");  uuidsDescriptions.put("1134", "Message Access Profile");  uuidsDescriptions.put("1135", "GNSS");  uuidsDescriptions.put("1136", "GNSS\_Server");  uuidsDescriptions.put("1200", "PnPInformationService");  uuidsDescriptions.put("1201", "GenericNetworkingService");  uuidsDescriptions.put("1202", "GenericFileTransferService");  uuidsDescriptions.put("1203", "GenericAudioService");  uuidsDescriptions.put("1204", "GenericTelephonyService");  }  **public** **static** ArrayList<ParcelUuid> **getDeviceUuids**(BluetoothDevice device) {  ArrayList<ParcelUuid> result = **new** ArrayList<ParcelUuid>();  **try** {  Method method = device.getClass().getMethod("getUuids", **null**);  ParcelUuid[] phoneUuids = (ParcelUuid[]) method.invoke(device, **null**);  **if** (phoneUuids != **null**) {  **for** (ParcelUuid uuid : phoneUuids) {  **if** (D) Log.d(TAG, device.getName() + ": " + uuid.toString());  result.add(uuid);  }  }  } **catch** (NoSuchMethodException e) {  e.printStackTrace();  **if** (D) Log.e(TAG, "getDeviceUuids() failed", e);  } **catch** (InvocationTargetException e) {  e.printStackTrace();  **if** (D) Log.e(TAG, "getDeviceUuids() failed", e);  } **catch** (IllegalAccessException e) {  e.printStackTrace();  **if** (D) Log.e(TAG, "getDeviceUuids() failed", e);  }  **return** result;  }  **private** **static** ArrayList<String> **getDeviceServices**(ArrayList<ParcelUuid> uuids) {  ArrayList<String> result = **new** ArrayList<String>();  **for** (ParcelUuid uuid : uuids) {  String s = uuid.toString().toUpperCase();  **boolean** found = **false**;  **for** (Map.Entry<String, String> entry : uuidsDescriptions.entrySet()) {  String key = entry.getKey().toUpperCase();  String value = entry.getValue();  **if** (s.startsWith("0000" + key)) {  found = **true**;  result.add(value);  **break**;  }  }  **if** (!found) {  String desc = "Unknown service UUID 0x" + s.substring(**4**, **8**);  result.add(desc);  }  }  **return** result;  }  **public** **static** ArrayList<String> **getDeviceServices**(BluetoothDevice device) {  ArrayList<ParcelUuid> uuids = getDeviceUuids(device);  **return** **getDeviceServices**(uuids);  }  **public** **static** BluetoothSocket **createRfcommSocket**(BluetoothDevice device) {  BluetoothSocket tmp = **null**;  **try** {  Class class1 = device.getClass();  Class aclass[] = **new** Class[**1**];  aclass[**0**] = Integer.TYPE;  Method method = class1.getMethod("createRfcommSocket", aclass);  Object aobj[] = **new** Object[**1**];  aobj[**0**] = Integer.valueOf(**1**);  tmp = (BluetoothSocket) method.invoke(device, aobj);  } **catch** (NoSuchMethodException e) {  e.printStackTrace();  **if** (D) Log.e(TAG, "createRfcommSocket() failed", e);  } **catch** (InvocationTargetException e) {  e.printStackTrace();  **if** (D) Log.e(TAG, "createRfcommSocket() failed", e);  } **catch** (IllegalAccessException e) {  e.printStackTrace();  **if** (D) Log.e(TAG, "createRfcommSocket() failed", e);  }  **return** tmp;  } | |

Sau khi kết nối thành công, chúng ta có các phương thức kiểm tra trạng thái của kết nối, phương thức để gửi/nhận dữ liệu theo cách đồng bộ liên tục qua bluetooth được hiện trong lớp DeviceConnector

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | |  |  |  |  | | --- | --- | --- | --- | | |  |  | | --- | --- | | 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  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228  229  230  231  232  233  234  235  236  237  238  239  240  241  242  243  244  245  246  247  248  249  250  251  252  253  254  255  256  257  258  259  260  261  262  263  264  265  266  267  268  269  270  271  272  273  274  275  276  277  278  279  280  281  282  283  284  285  286  287  288  289  290  291  292  293  294  295  296  297  298  299  300  301  302  303  304  305  306  307  308  309  310  311  312  313  314  315  316  317  318  319 | **private** **static** **final** String TAG = "DeviceConnector";  **private** **static** **final** **boolean** D = **false**;  // Constants that indicate the current connection state  **public** **static** **final** **int** STATE\_NONE = **0**; // we're doing nothing  **public** **static** **final** **int** STATE\_CONNECTING = **1**; // now initiating an outgoing connection  **public** **static** **final** **int** STATE\_CONNECTED = **2**; // now connected to a remote device  **private** **int** mState;  **private** **final** BluetoothAdapter btAdapter;  **private** **final** BluetoothDevice connectedDevice;  **private** ConnectThread mConnectThread;  **private** ConnectedThread mConnectedThread;  **private** **final** Handler mHandler;  **private** **final** String deviceName;  **public** **DeviceConnector**(DeviceData deviceData, Handler handler) {  mHandler = handler;  btAdapter = BluetoothAdapter.getDefaultAdapter();  connectedDevice = btAdapter.getRemoteDevice(deviceData.getAddress());  deviceName = (deviceData.getName() == **null**) ? deviceData.getAddress() : deviceData.getName();  mState = STATE\_NONE;  }  /\*\*  \* connect to device  \*/  **public** **synchronized** **void** **connect**() {  **if** (D) Log.d(TAG, "connect to: " + connectedDevice);  **if** (mState == STATE\_CONNECTING) {  **if** (mConnectThread != **null**) {  **if** (D) Log.d(TAG, "cancel mConnectThread");  mConnectThread.cancel();  mConnectThread = **null**;  }  }  **if** (mConnectedThread != **null**) {  **if** (D) Log.d(TAG, "cancel mConnectedThread");  mConnectedThread.cancel();  mConnectedThread = **null**;  }  // Start the thread to connect with the given device  mConnectThread = **new** ConnectThread(connectedDevice);  mConnectThread.start();  setState(STATE\_CONNECTING);  }  /\*\*  \* disconnect  \*/  **public** **synchronized** **void** **stop**() {  **if** (D) Log.d(TAG, "stop");  **if** (mConnectThread != **null**) {  **if** (D) Log.d(TAG, "cancel mConnectThread");  mConnectThread.cancel();  mConnectThread = **null**;  }  **if** (mConnectedThread != **null**) {  **if** (D) Log.d(TAG, "cancel mConnectedThread");  mConnectedThread.cancel();  mConnectedThread = **null**;  }  setState(STATE\_NONE);  }  /\*\*  \* set status of device  \*  \* @param state - the status  \*/  **private** **synchronized** **void** **setState**(**int** state) {  **if** (D) Log.d(TAG, "setState() " + mState + " -> " + state);  mState = state;  mHandler.obtainMessage(DeviceControlActivity.MESSAGE\_STATE\_CHANGE, state, -**1**).sendToTarget();  }  /\*\*  \* get device status  \*/  **public** **synchronized** **int** **getState**() {  **return** mState;  }  **public** **synchronized** **void** **connected**(BluetoothSocket socket) {  **if** (D) Log.d(TAG, "connected");  // Cancel the thread that completed the connection  **if** (mConnectThread != **null**) {  **if** (D) Log.d(TAG, "cancel mConnectThread");  mConnectThread.cancel();  mConnectThread = **null**;  }  **if** (mConnectedThread != **null**) {  **if** (D) Log.d(TAG, "cancel mConnectedThread");  mConnectedThread.cancel();  mConnectedThread = **null**;  }  setState(STATE\_CONNECTED);  // Send the name of the connected device back to the UI Activity  Message msg = mHandler.obtainMessage(DeviceControlActivity.MESSAGE\_DEVICE\_NAME, deviceName);  mHandler.sendMessage(msg);  // Start the thread to manage the connection and perform transmissions  mConnectedThread = **new** ConnectedThread(socket);  mConnectedThread.start();  }  **public** **void** **write**(**byte**[] data) {  ConnectedThread r;  // Synchronize a copy of the ConnectedThread  **synchronized** (**this**) {  **if** (mState != STATE\_CONNECTED) **return**;  r = mConnectedThread;  }  // Perform the write unsynchronized  **if** (data.length == **1**) r.write(data[**0**]);  **else** r.writeData(data);  }  **private** **void** **connectionFailed**() {  **if** (D) Log.d(TAG, "connectionFailed");  // Send a failure message back to the Activity  Message msg = mHandler.obtainMessage(DeviceControlActivity.MESSAGE\_TOAST);  Bundle bundle = **new** Bundle();  msg.setData(bundle);  mHandler.sendMessage(msg);  setState(STATE\_NONE);  }  **private** **void** **connectionLost**() {  // Send a failure message back to the Activity  Message msg = mHandler.obtainMessage(DeviceControlActivity.MESSAGE\_TOAST);  Bundle bundle = **new** Bundle();  msg.setData(bundle);  mHandler.sendMessage(msg);  setState(STATE\_NONE);  }  /\*\*  \* class stream connect to BT device  \*/  **private** **class** **ConnectThread** **extends** Thread {  **private** **static** **final** String TAG = "ConnectThread";  **private** **static** **final** **boolean** D = **false**;  **private** **final** BluetoothSocket mmSocket;  **private** **final** BluetoothDevice mmDevice;  **public** **ConnectThread**(BluetoothDevice device) {  **if** (D) Log.d(TAG, "create ConnectThread");  mmDevice = device;  mmSocket = BluetoothUtils.createRfcommSocket(mmDevice);  }  **public** **void** **run**() {  **if** (D) Log.d(TAG, "ConnectThread run");  btAdapter.cancelDiscovery();  **if** (mmSocket == **null**) {  **if** (D) Log.d(TAG, "unable to connect to device, socket isn't created");  connectionFailed();  **return**;  }  // Make a connection to the BluetoothSocket  **try** {  // This is a blocking call and will only return on a  // successful connection or an exception  mmSocket.connect();  } **catch** (IOException e) {  // Close the socket  **try** {  mmSocket.close();  } **catch** (IOException e2) {  **if** (D) Log.e(TAG, "unable to close() socket during connection failure", e2);  }  connectionFailed();  **return**;  }  // Reset the ConnectThread because we're done  **synchronized** (DeviceConnector.this) {  mConnectThread = **null**;  }  // Start the connected thread  connected(mmSocket);  }  **public** **void** **cancel**() {  **if** (D) Log.d(TAG, "ConnectThread cancel");  **if** (mmSocket == **null**) {  **if** (D) Log.d(TAG, "unable to close null socket");  **return**;  }  **try** {  mmSocket.close();  } **catch** (IOException e) {  **if** (D) Log.e(TAG, "close() of connect socket failed", e);  }  }  }  /\*\*  \* class stream to transfer data from BT device  \*/  **private** **class** **ConnectedThread** **extends** Thread {  **private** **static** **final** String TAG = "ConnectedThread";  **private** **static** **final** **boolean** D = **false**;  **private** **final** BluetoothSocket mmSocket;  **private** **final** InputStream mmInStream;  **private** **final** OutputStream mmOutStream;  **public** **ConnectedThread**(BluetoothSocket socket) {  **if** (D) Log.d(TAG, "create ConnectedThread");  mmSocket = socket;  InputStream tmpIn = **null**;  OutputStream tmpOut = **null**;  // Get the BluetoothSocket input and output streams  **try** {  tmpIn = socket.getInputStream();  tmpOut = socket.getOutputStream();  } **catch** (IOException e) {  **if** (D) Log.e(TAG, "temp sockets not created", e);  }  mmInStream = tmpIn;  mmOutStream = tmpOut;  }  /\*\*  \* read incoming stream  \*/  **public** **void** **run**() {  **if** (D) Log.i(TAG, "ConnectedThread run");  **byte**[] buffer = **new** **byte**[**512**];  **int** bytes;  StringBuilder readMessage = **new** StringBuilder();  **while** (**true**) {  **try** {  // read input data from the stream and collected in a response line  bytes = mmInStream.read(buffer);  String readed = **new** String(buffer, **0**, bytes);  readMessage.append(readed);  // check end of stream then stop stream line  **if** (readed.contains("\n")) {  mHandler.obtainMessage(DeviceControlActivity.MESSAGE\_READ, bytes, -**1**, readMessage.toString()).sendToTarget();  readMessage.setLength(**0**);  }  } **catch** (IOException e) {  **if** (D) Log.e(TAG, "disconnected", e);  connectionLost();  **break**;  }  }  }  /\*\*  \* write data to send to device  \*/  **public** **void** **writeData**(**byte**[] chunk) {  **try** {  mmOutStream.write(chunk);  mmOutStream.flush();  // Share the sent message back to the UI Activity  mHandler.obtainMessage(DeviceControlActivity.MESSAGE\_WRITE, -**1**, -**1**, chunk).sendToTarget();  } **catch** (IOException e) {  **if** (D) Log.e(TAG, "Exception during write", e);  }  }  /\*\*  \* write byte data  \*/  **public** **void** **write**(**byte** command) {  **byte**[] buffer = **new** **byte**[**1**];  buffer[**0**] = command;  **try** {  mmOutStream.write(buffer);  // Share the sent message back to the UI Activity  mHandler.obtainMessage(DeviceControlActivity.MESSAGE\_WRITE, -**1**, -**1**, buffer).sendToTarget();  } **catch** (IOException e) {  **if** (D) Log.e(TAG, "Exception during write", e);  }  }  /\*\*  \* close socket when cancel operation  \*/  **public** **void** **cancel**() {  **try** {  mmSocket.close();  } **catch** (IOException e) {  **if** (D) Log.e(TAG, "close() of connect socket failed", e);  }  }  } | |  | |  | |