Android SDK

Specification Version 2.1.7

NeoLAB Convergence Inc.

Revision History

Ver	Date	Contents	
1.40	04-May, 15	Added Offline data	ARM
1.50	30-Nov, 15	Connection procedure for Multi app	LMS
2.00	13-Jun, 16	Revised to SDK 2.0	LMS
2.10	07-Jul, 16	Separate IPenDotListerner from onReceiveDot() of IPenMsgListener	LMS
2.11	22-Feb, 17	Bluetooth LE	CJY
2.14	10-Aug, 17	Change history about Password	LMS
2.15	10-Jan, 18	Pen Profile, Multi Pen	LMS
2.16	10-Jan, 18	Revised text for BI	KDB
2.17	22-May, 19	Symbol detect	HRL

Contents

1. About this document

This document defines the communication protocol between Neo smartpen and apps. This protocol is not compatible with version 1.0, if the app needs to support NWP-F110 and NWP-F120, Protocol Ver 1.0 and Ver 2.0 have to be implemented.

Model Name	Pen Name	Protocol Version
NWP-F110	N2	1.0
NWP-F120	N/A	2.x
NWP-F50	N/A	2.x

2. Preface

Please prepare Android development environment(http://developer.android.com/sdk/index.html)

- Eclipse IDE (ADT) or android studio
- android SDK(min SDK 4.4-KitKat)

3. Add Library

nasdk_v[x.x].jar has to be copied into user's android project library

4. AndroidManifest.xml setting

Add manifest permission code to your android project.

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

5. Class summary

IPenCtrl	Connect Pen via Bluetooth
IPenMsgListener	Receive PenMsg Data from the pen

IPenDotListener	Receive Ncode™ data from the pen
PenMsg	Data structure
PenMsgType	A type of incoming data from the pen
Const/JsonTag	Definition of data information from the pen

6. register/unregister Broadcastreceiver

When pen controller instance initially created, context is registered by setContext(), in order to register Broadcastreceiver, execute the below method
If Penctrl.registerBroadcastBTDuplicate () executes, Broadcastreceiver is registered inside SDK.
If Penctrl.unregisterBroadcastBTDuplicate () executes, Broadcastreceiver is unregistered inside SDK.

```
public class NeoNoteApplication extends Application {
…
@Override
public void onCreate() {
   iPenCtrl = PenCtrl.getInstance();
    iPenCtrl.setContext(getApplicationContext())
    iPenCtrl.registerBroadcastBTDuplicate();
 …
}

@Override
public void onTerminate ()
{
    iPenCtrl.unregisterBroadcastBTDuplicate();
   super.onTerminate();
}
```

7. Get started Pen controller

All the function(such as connection, disconnection, stroke transmission) are implemented in pen controller class.

```
public class MainActivity extends Activity implements IPenMsgListener
{
        private IPenCtrl iPenCtrl;

        @Override
        protected void onCreate(Bundle savedInstanceState)
        {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_main);
              …
              // create pen controller instance
                iPenCtrl = PenCtrl.getInstance();

                // Start pen controller
```

```
        iPenCtrl.startup();

        // register pen event callback listener
        iPenCtrl.setListener(this);
          // register pen ncode callback listener
          iPenCtrl.setDotListener(this);
    }
```

8. Pen connection

Pairing process is required before connecting pen. please refer developer.android.com for pairing process and sample codes.

Connection function is called with parameters of pen mac_address and context.

```
String mac_address = "9c7bd20001d4";
iPenCtrl.connect(mac_address);
```

9. Pen Acknowledge process

Pen controller must register callback(IPenMsgListener) for receive response. And if pen controller method is executed, SDK transfer the PenMsg and Dot data to callback as an asynchronous response.

In this document, it depicts the realtime data transfer and connection. Please refer sample source code for more details.

step 1. Pen connection completion
step 2. Transmits password
step 3. Transmit use notes to pen by pen controller(IPenCtrl.reqAddUsingNote).
step 4. Exception process due to Bluetooth connections to other apps.

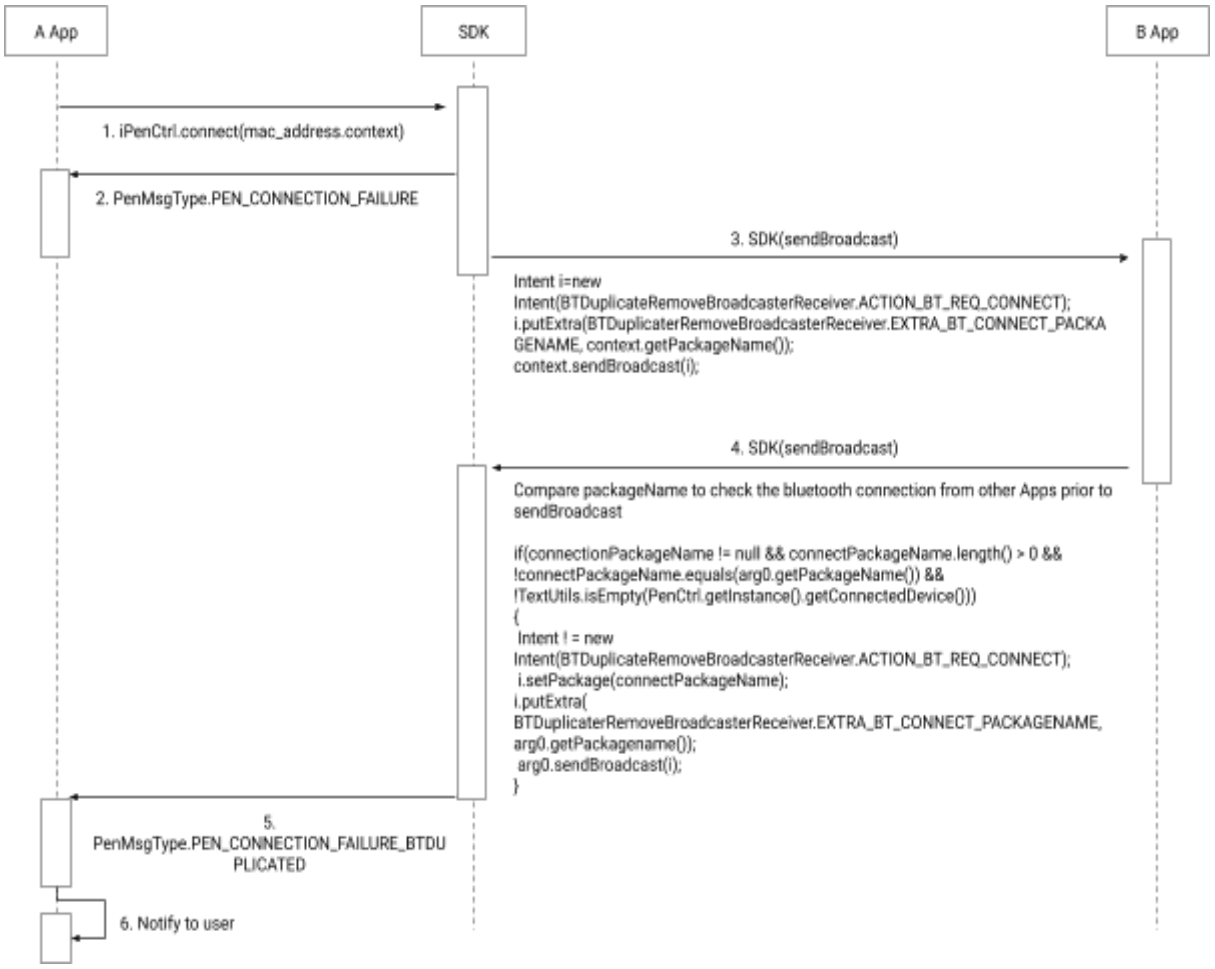
```
@Override
    public void onReceiveDot(Dot  dot) {
        // when connected, Ncode information is transfered as a Primitive type.
    }
    @Override
    public void onReceiveMessage(final PenMsg penMsg) {
        // data other than Ncode will be contained to a message object named PenMsg
        switch ( penMsg.penMsgType )
        {
            // step 1. Upon Connection competion (authentication is not proceeded)
            case PenMsgType.PEN_CONNECTION_SUCCESS:
                break;
            // step 2. Request pen's password (request right after
```

```
PEN_CONNECTION_SUCCESS)
                        case PenMsgType.PASSWORD_REQUEST:
                             String password = "";
                             iPenCtrl.inputPassword( password );
                                  break;
                        // step 3. Transmit use notes to pen by pen controller
                        // when authorized,  password of pen returned.
                        // and the setting for the page in use information is needed
                        // as well as section and owner in use
                        // care must be taken on these information
                       case PenMsg  Type.PEN_AUTHORIZED:
                            String password= "";
                            try
                            {
                               JSONObject obj= new JSONObject( content );
                               password = obj.getString(JsonTag.STRING_PEN_PASSWORD);
                            }
                            catch ( JSONException e )
                            {
                               e.printStackTrace();
                            }

                          iPenCtrl.regAddUsingNote( USING_SECTION_ID, USING_OWNER_ID);
                                  break;

                        // step 4.  Connection failure, pen is connected to another app
                        //acquire the packageName of the  app in connection
                        //disconnect Bluetooth by acquired info., retry alert must be given to UI
flow
                           // in this Sample, this procedure is implemented by Toast
                        case PenMsgType.PEN_CONNECTION_FAILURE_BTDUPLICATE:
                            String connected_Appname = "";
                            try
                            {
                               JSONObject job = new JSONObject( content );
                               connected_Appname = job.getString("packageName");
                            }
                            catch ( JSONException e )
                            {
                               e.printStackTrace();
                            }
                               Util.showToast( this, String.format("The pen is currently
connected to %s app. If you want to proceed, please disconnect the pen from %s
app.",connected_Appname,connected_Appname));
                        break;
                                                .
                                                .
                                                .
                }
        }
```

A sequential diagram is depicted as below.



10. onReceiveDot x, y, fx, fy

x, y : It coordinates of our dot code cell.(cell size is 2.371mm)
fx, fy : It is fractional part of our code.

Ex) (x + fx x 0.01) x 2.371 = millimeter unit

11. Notebook information

Note Type	NoteTitle	DX (Ncode™ unit)	DY (Ncode™ unit)	Width (Ncode™ unit)	Height (Ncode™ unit)
601	Pocket Note	5.40	5.45	35.26	62.59
602	Memo Note	5.40	5.45	35.26	62.59
603	Ring Note	5.52	5.41	63.46	88.88
605	FP Memo Pad	5.21	5.21	62.57	88.68
606	FP Original CEO	4.44	4.47	36.34	75.22
607	FP Original CL	4.33	4.46	62.81	91.09

608	Casual Planner	4.18	4.37	45.61	72.83

12 .nproj to Ncode™ unit

Scale = 72 / 600 * 56 = 6.72

nproj size /scale = Ncode™ size

ex) 500 height in nproj file, 500/Scale = 500/6.72 = 74.4 (Ncode™ unit)

13 . Metadata Processing

We use Ncode™ paper information as a metadata file called nproj. This section describes how to load and use the nproj file. nproj has an object named Symbol, so when the handwriting is written in the area of the symbol, the SDK can detect it and receive the symbol information with the callback.

Put nproj files at "nproj files path", and check symbols by pen. You can use sample files(NASDK2.0_sample_cole/nprojFiels).

See the link below for a detailed description of nproj.
Caster™ Lite XML format spec (english).pdf

```
public class NeoSampleService extends Service{
…
@Override
public void onCreate() {
   iPenCtrl = PenCtrl.getInstance();
   // Set MetadataListener at PenCtrl
   // You can get callback when the symbols are detected.
   iPenCtrl.setMetadataListener( mMetadataListener );

   // Load the Metadata file.
   metadataCtrl = MetadataCtrl.getInstance();
   metadataCtrl.loadFile( "nproj files path" );
 …
}

@Override
private IMetadataListener mMetadataListener = new IMetadataListener()
{
    @Override
    Public void onSymbolDetected( Symbol[] symbols )
    {
        // TODO : Processing for detected symbols
        // Implement functions corresponding to predefined Actions.
        // For example, if Symbol's Action is email, it sends an email.
    }
}
```

14. Offline Data Process

1. Request Offline data list : PenCtrl.reqOfflineDataList();

2. Receive Offline data list by callback

```
@Override
public void onReceiveMessage( PenMsg penMsg )
{
  switch ( penMsg.penMsgType )
  {
    // Offline Data List response of the pen
case PenMsgType.OFFLINE_DATA_NOTE_LIST:
  try
  {
    JSONArray list = new JSONArray( content );

    for ( int i = 0; i < list.length(); i++ )
    {
      JSONObject jobj = list.getJSONObject( i );

      int sectionId = jobj.getInt( JsonTag.INT_SECTION_ID );
      int ownerId = jobj.getInt( JsonTag.INT_OWNER_ID );
      int noteId = jobj.getInt( JsonTag.INT_NOTE_ID );
// request offline data
      Log.d( TAG, "offline(" + (i + 1) + ") note => sectionId : " + sectionId + ", ownerId
: " + ownerId + ", noteId : " + noteId );
    }
  }
  catch ( JSONException e )
  {
    e.printStackTrace();
  }
  Util.showToast( this, "offline data list is received." );

  break;
```

3. Reqeust ReqOfflineData

Request unit is a NoteBook unit,please make sure that OfflineData works asych communication
PenCtrl. reqOfflineData(int sectionId, int ownerId, int noteId)

4. Download information

Download start, finish, status, success/failure can be checked by a message of callback function.
If the message type is OFFLINE_DATA_FILE_CREATED, a download is completed.

Find offLineFile file path by JSON data(JsonTag.STRING_FILE_PATH).
Then if the offLineFile parsed, OffLineData work is completed.

```
@Override
public void onReceiveMessage( PenMsg penMsg )
{
  switch ( penMsg.penMsgType )
  {
    // Messages for offline data transfer begins
case PenMsgType.OFFLINE_DATA_SEND_START:
  break;

// Offline data transfer completion
case PenMsgType.OFFLINE_DATA_SEND_SUCCESS:
  break;

// Offline data transfer failure
case PenMsgType.OFFLINE_DATA_SEND_FAILURE:
  break;

// Progress of the data transfer process offline
case PenMsgType.OFFLINE_DATA_SEND_STATUS:
{
  try
  {
    JSONObject job = new JSONObject( content );
    int total = job.getInt( JsonTag.INT_TOTAL_SIZE );
    int received = job.getInt( JsonTag.INT_RECEIVED_SIZE );
    Log.d( TAG, "offline data send status => total : " + total + ", progress : " + received
);
  }
  catch ( JSONException e )
  {
    e.printStackTrace();
  }
}
  break;

// When the file transfer process of the download offline
case PenMsgType.OFFLINE_DATA_FILE_CREATED:
{
  try
  {
    JSONObject job = new JSONObject( content );
    int sectionId = job.getInt( JsonTag.INT_SECTION_ID );
    int ownerId = job.getInt( JsonTag.INT_OWNER_ID );
    int noteId = job.getInt( JsonTag.INT_NOTE_ID );
    int pageId = job.getInt( JsonTag.INT_PAGE_ID );
    String filePath = job.getString( JsonTag.STRING_FILE_PATH );
    Log.d( TAG, "offline data file created => sectionId : " + sectionId + ", ownerId : " +
ownerId + ", noteId : " + noteId + ", pageId : " + pageId + " filePath : " + filePath );
  }
  catch ( JSONException e )
  {
    e.printStackTrace();
  }
}
  break;
```

5. OffLineData Deletion

Received upon OffLineData를, the request of the file to pen can be made based on OffLineData policy(if it is not deleted, the file information can be duplicated

when the next OffLineData request is made)

PenCtrl. removeOfflineData(int sectionId, int ownerId);

15. Protocol Ver 2.0 major changes in function

1. PenCap Off

A Method to on/off the power via pen cap is added

2. Stroke(Dot) Data added

More info added penTipType(pen/eraser) , tiltX/ tiltY(pen tilt), twist (pen skew)

3. Renderer class deletion

Stroke render is no longer supported in SDK
Rendering is provided as a sample code

4. PenHover mode

Hover mode can be turned on/off

5. Pen CALIBRATION

Pen calibration function is deleted from SDK

6. Changes in Pen Time value

Pen time value is referred by UTC

7. OffLineData

Offline date can be requested per page.
Unlike Protocol ver 1.0, Protocol ver 2.0 has been removed step *OFFLINE_DATA_FILE_CREATED*.
Thus the stroke data can be received by callback directly without file work.

8. FW Upgrade

The fw file you transfer to the SDK should not be compressed.

9. reqAddUsingNote

Every Method related reqAddUsingNote overrides data from Protocol specification 2.0. If the multiple type of notebooks are requested, it has to be requested at a time via reqAddUsingNote(ArrayList<UseNoteData> noteList) Method

If your app support the protocol 1.0 and 2.0 regarding firmware, please check the comparison table below.

Method deleted	Method added
public void upgradePen(File fwFile, String targetPath) public void upgradePen(File fwFile)	public void upgradePen2(File fwFile, String fwVersion);
public void removeOfflineData(int sectionId, int ownerId)	public void reqOfflineDataRemove(int sectionId, int ownerId, int[] noteIds)
	public void reqOfflineDataList(int sectionId, int ownerId);
	public void reqOfflineDataPageList(int sectionId, int ownerId, int noteId);
	public void setOffLineDataListener(IOfflineDataListener listener);
	public IOfflineDataListener getOffLineDataListener();
	public void reqOfflineData(int sectionId, int ownerId, int noteId, int[] pageIds)
	public void reqAddUsingNote(ArrayList<UseNoteData> noteList)
	public void reqSetupPenCapOnOff(boolean on)
	public void reqSetupPenHover (boolean on)
	public int getProtocolVersion()
public void onReceiveDot(int sectionId, int ownerId, int noteId, int pageId, int x, int y, int fx, int fy, int pressure, long timestamp, int type, int color);	public void onReceiveDot(Dot dot)

PenMsgType deleted	PenMsgType Changed	PenMsgType Modified
PEN_CALIBRATION_START		
PEN_CALIBRATION_FINISH		
	PEN_SETUP_PEN_CAP_ONOFF	
	PEN_SETUP_HOVER_ONOFF	
	PEN_SETUP_OFFLINEDATA_SAVE_ONOFF	
	EVENT_LOW_BATTERY JSON Data "battery"	
	EVENT_POWER_OFF JSON Data "poweroff_reason"	
	OFFLINE_DATA_PAGE_LIST	
		PEN_STATUS JSON Data deleted "status" "timezone" "pen_tip_color" "acceleration_sensor_onoff" JSON Data added "pencap_onoff" "hover_mode" "offlinedata_save" "battery" the value 128 indicates charging\
		PASSWORD_SETUP_FAILURE

	JSON Data Added "retry_count" "reset_count"

16. Realtime stroke date in Protocol Ver 2.0

Due to the changes in protocol 2.0, IPenMsgListener is changed accordingly.

In Protocol 1.0
public void onReceiveDot(int sectionId, int ownerId, int noteId, int pageId, int x, int y, int fx, int fy, int pressure, long timestamp, int type, int color)

In Protocol 2.0
public void onReceiveDot(Dot dot)
Dot object includes the old data and new added data.
In protocol 1.0, the value of newly added data will be filled with initial value.

```
@Override
public void onReceiveDot( Dot dot )
{
  NLog.d( "onReceiveDot sectionId=" + dot.sectionId + ",ownerId=" + dot.ownerId +
";noteId=" + dot.noteId + ";pageId=" + dot.pageId + ";x=" + dot.x + ";x=" + dot.y +
";y=" + dot.fx + ";fy=" + dot.fy + ";pressure=" + dot.force + ";timestamp=" +
dot.timestamp + ";type=" + dot.dotType + ";color=" + dot.color + ";tiltX=" + dot.tiltX +
";tiltY=" + dot.tiltY + ";twist=" + dot.twist + ";penTipType=" + dot.penTipType );
  sendPenDotByBroadcast( dot );
}
```

17. Offline processing in Protocol Ver 2.0

1. Request the stored Offline data list from the pen
PenCtrl.reqOfflineDataList();

2. In case if you are requesting per page, request Offline data via PenCtrl.reqOfflineDataPageList(int sectionId, int ownerId, int noteId) based on the acquired sectioned, ownerId, noteId
3. Offline data is requested via PenCtrl.reqOfflineData()
4. Receive Offlinedata by a callback function

```
@Override
public void onReceiveOfflineStrokes ( Stroke[] strokes, int sectionId, int ownerId, int noteId )
{
  NLog.d( "onReceiveOfflineStrokes strokes="+strokes.length );
  Intent i = new Intent( Const.Broadcast.ACTION_OFFLINE_STROKES );
  i.putExtra( Const.Broadcast.OFFLINE_STROKES, strokes );
```

```
    context.sendBroadcast( i );
}
```

18. Bluetooth LE

17. 1 Requirement

Andorid 5.0, API Level 21(Lollipop)

17.2 Supported Neo smartpen

Pen that use protocol ver2 (ex. F-50, F-120, C200, D100 etc.)

17.3 How To Use

SPP and LE can be selected using setLeMode method added to iPenCtrl class.

If parameter is true, mode is LE. Otherwise is SPP.

```
iPenCtrl.setLeMode(true);
```

Other API of SPP and LE are all the same.

19. Throughput of a Bluetooth SPP/LE Connection

A SPP profile ensures that data rates up to 128 kbps can be used, however the actual usable SPP bandwidth is much lower than 128 kbps due to the additional processings such as buffer-wise transmission, Error corrections, Exception handling so on and so forth, the maximum achievable data rate of the SDK is closer to 80kbps but this is device dependent.

As a rule of thumb;

In SPP SDK under Android OS:

1 minutes: 10 kbytes

10 minutes : 10k x 60 = 600 kbytes

60 minutes: 36 Mbytes

Please refer Bluetooth SPP specification for more information

Bluetooth Low Energy (BLE) is designed for low data rate applications, like control signals, sensor readings, etc. However, occasionally there is a need for higher data rates with BLE such Neo smartpen. The theoretical over-the-air data rate of BLE is 1 Mbps, but that's the PHY Layer transfer rate between devices and doesn't account for protocol overhead. The maximum achievable data rate is for the SDK is closer to 16kbps but this is device dependent.

As a rule of thumb;

In LE SDK under IOS:

1 minutes: 2k x 60 = 120 kbytes

10 minutes : 1.2 Mbytes

60 minutes : 7.2 Mbytes

Please refer Bluetooth 4.0 specification for more information

20. Pen Profile

19.1 What is Pen Profile?

Pen Profile is the API to directly write / read Pen information to the Pen without being bound to the application.

For example, there are data such as Pen Nickname, Pen Drawing Brush Type and so on.

Profile-related API responses are delivered in JSON format.

You can create or delete a Pen Profile File for each vendor, but if you want to create your own Pen Profile File, you need to be assigned a FileName and Password after consulting with related department (_srvplan@neolab.kr).

19.2 getProfileInfo(String proFileName)

This API gets the information for the Profile File.

```
public void handleMsg(final int penMsgType,final String content )
{
    NLog.d( "handleMsg : " + penMsgType);
    runOnUiThread( new Runnable()
    {
        @Override
        public void run ()
        {
            JSONObject profileObj = null;
            JSONArray array = null;
            String profileName = "";
            String result = "";
            switch ( penMsgType )
            {
                case PenMsgType.PROFILE_INFO:
                    try
                    {
                        profileObj = new JSONObject( content );
                        profileName = profileObj.getString( JsonTag.STRING_PROFILE_NAME );
                        int status = profileObj.getInt( JsonTag.INT_PROFILE_RES_STATUS);
                        result = "Response
PROFILE_INFO\nprofileName="+profileName+",status="+status+"(0x"+Integer.toHexSt
ring(status)+")";
                        if(status == PenProfile.PROFILE_STATUS_SUCCESS)
                        {
                            int TOTAL_SECTOR_COUNT = profileObj.getInt(
JsonTag.INT_PROFILE_INFO_TOTAL_SECTOR_COUNT );
                            int SECTOR_SIZE = profileObj.getInt(
```

```
JsonTag.INT_PROFILE_INFO_SECTOR_SIZE );
                    int USE_SECTOR_COUNT = profileObj.getInt(
JsonTag.INT_PROFILE_INFO_USE_SECTOR_COUNT );
                    int USE_KEY_COUNT = profileObj.getInt(
JsonTag.INT_PROFILE_INFO_USE_KEY_COUNT );
                    result +=
"\nTOTAL_SECTOR_COUNT="+TOTAL_SECTOR_COUNT+"\nSECTOR_SIZE="+SECTOR_SI
ZE+"\nUSE_SECTOR_COUNT="+USE_SECTOR_COUNT+"\nUSE_KEY_COUNT="+USE_KEY
_COUNT;

                        }

                    }
                    catch ( JSONException e )
                    {
                        e.printStackTrace();
                    }
                    break;

                }
                if(result.length() == 0)
                    return;
                result_text.append( "\n"+result );
            }
        } );

    }
```

If there is a file corresponding to proFileName, Response of JsonTag.INT_PROFILE_RES_STATUS is PenProfile.PROFILE_STATUS_SUCCESS. And the value is as follows.

JsonTag.***INT_PROFILE_INFO_TOTAL_SECTOR_COUNT***
JsonTag.***INT_PROFILE_INFO_SECTOR_SIZE***
JsonTag.***INT_PROFILE_INFO_USE_SECTOR_COUNT***
JsonTag.***INT_PROFILE_INFO_USE_KEY_COUNT***

The size of the Profile File is Sector Size * Sector count.

19.3 createProfile (String proFileName, byte[] password)

This API creates the Profile File.

The proFileName and password assigned from the related department are input as parameters.

```
public void handleMsg(final int penMsgType,final String content )
{
    NLog.d( "handleMsg : " + penMsgType);
    runOnUiThread( new Runnable()
```

```
    {
        @Override
        public void run ()
        {
            JSONObject profileObj = null;
            JSONArray array = null;
            String profileName = "";
            String result = "";
            switch ( penMsgType )
            {
                case PenMsgType.PROFILE_CREATE:
                    try
                    {
                        profileObj = new JSONObject( content );
                        profileName = profileObj.getString( JsonTag.STRING_PROFILE_NAME );
                        int status = profileObj.getInt( JsonTag.INT_PROFILE_RES_STATUS);
                        result = "Response
PROFILE_CREATE\nprofileName="+profileName+",status="+status+"(0x"+Integer.toHex
String(status)+")";
                    }
                    catch ( JSONException e )
                    {
                        e.printStackTrace();
                    }
                    break;


            }
            if(result.length() == 0)
                return;
            result_text.append( "\n"+result );
        }
    } );

  }
```

If the status value is not PenProfile.PROFILE_STATUS_SUCCESS but PenProfile.PROFILE_STATUS_NO_PERMISSION , please ask proFileName and password for related department (_srvplan@neolab.kr).

19.4 deleteProfile (String proFileName, byte[] password)

This API deletes Profile File.

```
  public void handleMsg(final int penMsgType,final String content )
  {
      NLog.d( "handleMsg : " + penMsgType);
      runOnUiThread( new Runnable()
      {
          @Override
          public void run ()
```

```
        {
            JSONObject profileObj = null;
            JSONArray array = null;
            String profileName = "";
            String result = "";
            switch ( penMsgType )
            {
                case PenMsgType.PROFILE_DELETE:
                    try
                    {
                        profileObj = new JSONObject( content );
                        profileName = profileObj.getString( JsonTag.STRING_PROFILE_NAME );
                        int status = profileObj.getInt( JsonTag.INT_PROFILE_RES_STATUS);
                        result = "Response
PROFILE_DELETE\nprofileName="+profileName+",status="+status+"(0x"+Integer.toHex
String(status)+")";
                    }
                    catch ( JSONException e )
                    {
                        e.printStackTrace();
                    }

                    break;

            }
            if(result.length() == 0)
                return;
            result_text.append( "\n"+result );
        }
    } );

}
```

If the status value is not PenProfile.PROFILE_STATUS_SUCCESS but PenProfile.PROFILE_STATUS_NO_PERMISSION , please contact _srvplan@neolab.kr for proFileName and password.

19.5 writeProfileValue (String proFileName, byte[] password, String[] keys, byte[][] data)

Because The writeProfileValue, readProfileValue, and deleteProfileValue APIs is inputted a Keys as a Array, Response is passed as a JsonArray.

And Status for Key exist each other.

```
public void handleMsg(final int penMsgType,final String content )
{
    NLog.d( "handleMsg : " + penMsgType);
    runOnUiThread( new Runnable()
    {
        @Override
        public void run ()
        {
            JSONObject profileObj = null;
            JSONArray array = null;
```

```
            String profileName = "";
            String result = "";
            switch ( penMsgType )
            {
               case PenMsgType.PROFILE_WRITE_VALUE:
                  try
                  {
                     profileObj = new JSONObject( content );
                     profileName = profileObj.getString( JsonTag.STRING_PROFILE_NAME );
                     result = "Response
PROFILE_WRITE_VALUE\nprofileName="+profileName;
                     array = profileObj.getJSONArray( JsonTag.ARRAY_PROFILE_RES );
                     for(int i = 0; i < array.length(); i++)
                     {
                        JSONObject obj = array.getJSONObject( i );
                        String key = obj.getString( JsonTag.STRING_PROFILE_KEY );
                        int status = obj.getInt( JsonTag.INT_PROFILE_RES_STATUS );
                        result +=
"\nkey="+key+",status="+status+"(0x"+Integer.toHexString(status)+")";
                     }
                  }
                  catch ( JSONException e )
                  {
                     e.printStackTrace();
                  }
                  break;

            }
            if(result.length() == 0)
               return;
            result_text.append( "\n"+result );
         }
      } );

   }
```

19.6 readProfileValue (String proFileName, String[] keys)

```
   public void handleMsg(final int penMsgType,final String content )
   {
      NLog.d( "handleMsg : " + penMsgType);
      runOnUiThread( new Runnable()
      {
         @Override
         public void run ()
         {
            JSONObject profileObj = null;
            JSONArray array = null;
            String profileName = "";
            String result = "";
            switch ( penMsgType )
            {
               case PenMsgType.PROFILE_READ_VALUE:
                  try
                  {
                     profileObj = new JSONObject( content );
```

```
                    profileName = profileObj.getString( JsonTag.STRING_PROFILE_NAME );
                    result = "Response
PROFILE_READ_VALUE\nprofileName="+profileName;
                    array = profileObj.getJSONArray( JsonTag.ARRAY_PROFILE_RES );
                    for(int i = 0; i < array.length(); i++)
                    {
                        JSONObject obj = array.getJSONObject( i );
                        String key = obj.getString( JsonTag.STRING_PROFILE_KEY );
                        int status = obj.getInt( JsonTag.INT_PROFILE_RES_STATUS );
                        String value = null;
                        if(status == PenProfile.PROFILE_STATUS_SUCCESS)
                        {
                            String data = (String)obj.get( JsonTag.BYTE_PROFILE_VALUE );
                            byte[] decodeByte = Base64.decode(data ,Base64.DEFAULT);
                            value = new String( decodeByte );
                            NLog.d( "test e = " +decodeByte.length);
                        }
                        result +=
"\nkey="+key+",status="+status+"(0x"+Integer.toHexString(status)+")";
                        if(value != null)
                            result += ",value="+value;
                    }
                }
                catch ( JSONException e )
                {
                    e.printStackTrace();
                }
                break;

            }
            if(result.length() == 0)
                return;
            result_text.append( "\n"+result );
        }
    } );

  }
```

19.7 deleteProfileValue (String proFileName, byte[] password, String[] keys)

```
  public void handleMsg(final int penMsgType,final String content )
  {
      NLog.d( "handleMsg : " + penMsgType);
      runOnUiThread( new Runnable()
      {
          @Override
          public void run ()
          {
              JSONObject profileObj = null;
              JSONArray array = null;
              String profileName = "";
              String result = "";
              switch ( penMsgType )
              {
                  case PenMsgType.PROFILE_DELETE_VALUE:
                      try
```

```
                {
                    profileObj = new JSONObject( content );
                    profileName = profileObj.getString( JsonTag.STRING_PROFILE_NAME );
                    result = "Response
PROFILE_DELETE_VALUE\nprofileName="+profileName;
                    array = profileObj.getJSONArray( JsonTag.ARRAY_PROFILE_RES );
                    for(int i = 0; i < array.length(); i++)
                    {
                        JSONObject obj = array.getJSONObject( i );
                        String key = obj.getString( JsonTag.STRING_PROFILE_KEY );
                        int status = obj.getInt( JsonTag.INT_PROFILE_RES_STATUS );
                        result +=
"\nkey="+key+",status="+status+"(0x"+Integer.toHexString(status)+")";
                    }
                }
                catch ( JSONException e )
                {
                    e.printStackTrace();
                }
                break;
            }
            if(result.length() == 0)
                return;
            result_text.append( "\n"+result );
        }
    } );

  }
```

19.8 Error Code

PenProfile.PROFILE_STATUS_SUCCESS	API Success
PenProfile.PROFILE_STATUS_FAILURE	API Fail
PenProfile.PROFILE_STATUS_EXIST_PROFILE_ALREADY	Occurs if the file already exists when the createProfile API request.
PenProfile.PROFILE_STATUS_NO_EXIST_PROFILE	Occurs if the file does not have when the deleteProfile API request.
PenProfile.PROFILE_STATUS_NO_EXIST_KEY	Occurs if the Key does not exist when the readProfileValue, deleteProfile API request.
PenProfile.PROFILE_STATUS_NO_PERMISSION	Please contact the related department for proFileName and password.

PenProfile.PROFILE_STATUS_BUFFER_SIZE_ERR	Occurs when storage exceeds

19.9 Profile using at NeoNotes

Profile File Name : “neolab”

Key	String	Descript
PenProfile.KEY_PEN_NAME	**"N_name"**	Nickname of Pen
PenProfile.KEY_PEN_STROKE_THICKNESS_LEVEL	**"N_thickness"**	Thickness level of Drawing
PenProfile.KEY_PEN_COLOR_INDEX	**"N_color_index"**	Color Index of Drawing
PenProfile.KEY_PEN_COLOR_AND_HISTORY	**"N_color"**	History Color value of Color Palette
PenProfile.KEY_USER_CALIBRATION	**"N_pressure"**	User Calibration Point value

The above table is the Profile Key actually used since NeoNotes ver 1.37.150.

You can test getProfileInfo and readProfileValue using profile file name("neolab") and the key in the table .

19.9 Notice

Because the data stored in the profile is used in various platforms, it is the rule to write / read in Little-Endian format.

Data was encoded using Base64.encodeToString when passing in Json format from SDK.

Please use Base64.decode (data, Base64.DEFAULT) to decode when parsing.

21. Multi Connect Pen

We provide a way to connect multiple pens at the same time, just by adding an address parameter.

If you use MultiPenCtrl class instead of PenCtrl class, you can try to connect multiple pens.

However, keep in mind that connecting more than two pens at the same time may cause the connection to fail.

Generally, Bluetooth spec supports up to max 7, but it differs from device to OS.

```
iPenCtrl = MultiPenCtrl.getInstance();
iPenCtrl.setListener( this );
iPenCtrl.setDotListener( mPenReceiveDotListener );
iPenCtrl.setOffLineDataListener( mOfflineDataListener );
iPenCtrl.connect( address , isLeMode);

…

private IPenDotListener mPenReceiveDotListener = new IPenDotListener() {

  @Override
  public void onReceiveDot ( String macAddress, Dot dot )
  {
    NLog.d( "NeoSampleService onReceiveDot mac_address="+macAddress+"dotType="
+ dot.dotType+" ,pressure="+dot.pressure+",x="+dot.getX()+",y="+dot.getY() );

  }

};

@Override
public void onReceiveMessage( String macAddress, PenMsg penMsg )
{
  NLog.d( "PenClientCtrl onReceiveMessage
penMsg="+penMsg.getPenMsgType()+",getContent:"+penMsg.getContent() );
}
```

22. Appendix

This article is excerpted from https://atmosphere.anaren.com/wiki/Data_rates_using_BLE in order to give a better understanding of BLE and its throughput

1. BLE Connection Parameters

There are multiple connection parameters (defined in Bluetooth 4.0 specification, Volume 3, Part A, Section 4.20) that will determine the throughput. Keep in mind that higher throughput will result in more power consumption.

1.1 Connection Interval

This defines how often that the central communicates with the peripheral. There can be a maximum of four packets sent per connection interval, and each packet can have up to 20B of payload. According to the BLE specification, the allowable range for connection parameters is from 7.5mSec to 4000mSec. The Connection Interval is the parameter that most affects data rate. Think of the Connection Interval as the train schedule: for example, trains leave the station every half hour. Each train can have 1-4 cars, and each car can hold 0 to 20 bytes. So if you have a 20 ms Connection Interval, then the theoretical maximum for sending a message and receiving the ACK is 80 bytes (of data) in 40 mS (one Connection Interval to send, plus one Interval to receive the ACK). But trains leave whether full or not. So you might only send 20 bytes every 80 mS -- or less.

1.2 Slave latency

This is the number of connection intervals that the slave is allowed to skip. For example, if the connection interval is 20mSec and slave latency is 4, then if the peripheral wants to then it only needs to answer the master every 80mSec. Slave latency is useful if you want to mostly stay asleep, but then burst out data at a faster rate occasionally. In this case, the peripheral only needs to respond to the master every 80mSec to keep the connection alive, but if it has a lot of data then it can send data every 20mSec. Slave Latency is any value between 0 and 499, though it cannot exceed: ((supervisionTimeout / connInterval) – 1)

1.3 Connection Supervision Timeout

This parameter specifies the maximum amount of time that either the master or slave can go before receiving a link-layer packet. Both slave and master device maintain their own “Supervision timer”, which resets to zero every time a packet is received. If supervision timer ever reaches the supervision timeout, the device considers the connection lost, and exits the connection state (returning to the advertising, scanning, or standby state). The Connection Supervision Timeout is in the range of 100ms and 32.0s. It Must be larger than (1 + slaveLatency) * (ConnInterval).

1.4 Packets Per Connection Interval

This is the number of packets that can be sent in each connection interval. According to the Bluetooth specification, the maximum packets per connection interval is 6, but iOS limits this to 4.

Connection Parameters between two modules

Here you may set the Connection Parameters to their maximum values, with a Connection Interval as low as 7.5mSec to give the highest throughput.

2. Connection Parameters for iOS

Recall earlier where we mentioned that the maximum achievable data rate is device dependent. According to the Apple Bluetooth Accessory Design Guidelines for Apple Products, the connection parameters are restricted as follows:

- Interval Max * (Slave Latency + 1) ≤ 2 seconds Interval Min ≥ 20 ms (not 7.5mSec)
- Interval Min + 20 ms ≤ Interval Max Slave Latency ≤ 4 connSupervisionTimeout ≤ 6 seconds
- Interval Max * (Slave Latency + 1) * 3 < connSupervisionTimeout

If Bluetooth Low Energy HID is one of the connected services of an accessory, connection interval down to 11.25 ms may be accepted by the Apple product. Theoretical maximum throughput for iOS is: 80B per 40mSec, which results in 2kB/Sec. Using Notifications / Indications may increase this.

3. Connection Parameters for Android

Connection Parameters for Android are device dependent, but some devices can support a Connection Interval as short as 7.5mSec.

4. Password

"0000" can not use by password.
if you input password as "0000", error message will be sent.
(PEN_ILLEGAL_PASSWORD_0000)

/End of Document/