BluPrints
Thermal Receipt Printer
Developer Guide
(2 Inch-58/mm)

Aadharshila Mobility Solutions Pvt. Ltd.

E - 22, Sector – 51 NOIDA - 201301 Uttar Pradesh
Web : http://www.bluprints.in/

VERSION CONTROL: 6.0 / June 2018

PLEASE NOTE THAT THIS SDK VERSION, NAMELY, AEM_SDK6.0 IS APPLICABLE FOR AEM SCRYBE PRINTER FIRMWARE VERSIONS 5.0 AND ABOVE.

DOCUMENT NAME: ANDROID DEVELOPER GUIDE RELEASE

DATE: 4-June-2018

SDK SUPPORTED: AEM_ SDK6.0 FIRMWARE

SUPPORTED: 5.0 AND ABOVE

TABLE OF CONTENTS

SDK (Software Development Kit) for ANDROID

This document describes the use of AEM SCRYBE Thermal Printer SDK for Bluetooth, USB and WiFi Operations. This SDK provide an Interface between an Android Application and the AEM Thermal Printer namely SCRYBE. The SDK is android based, and requires at least 10 level of android SDK and 1.6 java compiler. It comprises of the following Interfaces, Classes and functions.

com.aem.api consists of Classes – AEMPrinter, AEMScrybeDevice, IAEMCardScanner and AEMWifiPrinter. The USB SDK Comprises of USBController class.

AEMPrinter: AEMPrinter class handles all the Bluetooth related functions. These include:

1. **setFontType: public void setFontType(byte FONT) throws IOException**

   In order to set any of the above four fonts in the printer, you need to call this function and pass the desired font.

   Normal Font: FONT_NORMAL
   Calibri Font: FONT_001
   Tahoma Font: FONT_002
   Verdana Font: FONT_003

2. **setLeftAlign: public void setLeftAlign() throws IOException**

   In order to set the text alignment to left ,you need to call this function.

3. **setCenterAlign: public void setCenterAlign() throws IOException**

   In order to set the text alignment to center ,you need to call this function.

4. **setRightAlign: public void setCenterAlign() throws IOException**

   In order to set the text alignment to center ,you need to call this function.

5. **setTextUnderline: public void setTextUnderline() throws IOException**

   In order to underline the text ,you need to call this function.

6. **setTextDoubleHeight: public void setTextDoubleHeight() throws IOException**

   In order to set the text to double height ,you need to call this function.

7. **setTextDoubleWidth: public void setTextDoubleWidth() throws IOException**

   In order to set the text to double width ,you need to call this function.

8. **setFontNormal: public void setFontNormal() throws IOException**

In order to set the normal font ,you need to call this function.

9. **setFontCalibri: public void setFontCalibri() throws IOException**

   In order to set the font to calibri ,you need to call this function.

10. **setFontTahoma: public void setFontTahoma() throws IOException**

    In order to set the font to tahoma ,you need to call this function.

11. **setFontVerdana: public void setFontVerdana() throws IOException**

    In order to set the font to verdana ,you need to call this function.

12. **setFontSize: public void setFontSize(byte DIMENSION) throws IOException**

    In order to set the font size to Double width or double height of a particular line in the printer, you need to call this function and pass the desired parameter of Double Width or Double Height. Please note that double width functionality will not work properly in case of 2 inch 48 character per line Printer. Alternatively, you can use the sendByte function and simply pass

    the desired parameter DOUBLE_WIDTH or DOUBLE_HEIGHT as defined below.

    DOUBLE_WIDTH = 0X04
    DOUBLE_HEIGHT = 0X08

3. **sendByte: public void sendByte(byte bt) throws IOException**

   In order to send any of specific byte to the printer via Bluetooth, you need to call this function and pass the desired byte.

4. **sendByteArray: public void sendByteArrayBT(byte[] byteArr) throws IOException**

   In order to send any of specific byte array (sequence) to the printer via Bluetooth, you need to call this function and pass the desired Byte Array.

5. **printInNegative: public void printInNegative() throws IOException**

   NEGATIVE_CHAR = 0X0E

   In order to print any of specific line in Negative Text Mode you need to call this function. Alternatively, you can use the sendByte function and simply pass the value of NEGATIVE_CHAR as defined above.

6. **enableUnderline: public void enableUnderline() throws IOException**

   UNDERLINE = 0X15

   In order to underline any of specific line you need to call this function. Alternatively, you can use the sendByte function and simply pass the value of UNDERLINE as defined above.

7. **setLineFeed: public void setLineFeed(int noOfFeeds) throws IOException**

   LINE_FEED = 0X0A

   In order to print specific number of blank lines or line feeds you need to call this function. Alternatively, you can use the sendByte function and simply pass the value of LINE_FEED as defined above. You can call sendByte multiple times for printing multiple blank lines.

8. **setCarriageReturn: public void setCarriageReturn() throws IOException**

   CARRIAGE_RETURN = 0X0D

   In order to print a single blank line you need to call this function. Alternatively, you can use the sendByte function and simply pass the value of CARRIAGE_RETURN as defined above. You can call sendByte multiple times for printing multiple blank lines.

9. **print: public void print(String text) throws IOException**

   This function is used to print a specified text (in the form of a string) to the printer.

10. **printHindi: public void printHindi(String text) throws IOException**

    In the specific Hindi Font Printer Models, this function is useful in printing Devnagri Font text for generating Bills / Receipts in HINDI (in the form of a Unicode string). This function is only usable in SCRYBE Printer Models that have firmware version ending in 'H'. Use of this function by passing a normal English (or numeric/alphanumeric) text will only print those charaters normally.

General Guidelines for Text Printing

1. **Send the command to print the various fonts and styles.**
2. **If multiple commands are given then last command will be effective only.**
3. **Location to apply – at the start of the line to print**

Following Printing commands are for formatting the text of a single line.

Command Type	Packets	Hex Value	Command
Normal Font	CTRL+F	0x06	ESC ! 0x00
Font 001: Calibri Font	CTRL+C	0x03	ESC ! 0x01
Font 002: Tahoma Font	CTRL+T	0x14	ESC ! 0x02
Font 003: Verdana Font	CTRL+V	0x16	ESC ! 0x03
Double Width	CTRL+D	0x04	ESC ! 0x20
Double Height	CTRL+H	0x08	ESC ! 0x10
Underline	CTRL+U	0x15	ESC ! 0x80
Line Feed (New Line)	ENTER	0x0A	
Negative Char	CTRL+N	0x0E	
Carriage Return(New Line)			
Left Align			ESC a 0x00
Center Align			ESC a 0x01
Right Align			ESC a 0x02

Rules to apply commands:

1. **You can change the fonts, as well as Double width, Double height, Negative and underline characters. Command should be sent before the line to print.**
2. **Should not repeat the command more than one time at the start of line.**
3. **Line feed of carriage return should be kept in the end of line if length of text is less than 32. E.g. To print underline. First send 0x15 (in hex) and then text, say,**

"Hello world" and then carriage Return. Then it will print Hello world.

Functions for Non-Text Printing

11. **printBarcode: public void printBarcode(String barcodeData, BARCODE_TYPE Btype, BARCODE_HEIGHT bHeight) throws IOException**

    BARCODE_TYPE_CODE39 = 0X45

    This function is used to print a 2-Dimensional Barcode on the Printer, generated automatically according to the string passed to this function. The string should be of Capital English Letters or Numerals. The maximum characters in the strings can by up to 11. Under Barcode Type, the Printer supports BARCODE_TYPE_CODE39. The Barcode Height supported is DOUBLEDENSITY_FULLHEIGHT.

    The following packet is generated internally within this function and sent to the printer:

    barcodePacket[0] = 0x1D;
    'GS' barcodePacket[1] =
    0x6B; 'k'

    barcodePacket[2] = BARCODE_TYPE_CODE39; 0x45 barcodePacket[3] =
    (byte) (barcodeBytes.length + 2); //length of barcode data barcodePacket[4]
    = 0x2A;

    barcodePacket[5] to barcodePacket[length of barcode string] =
    BarcodeBytes; barcodePacket[Length of barcode string + 1] = 0x2A;

12. **printImage: public void printImage(Bitmap originalBitmap) throws IOException**

    This function is used to print an Image in the form of a Raster Image, based on the standard ESC/POS Raster Image command set of GS v protocol. You need to pass the bitmap of the desired image to be printed. Please note that this function can be used to print QR Codes as well, by first generating the QR code from a given string and thereafter, sending its bitmap to the PrintImage Function. The source code for generating the QR Code has been provided at the end of this document.

13. **printTextAsImage: public void printTextAsImage(String TextToConvert)throws IOException**

This function is used to print any text (multilingual, Unicode type characters, etc) in the form of a Raster Image. You need to pass the String that you need to be printed as an image. The main utility of this function is that the user can easily input any multilingual string, that can be printed as is on the printer, so as to enable printing in any language irrespective of the fonts. (Printing characters of Urdu, Gujrati, Arabic, Oriya, Tamil, Tamil, Telugu, etc.). The benefit of this function is that the print is so clear and fast that there is no perceptible difference to an end user in understanding whether this text has been printed in the form of text or in the form of an image.

14. **printBitImage: public void printBitImage(Bitmap originalBitmap, Context context, byte image_alignment)throws IOException**

IMAGE_LEFT_ALIGNMENT = 0x6C;
IMAGE_CENTER_ALIGNMENT = 0x63;
IMAGE_RIGHT_ALIGNMENT = 0x72;

This is a deprecated function that has only been kept for backward compatibility.

PrintBitImage fiunction uses the standard ESC * algorithm for printing of a bitmap The maximum image size should be in the following range: - 355 X 500 (WxH) pixels.

For image printing, you need to half the pixel size of height of an image to get the desired width and height.

E.g.: - Suppose you need to print a logo of dimensions 355 X 300 (WXH) pixels, then half the size of the height of an image i.e. 150 pixel for one time in your code by scaling function (as explained below). Width will remain same.

Bitmapscaled_bitmap = bitmap.createScaledBitmap(bitmap, 355, 150, false); 355 pixel= Width (Original Width)

150 pixel= Height (Height will get double i.e. 300 pixels as original height of an image while printing).

Likewise, you can print the logo of desired

dimensions. Note: - Maximum Width size is 355 pixels

Maximum Height size is 500 pixels

AEMScrybeDevice:

This class is used to instantiate a BLUETOOTH SCRYBE Device, containing the Context, the Bluetooth Adapter, Bluetooth Device and Bluetooth Socket. An object of AEMScrybeDevice once instantiated, is capable of returning the AEMPrinter object that has been described previously. This class has the following functions:

AEMScrybeDevice

1. **Constructor: For Creating the Object: public AEMScrybeDevice (IAemScrybeimpl)**
2. **startDiscover: public void startDiscover(Context iContext)**

   By calling this method, a list of local Bluetooth devices will be returned.
3. **pairDevice: Public String pairDevice(String printerName)**

   Before pairing any printer by name, first call the method startDiscover(Context iContext)

   and get the result in the method public void

   onDiscoveryComplete(ArrayList<String>aemPrinterList) of the Interface IAemScrybe.

   Example:

```
AEMScrybeDevice m_AemScrybeDevice = new AEMScrybeDevice (newIAemScrybe()
    {
@Override
Public void onDiscoveryComplete(ArrayList<String>aemPrinterList)
 {
// TODO Auto-generated method stub
}
});
```

   Now call the method public String pairDevice(String printerName) which returns a String which may be:

   NOT_SCANNED when you call this method before scanning.

   DEVICE_NOT_FOUND when the printer is not found. PAIRED when the

   device is successfully paired FAILED_TO_PAIREDwhen the device is failed to

   paired

4. **connectToPrinter: public Boolean connectToPrinter(String printerName) throws IOException**

   By calling this method, printer gets connected.
5. **disConnectPrinter: public Boolean disConnectPrinter() throws IOException**

   By calling this method, the connected printer gets disconnected.

6. **getCardReader: public CardReader getCardReader(IAemCardScannerreaderimpl)**

   By calling this method, you can create the object of the class CardReader by passing the reference of the class which implements IAemCardScanner Interface.

7. **AEMPrinter: public AEMPrinter getAemPrinter()**

   By calling this method, you can create the object of the class AEMPrinter.

8. **ArrayList: public ArrayList<String>getPairedPrinters()**

   By calling this method, a list of paired printers is returned.

9. **getSDKVersion: public String getSDKVersion()**

   By calling this method, SDK version is obtained.

10. **BtConnStatus: public Boolean BtConnStatus()**

    This method returns True if the Printer is already connected on Bluetooth (Bluetooth connection is already alive). It returns False if the connection status is False, i.e. if Bluetooth socket is disconnected.

IAemCardScanner:

1. **onScanMSR: public void onScanMSR(String buffer, CARD_TRACK cardtrack)**

   This method will provide the MSR card data (buffer) with track type (Track1 or Track 2). You can decode the MSR card data by calling the method:-

   publicCardReader.MSRCardDatadecodeCreditCard (String buffer, CARD_TRACK track) of the Class CardReader.

2. **onScanDLCard: public void onScanDLCard(String Buffer)**

   This method will provide the DL card data. You can decode the DL card data (buffer) by calling the method:-

   publicCardReader.DLCardDatadecodeDLCardData(String buffer) of the Class CardReader.

3. **onScanRCCard: public void onScanRCCard(String Buffer)**

   This method will provide the RC card data. You can decode the RC card data (buffer) by calling the method:-

   publicCardReader.RCCardDatadecodeRCCardData(String buffer) of the Class CardReader.

4. **onScanRFD: public void onScanRFD(String Buffer) This method is for RFID data.**

5. **onScanPacket: public void onScanPacket(String Buffer) This method is for getting various responses:-**

Response Name	Type	Explanation
No Paper	Error	When there is no paper roll
Printer Head High Temp	Error	Printer mechanism is not connected / High temp of Printer Head
Low Battery	Error	Battery is low
Battery Charging	Notification	Battery is charging
Paper OK	Response	Paper roll is present

FUNCTIONS FOR WIFI PRINTER

AEMWifiPrinter:

This Class is for communicating with AEM SCRYBE Wifi Series Printers. This class does the entire processing for various functions such as generating the bytes for Barcode and Image.

In order to send these bytes out on a physical TCP/IP Socket, there needs to be a separate Client class to be maintained by the user in the main Java Application Project. This class has been purposely kept outside from the SDK so that the user can customize the messages thrown by the interfaces and accordingly process the commands. For ease of use the AEMWifiPrinter Class has been provided so as to carry out complex processing of Images and Barcode packet generation. Since all these functions return array of Bytes, these bytes can be easily send out to the client socket by the user. MyClient,java class has been provided as extending AsyncTask for the user to override the DoinBackground and OnPostExecute functions according to the functionality as needed.

The functions of AEMWifiPrinter class are described below:

1. **printBarcode: public byte[] printBarcode(String barcodeData, BARCODE_TYPE Btype, BARCODE_HEIGHT bHeight) throws IOException**

   BARCODE_TYPE_CODE39 = 0X45

   This function is used to return the byte array for printing a 2-Dimensional Barcode on the Printer, generated automatically according to the string passed to this function. The string should be of Capital English Letters or Numerals. The maximum characters in the strings can by up to 11. Under Barcode Type, the Printer supports BARCODE_TYPE_CODE39. The Barcode Height supported is DOUBLEDENSITY_FULLHEIGHT. The following packet is generated internally within this function and sent to the printer:

   barcodePacket[0] = 0x1D; 'GS'
   barcodePacket[1] = 0x6B; 'k'
   barcodePacket[2] = BARCODE_TYPE_CODE39; 0x45 barcodePacket[3] = (byte) (barcodeBytes.length + 2); //length of barcode data barcodePacket[4] = 0x2A;

   barcodePacket[5] to barcodePacket[length of barcode string] = BarcodeBytes;
   barcodePacket[Length of barcode string + 1] = 0x2A

2. **printImage: public byte[] printImage(Bitmap originalBitmap) throws IOException**

   This function returns the byte array for printing an Image in the form of a Raster Image, based on the standard ESC/POS Raster Image command set of GS v protocol. You need to pass the bitmap of the desired image to be printed. Please note that this function can be used to print QR Codes as well, by first generating the QR code from a given string and thereafter, sending its bitmap to the PrintImage Function. The source code for generating the QR Code has been provided at the end of this document.

3. **printTextAsImage: public void printTextAsImage(String TextToConvert)throws IOException This function returns the byte array for printing any text (multilingual, Unicode type characters, etc) in the form of a Raster Image. You need to pass the String that you need to be printed as an image. The main utility of this function is that the user can easily input any multilingual string, that can be printed as is on the printer, so as to enable printing in any language irrespective of the fonts. (Printing characters of Urdu, Gujrati, Arabic, Oriya, Tamil, Tamil, Telugu, etc.). The benefit of this function is that the print is so clear and fast that there is no perceptible difference to an end user in understanding whether this text has been printed in the form of text or in the form of an image.**

MyCLient Class for TCP/IP Socket for WIFI Connection:

```
public class MyClient extends AsyncTask<Void, Void, Socket> {
   String response = "";
   public Socket socket = null;
   public static final int BUFFER_SIZE = 2048;

   private PrintWriter out = null; private
   BufferedReader in = null; private String
   dstAddress = null; private int dstPort = 9100;
   private Context context; private String text;
   public static boolean wifiConnection=false;
   public static boolean connection=true;

   /**
    * Constructor with Host, Port and MAC Addres
```

```
  public void configSocket(String host)
{
    this.dstAddress = host;
    this.dstPort = 9100;

  }

  @Override
  protected Socket doInBackground(Void...
  arg0) {

    String message = ""; int charsRead = 0;
    char[] buffer = new
    char[BUFFER_SIZE];

    try {
      this.socket = new Socket(dstAddress,
      dstPort);
      Log.i("SocketConnection",this.socket
      +"");

      out = new
      PrintWriter(s
      ocket.getOutp
      utStream());
      in = new
      BufferedRead
      er(new
      InputStreamReader(socket.getInputSt
      ream())); wifiConnection = true;
    } catch (UnknownHostException e) {
      connection = false;
      e.printStackTrace();
      this.response =
    "UnknownHostException: " +
    e.toString(); Log.e("Exception",
    "UnknownHostException: " +
    e.getMessage()); } catch (IOException e)
    {
      this.response =
    "IOException: " +
    e.toString();
    Log.e("Exception",
    "IOException: " +
    e.getMessage()); } finally
    {
      if (socket != null) {
```

```
    }
    return this.socket;
}

@Override
protected
    void
    onPostEx
    ecute(Soc
    ket
    result) {
    try{ if
    (result!=n
    ull)
    {
            ((MainActivity)context).onSuccess(thi
            s.response,this.socket,dstAddress,dstP
            ort,out);
    }
    else
    {
       ((MainActivity)context).onError(this.res
       ponse);
    }
    //super.onPostExecute(result);
}
    catch (Exception ec){
    ((MainActivity)context).onError(this.respo
    nse);
    }}

public void
    sendDataOnSocket(String
    message,PrintWriter out) { if
    (message != null) {
       out.write(message);
       out.flush();
    }
}

public void
    sendBytesOnSocket(byte[]
    btPkt, int numBytes,Socket
    socket) { try {
       socket.getOutputStream().write(btPkt, 0,
       numBytes);
    } catch (IOException e) {
      //  showAlert("error in outputstream " +
      e.toString());
       ((MainActivity)context).onError(this.res
       ponse);
    }
```

```
}

public int disConnectWithServer()
{
   int retVal = 0;
          if (socket != null)
          {if (socket.isConnected()){









                        try { socket.close(); while(true)
                              {
                                if(socket.isClosed() == true)
                                {
                                          retVal = 1; return retVal;
                                }
                              }

          } catch (IOException e) { ((MainActivity)context).onError(this.response);
                              }
                          }
                      }
                          return retVal;
                  }
          }
```

Packet Protocol

The communication is done in either of three modes:

a. **Bluetooth Mode**

b. **USB Mode**

c. **WiFI Mode**

Device reads the packet, which is connected to host application, or you can connect through any one of the above three modes. Device reads the packet and parses it, then processes the command.

Bluetooth (Host) to Printer Commands:

To send commands to printer first add [ESC] i.e. 0x1B in starting and then append the query packet. Host application to Printer command:

ESC(0x1B)	Start delimiter (0x7E)	BP*	Separator	Packet Type	Separator	Packet Data	End delimiter (0x5E)
1 Byte	1 Byte	2 Byte	1 Byte	2 Byte	1 Byte	-	1 Byte

Host App to Printer Command

***BP: Bluetooth (host) to Printer**

Command Type	Packets
*Get Printer Status :	[ESC]~BP|GET| PRN_ST^
Get Printer Configuration:	[ESC]~BP|GET| CONFIG ^
Get Battery Status	[ESC]~BP|GET| BAT_ST ^
Get Printer Version Status:	[ESC]~BP|GET| PRNVER ^ (Returns Ver 1.1.3.21 for Bluetooth and 1.1.3.20 for Non BT)
Printer Power off	[ESC]~BP|PRN| PWROFF ^
Test LED	[ESC]~BP|TST| TSTLED ^
Test Print	[ESC]~BP|TST| PRINT ^
Set Printer Configuration	[ESC]~BP|CON|Name,Password,FontType,EnableEncryption,MSRTimeOut,ICTimeOut^

BT to Printer

Responses from Printer:

Packet Structure:

Start delimiter (0x7E)	PB*	Separator	Packet Type	Separator	Response	End delimiter (0x5E)
1 Byte	1 Byte	1 Byte	1 Byte	1 Byte	-	1 Byte

Packet Structure

***PB: Printer to Android (Host)**

Response table:

Response Name	Type	Packets
No Paper	Error	~PB|PRN|NOPAPER^
Printer Head High Temp:	Error	~PB|PRN|HGHTEMP^
Printer Platen	Error	~PB|PRN|PLTNREL^
Wrong Packet:	Error	~PB|PRN|PKTEROR^
MSR Swiped	Notification	~PB|CRD|MSR_SWP^
Low Battery	Error	~PB|BAT|LOWBATT^
Battery Charging	Notification	~PB|BAT|BATCHG1^
Battery Not Charging	Notification	~PB|BAT|BATCHG0^
Battery Charge Status	Notification	~PB|BAT|BAT%3d%%^
Printer Configuration	Response	~PB|CON| Name, Password, FontType, EnableEncryption, MSRTimeOut, ICTimeOut^
Printer Power Off	Response	~PB|PRN|PWR_OFF^

Response Table

Bit Map Image Print – Note that this method of Printing is only maintained for the purpose of Backward compatibility. The Faster, easier, clearer and more standard way of printing is by using PrintImage function and passing it a Bitmap object.

1. **Select the image either from PC or from mobile's SD card.**
2. **Convert the image into monochrome bmp.**
3. **Resize the image to fit into the printer paper area if it is exceeding**
4. **Now read the processed image through fileinputstream and save it into byte array.**
5. **Make the packet of image and then send it to printer over outputstream.**

Bit-Image Packet Structure:

ESC(0x1B)	(0x2A)	Mode m	Alignment Of Image (a)	Width of Image (w)*	Height of Image (h)*	Image Byte Array (D1 to Dn)

Bit-Map Image Structure

Mode M (In Hex)	Mode M Description	Vertical Dot	Horizontal Dot	Max Dot/Line
0x64	Single width single height	1	1	384 (48 bytes)
0x65	Single width double height	2	1	384
0x66	Single width triple height	3	1	384
0x67	Single width Quad. height	4	1	384
0x68	Double width single height	1	2	192 (24 bytes)
0x69	Double width double height	2	2	192
0x6A	Double width triple height	3	2	192
0x6B	Double width Quad. Height	4	2	192
0x6C	Triple width single height	1	3	128 (16 bytes)
0x6D	Triple width double height	2	3	128
0x6E	Triple width triple height	3	3	128
0x6F	Triple width Quad. height	4	3	128

Mode Table

Alignment (a)	Hex Value

Left align	0x6C
Centre align	0x63
Right align	0x72

QR Code Generation Function:

```
public void onPrintQRCode(View v) throws WriterException, IOException
  {
      Writer writer = new QRCodeWriter(); String text= editText.getText().toString();
      String finalData = Uri.encode(text, "UTF-8"); showAlert("QR " + text);
 try
 {
    BitMatrix bm = writer.encode(finalData,BarcodeFormat.QR_CODE, 300, 300); Bitmap bitmap =
    Bitmap.createBitmap(300, 300, Config.ARGB_8888); for(int i = 0; i < 300; i++)
    {
        for(int j = 0; j < 300; j++)
        {
                bitmap.setPixel(i, j, bm.get(i, j) ? Color.BLACK: Color.WHITE);
        }
    }

    Bitmap resizedBitmap = null; int numChars = glbPrinterWidth;

    resizedBitmap = Bitmap.createScaledBitmap(bitmap, 384, 384, false);
        m_AemPrinter.printImage(resizedBitmap);
 }
  catch(WriterException e)
  {
        showAlert("Error WrQR: " + e.toString());
  }
 }
```

SDK FOR ACCESSING ANDROID USB

USBSDK:

The USB SDK for AEM Printers contains the basic USBController class for accessing the USB interface of an ANDROID Device.

usbController Class:

1. **Constructor: public UsbController (Activity ParentActivity, Handler mHandler)**

   The constructor requires a pointer to the current activity (this) and an object of handler.

2. **getDev: public usbDevice getDev (int VID, int PID)**

This function returns the object of the usbDevice based on the USB product's Vendor ID and Product ID passed to it. For AEM SCRYBE USB Printers, the VID and PID are as follows:

0x0416: VID: AEM Printer
0x5011: PID: AEM Printer

3. **isHasPermission: Boolean isHasPermission (UsbDevice dev)**

   This function returns TRUE if the connected USB device on the USB Port of the Android device has obtained the permission for USB Access. This must be checked prior to sending any command or data to the USB Port. An object of the USBDevice needs to be passed to it.

4. **getPermission: void getPermission (UsbDevice dev)**

   This function obtains the permission for USB Access. This must be obtained after isHasPermission returns FALSE, and prior to sending any command or data to the USB Port. An object of the USBDevice needs to be passed to it.

5. **sendByte: void sendByte(byte[] bytesPkt, usbDevice dev)**

   This function sends an array of bytes out to the USB Port. An Array of bytes to be sent along with an object of the USBDevice needs to be passed to it.

6. **sendMsg: void sendMsg(String textToSend, String CharSet, usbDevice dev)**

   This function sends a text String out to the USB Port. An string of text to be sent, the character set along with an object of the USBDevice needs to be passed to it.

   For English, the CharSet is "GBK". E.g.: usbCtrl.sendMsg("Hello", "GBK", dev);

7. **handleMessage: void handleMessage(Message msg) This is to be overridden in the following manner:**

```
private final Handler mHandler = new Handler() { @Override
public void handleMessage(Message msg)
{
switch (msg.what) {
case UsbController.USB_CONNECTED:
Toast.makeText(getApplicationContext(),getString(R.string.msg_getpermission),
Toast.LENGTH_SHORT).show();

break; default: break;
}
}
};
```

8. **close: void close(String void)**

   This function closes the USB Connection.

Aadharshila Mobility Solutions Pvt. Ltd
E - 22, Sector – 51 NOIDA - 201306
uttarPradesh
Web : http://www.bluprints.in/