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

Aadharshila Mobility Solutions Pvt. Ltd.
Web: http://www.BluPrints.in/

VERSION CONTROL: 6.2 / March 2021

PLEASE NOTE THAT THIS SDK VERSION, NAMELY, BLUPRINTS_SDK6.1 IS APPLICABLE FOR BLUPRINTS SCRYBE PRINTER FIRMWARE VERSIONS 8.0 AND ABOVE.

DOCUMENT NAME: ANDROID DEVELOPER GUIDE RELEASE DATE: 24-March-2021

SDK SUPPORTED: BLUPRINTS_ SDK6.1 FIRMWARE SUPPORTED: 8.0 AND ABOVE

## TABLE OF CONTENTS

# SDK (Software Development Kit) for ANDROID

This document describes the use of BLUPRINTS SCRYBE Thermal Printer SDK for Bluetooth and Wi-Fi

Operations. This SDK provide an Interface between an Android Application and the BLUPRINTS Thermal

Printer namely SCRYBE. The SDK is android based, and requires at least 10 level of android SDK and

java compiler. It comprises of the following Interfaces, Classes and functions. Com.BluPrints.api consists of Classes – BLUPRINTSPrinter, BLUPRINTSScrybeDevice, and BLUPRINTSWifiPrinter.
BLUPRINTSPrinter: BLUPRINTSPrinter class handles all the Bluetooth related functions. These include:

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

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.

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.

sendByteArray: public void send ByteArrayBT(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.

printinNegative: public void prinInNegative() 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.

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.

setLineFeed: public void setLineFeed(int noOfFeed) 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.

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.

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.

General Guidelines for Text Printing

Send the command to print the various fonts and styles.
If multiple commands are given then last command will be effective only.
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 |
|---|---|---|
| Normal Font | CTRL+F | 0x06 |
| Font 001: Calibri Font | CTRL+C | 0x03 |
| Font 002: Tahoma Font | CTRL+T | 0x14 |
| Font 003: Verdana Font | CTRL+V | 0x16 |
| Double Width | CTRL+D | 0x04 |

| | | |
|---|---|---|
| Double Height | CTRL+H | 0x08 |
| Negative Char | CTRL+N | 0x0E |
| Underline | CTRL+U | 0x15 |
| Line Feed (New Line) | ENTER | 0x0A |
| Carriage Return(New Line) | ENTER | 0x0D |

Rules to apply commands:

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.

Should not repeat the command more than one time at the start of line.

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 Barcode, BARCODE_TYPE Btype, BARCODE_HIEGHT bHieght) 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
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;

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.

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.)

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.

printBitImage: public void printBitImage (BitMap original ,Context context, byte imageAlignment) 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

BLUPRINTSScrybeDevice:

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

BLUPRINTSScrybeDevice

Constructor: For Creating the Object: public BLUPRINTSScrybeDevice (IBluPrintsScrybeimpl)
startDiscover: public void startDiscover(Context iContext)

By calling this method, a list of local Bluetooth devices will be returned.
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>BluPrintsPrinterList) of the Interface IBluPrintsScrybe. Example:

```
BLUPRINTSScrybeDevice m_BluPrintsScrybeDevice = new BLUPRINTSScrybeDevice (newIBluPrintsScrybe()
{
@Override
Public void onDiscoveryComplete(ArrayList<String>BluPrintsPrinterList)
{
// 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

connectToPrinter: public Boolean connectToPrinter(String printerName) throws IOException By calling this method, printer gets connected.

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

BLUPRINTSPrinter: public BLUPRINTSPrinter getBluPrintsPrinter()
By calling this method, you can create the object of the class BLUPRINTSPrinter.

ArrayList: public ArrayList<String>getPairedPrinters() By calling this method, a list of paired printers is returned.

getSDKVersion: public String getSDKVersion() By calling this method, SDK version is obtained.

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

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

BLUPRINTSWifiPrinter:

This Class is for communicating with BLUPRINTS 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 BLUPRINTSWifiPrinter 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 BLUPRINTSWifiPrinter class are described below:

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;

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

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.

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;

*/
public MyClient(String text,Context context) { this.dstAddress = text; this.dstPort = 9100; this.context = context;
}
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(socket.getOutputStream()); in = new BufferedReader(new InputStreamReader(socket.getInputStream())); wifiConnection = true;
} catch (UnknownHostException e) {
// TODO Auto-generated catch block wifiConnection = false; connection = false; e.printStackTrace(); this.response = "UnknownHostException: " + e.toString(); Log.e("Exception", "UnknownHostException: " + e.getMessage()); } catch (IOException e) { wifiConnection = false; connection = false; e.printStackTrace(); this.response = "IOException: " + e.toString(); Log.e("Exception", "IOException: " + e.getMessage()); }
finally { if (socket != null) {
}
```

```
}
return this.socket;}
@Override
protected void onPostExecute(Socket result) { try{ if (result!=null)
{
((MainActivity)context).onSuccess(this.response,this.socket,dstAddress,dstPort,out);}
else{
((MainActivity)context).onError(this.response);}
//super.onPostExecute(result);}
catch (Exception ec){ ((MainActivity)context).onError(this.response);}}
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.response);
}
}
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 ProtocolThe
communication is done in either of three modes:
Bluetooth Mode
Wi-Fi 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:

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 | Packe t Type | Separator | Packet Data | Delimite r (0x5E) |
|---|---|---|---|---|---|---|---|
| 1 Byte | 1 Byte | 2 Byte | 1 Byte | 2 Byte | 1 Byte | - | 1Byte |

Host App to Printer Command

*BP: Bluetooth (host) to Printer:

| Command Type | Packets |
|---|---|
| *Get Printer Status : | [ESC]~BP\|GET\| PRN_ST^ |
| Get Printer Configurat | [ESC]~BP\|GET\| CONFIG ^ |
| Get Battery Status | [ESC]~BP\|GET\| BAT_ST ^ |
| Get Printer Version Sta | [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,ICT imeOut^ |

BT to Printer
Responses from Printer:
Packet Structure:

| Start Delimiter (0x7E) | PB* | Separato r (1) | Packet Type | Separato r (1) | Response | End delimiter (0x5E) |
|---|---|---|---|---|---|---|
| 1 Byte | 2 Byte | 1 Byte | 3 Byte | 1 Byte | - | 1 Byte |

*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^ |

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

| Mode M (In Hex ) | Mode M Description | Vertical Dot | Horizontal Dot | Max Dot/Line |
|---|---|---|---|---|
| 0x08 | double height | 1 | 1 | 384 (48bytes) |
| 0x04 | double width | 2 | 1 | 384 |

| Alignment (a) | Hex Value |
|---|---|
| Left align | 0x6C |
| Centre align | 0x63 |
| Right align | 0x72 |

Alignment

QR Code Generation Function:

```
Public void onPrintQRCodeRaster (View v) throws WriterException, IOException {

String text= editText.getText().toString(); if(text.isEmpty()){
showAlert("Write Text To Generate QR Code");
}
else {
Writer writer = new QRCodeWriter();
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;
```

```
if(numChars == 32){
resizedBitmap = Bitmap.createScaledBitmap(bitmap, 384, 384, false);
m_BluPrintsPrinter.printImage(resizedBitmap); m_BluPrintsPrinter.setCarriageReturn();
m_BluPrintsPrinter.setCarriageReturn(); m_BluPrintsPrinter.setCarriageReturn();

}else {
resizedBitmap = Bitmap.createScaledBitmap(bitmap, 384, 430, false);
m_BluPrintsPrinter.printImageThreeInch(resizedBitmap);
m_BluPrintsPrinter.setCarriageReturn(); m_BluPrintsPrinter.setCarriageReturn();
m_BluPrintsPrinter.setCarriageReturn();
}

} catch (Writer Exception e) { showAlert("Error WrQR: " + e.toString());
}
```

## Unique Identification Authority of India (UIDAI)

**Introduction** :- Aadhaar authentication is the process wherein Aadhaar Number, along with other attributes, including biometrics, are submitted online to the Aadhaar system for its verification on the basis of information or data or documents available with it. During the authentication transaction, the resident's record is first selected using the Aadhaar Number and then the demographic/biometric inputs are matched against the stored data which was provided by the resident during enrolment/update process

**Registered Devices :-** Registered devices specification described in this document addresses the solution to eliminate the use of stored biometrics. It provides three key additional features compared to public devices

**Device identification :-** – every device having a unique identifier allowing traceability, analytics, and fraud management.

**Eliminating use of stored biometrics:-** biometric data is signed within the device using the device key to ensure it is indeed captured live. Then the Registered Device (RD) Service of the device provider must form the encrypted PID block before returning to the host application.

**A standardized RD Service provided by the device providers that is certified:-** This RD Service (exposed via Service interface defined in this spec) encapsulates the biometric capture, any user experience while capture (such as preview), and signing and encryption of biometrics all within it

Following is the sequence of typical operations using the registered devices:

AUA/Sub-AUA provided application starts in host machine.

Application does a RD Service discovery (see later sections for details).

When ready for biometric input capture, application connects the RD Service to initiate the PID creation (which contains digitally signed and encrypted biometrics).

When the RD Service detects a good capture, it does necessary processing / extraction, creates the signed biometric record (FMR, FIR, IIR, FID), forms the encrypted PID block, and give the encrypted PID block back to application along with other details including Device Info

Application obtains the encrypted PID block along with other information from the RD Service for calling Aadhaar authentication (see Aadhaar Authentication API 2.x specification for details).

**Device Identity :-** One of the key aspects about Registered devices is to identify the devices uniquely. Between the L0 and L1 there are difference in identification of the devices. L0 - In a L0 device we would use the following

Idhash SHA-256 of any internal physical ID that is used to recognize physical device (such as serial number). This should be read automatically without any user input. This ID is not expected to change during the life of that physical device. idHash MUST match what was sent during registration (see Register API call later)

**RD Service APIs**

All RD Services MUST provide the following standard APIs to ensure applications have a common way to interface with all Aadhaar compliant registered devices. This is a necessary condition for certification of registered devices.

**Interface Methods**

All RD Services must provide a standard interface having the following two methods and MUST work without ANY state stored within driver across calls.

// main capture call which takes PidOptions XML data as input and returns PidData XML as output String capture (String pidOptions); // utility method to obtain Device Info XML from the RD Service String getDeviceInfo (); A mechanism to discover the RD Service and invoke these methods are described in later sections of this document.

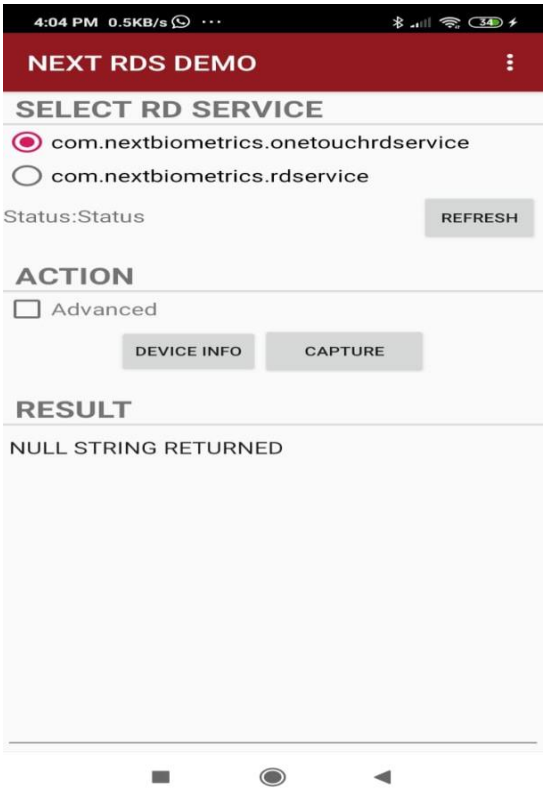
**Certified RD Services" Registry**

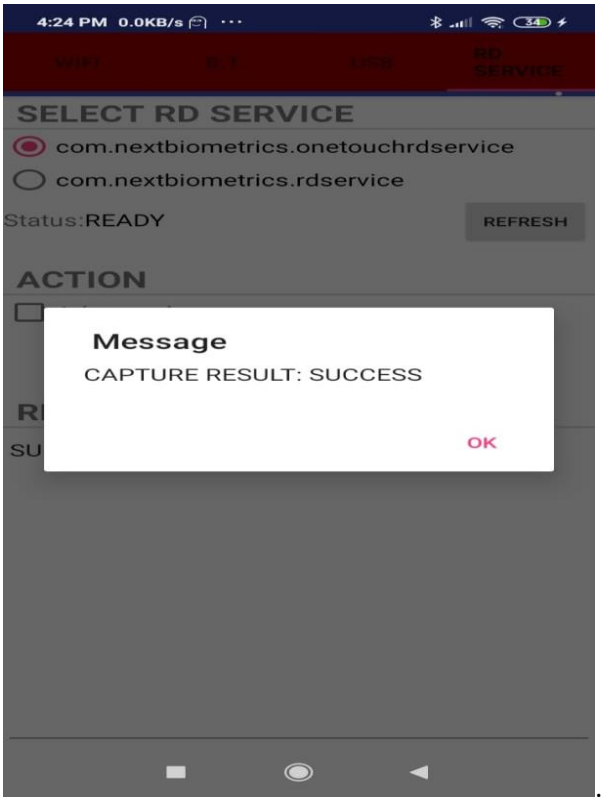
Certified RD services details are made available via a registry for applications to use. Applications are expected to check with this registry during RD service installation (if applications are managing these) and use. Information about RD services in various status are made available at the following URL: https://authportal.uidai.gov.in/devices/rdservice_registry.xml/

One provider has many models
One provider has one or more RD service per OS (many versions)
One RD service may handle many models Following is the XML for the service registry.

**CAPTURE BUTTON**

The capture call is a blocking call and only one client can call at any point in timeResponse to the capture call figure 1 & 2.




When the status RD service is ready . After clicking capture button the biometric should scan the put finger with result successful

**DEVICEINFO**

All connection are closed after the response.
The RD service will allow only one capture call at any given point in time.
In case a app calls capture when the RD service is in between the capture then it should return appropriate error code as per specification.

**Android Discovery & API Calling**

RD service should do the following actions: o Fingerprint devices should register "in.gov.uidai.rdservice.fp.INFO" and “in.gov.uidai.rdservice.fp.CAPTURE” o Iris devices should register "in.gov.uidai.rdservice.iris.INFO" and “in.gov.uidai.rdservice.iris.CAPTURE” o INFO should return the same DeviceInfo XML when called. o CAPTURE should return the same PidData XML when called.

Applications integrating on Android should do the following: o Browse providers and let user choose an appropriate RD service. Application may provide “remember default” and other options based on their needs. o Call “INFO” intent and verify the provider package against locally cached UIDAIRDServices registry to make sure only certified ones are being used. o Call "CAPTURE" intent to capture the encrypted biometrics.

**Register API** https://rdprod.uidai.gov.in/register **Input**
**<RegisterDevice ver =""ts=""txn="">**
**<Device dpid="" dc=" mi=""idHash="" PCHCertificate=">**

**</RegisterDevice>**

In case of L0 idHash must be SHA-256 of device serial number, that is used to recognize physical device.This should be read automatically without any user input. This ID is not expected to change during the lifetime of the physical device. idHash MUST match what was sent during registration

**Output**

**<RegisterDeviceResp ts=""txn=""code="" err="">**

**</RegisterDeviceResp>**

Integration NEXT RD SERVICE
Function Capture Button

```
protected void requestDiscovery() {
//   cleartext();
try {
Intent infoIntent = new Intent(INFO_INTENT); infoIntent.setPackage(NextSelectedpkg);
startActivityForResult(infoIntent, DISCOVERY_REQUEST_CODE);
} catch (Exception ee) { textStatus.setTextColor(Color.parseColor("#FF0000"));
textStatus.setText("RDService Not Installed."); download_Link();
}
}

protected void requestCapture(boolean auth) { cleartext();
try {
Intent capIntent = new Intent(CAPTURE_INTENT); pidOptXML = createPidOptXML();
capIntent.putExtra("PID_OPTIONS", pidOptXML); capIntent.setPackage(NextSelectedpkg);
if (auth) {
startActivityForResult(capIntent, AUTH_REQUEST_CODE);
} else {
startActivityForResult(capIntent, CAPTURE_REQUEST_CODE);
}

} catch (Exception ee) { download_Link();
}
}


private void download_Link() {
android.app.AlertDialog.Builder alertDialog = new
android.app.AlertDialog.Builder(this); alertDialog.setTitle("Please install NEXT
Biometrics LO Registered Device Service"); alertDialog.setMessage("Please install NEXT
Biometrics LO Registered Device Service from Play
Store ");
alertDialog.setIcon(R.drawable.app_icon); alertDialog.setPositiveButton("OK", new
DialogInterface.OnClickListener() {
public void onClick(DialogInterface dialog, int which) {
/* Intent intent = new Intent(Intent.ACTION_VIEW); intent.setData(Uri.parse(
"https://play.google.com/store/apps/details?id=com.nextbiometrics.rdservice"));

startActivity(intent);*/
}
});
alertDialog.show();
}
```

Function Device Info button

```
protected void requestInfo() { cleartext();try {
Intent infoIntent = new Intent(INFO_INTENT); infoIntent.setPackage(NextSelectedpkg);
startActivityForResult(infoIntent, INFO_REQUEST_CODE);
```

```
} catch (Exception ee) { download_Link();
}
}


private void download_Link() {
android.app.AlertDialog.Builder alertDialog = new
android.app.AlertDialog.Builder(this); alertDialog.setTitle("Please install NEXT
Biometrics LO Registered Device Service"); alertDialog.setMessage("Please install NEXT
Biometrics LO Registered Device Service from Play
Store ");
alertDialog.setIcon(R.drawable.app_icon); alertDialog.setPositiveButton("OK", new
DialogInterface.OnClickListener() {
public void onClick(DialogInterface dialog, int which) {
/* Intent intent = new Intent(Intent.ACTION_VIEW); intent.setData(Uri.parse(
"https://play.google.com/store/apps/details?id=com.nextbiometrics.rdservice"));

startActivity(intent);*/
}
});
alertDialog.show();
}


Funtion Authentication Button


btnAuth.setOnClickListener(new Button.OnClickListener() { @Override
public void onClick(View v) { requestDiscovery();
if (!envSel.getSelectedItem().toString().toUpperCase().equals("STAGING")) {
showMessageDialogue("This tool supports authentication in STAGING environment only",
"Select Environment->STAGING"); return;
}
String aadh_No = txtAadharNo.getText().toString().trim();
if (aadh_No != null && !aadh_No.isEmpty() && aadh_No.length() == 12) {
requestCapture(true);
} else {
showMessageDialogue("Please Enter Valid Aadhar No", "Error");
}
}
});
```

If you want to know any further details please view Aadhaar_Registered_Devices_2_0_4