SunmiPrinter Developer Docs

Revision Records

Version	Update date	Contents
1.0.0		The original version
1.1.170301	2017/03/01	Added c of image size; Added descriptions of whether there is a hardware query interface.
1.1.170315	2017/03/15	Added AIDL interface of the cutter; Added AIDL interface to open the cash drawer; Added AIDL interface to obtain the number of times the cutter has been used; Added AIDL interface to obtain the number of times the cashing drawer has been opened.
1.1.170322	2017/03/22	Added AIDL transaction printing interface with feedback; Added callback of transaction printing's result feedback; Modified descriptions of barcode format.
1.1.170329	2017/03/29	Added descriptions of modifying character sets.
1.1.170615	2017/06/15	Added descriptions of transaction printing.
1.1.170726	2017/07/26	Modified descriptions of AIDL calling.
1.1.170802	2017/08/02	Added descriptions of T1's black mark printing.
1.1.170803	2017/08/03	Added descriptions of T1's black mark commands.
1.1.180523	2018/05/23	Modified and the document format and organized documents in a standard way.
1.1.180601	2018/06/01	Added extension interface of image printing and descriptions of ICallback examples.
1.1.180612	2018/06/12	Bolded commands and part of the descriptions.
1.1.180629	2018/06/29	Modified errors and descriptions of ICallback interface.
1.1.180912	2018/09/12	Reorganized the documents.
1.1.190225	2019/02/25	Reorganized the FAQs.
1.1.191008	2019/10/08	Reorganized the FAQs.
1.1.191112	2019/11/12	Updated links and introductions of the resources.
1.1.191211	2019/12/11	Added introductions of the label functions.

1.1.200818	2020/08/18	Updated AIDL interface files.
1.1.200909	2020/09/09	Added descriptions for printing special symbols in the FAQs; Limited the label function to V2 pro models only.
1.1.210224	2021/02/24	Updated the version of remote library; Fixed writing mistakes.
1.1.210807	2021/08/07	Updated the version of remote dependency library; Added descriptions for importing custom fonts; Modified descriptions of the label function; Removed the method of H5 calling native interface (You can refer to demos on the official site).

Contents

Introduction

Some SUNMI devices have **built-in thermal printers with a buffer**, allowing Apps to print thermal receipts directly through SDKs. The SUNMI devices with a printer are:

Handheld POS — V1, V1s, V2 Pro, etc.

Handheld Payment POS — P1, P1-4g, etc.

Desktop POS — T1, T2, T1mini, T2mini, etc.

Desktop POS Scale — S2, etc.

There are two specs of SUNMI's built-in printers:

- Support 80mm paper width, equipped with paper cutter, compatible with the 58mm spec. T1 is equipped with a printer of this spec.
- Support 58mm paper width, no paper cutter. V1 is equipped with a printer of this spec.

You can use the following methods to call the built-in thermal printer:

- **Use the built-in print services interface.** If you are new to print-related Apps and are not familiar with Epson commands, this method works better for you. You can achieve the needed printing effect through multiple printing interfaces provided by SUNMI's printing services.

- **Use the built-in virtual Bluetooth devices.** If you have previous experience developing Bluetooth or USB printers, or have already implemented printing by Bluetooth printers, this method works better for you. You only need to slightly modify the code to achieve the needed printing effect.

1. • Use the built-in print services interface to call a printer.

1.1. Connect to the Built-in Print Service

1.1.1. By Remote Dependency

If you use Gradle configuration, remote dependency library is recommended to call the built-in printing service conveniently. Besides, our library has adapted to different models, and the remote dependency approach will automatically identify interfaces of different models. You do not need to configure different AIDL files for different models, and there will be corresponding prompts when a wrong interface is used.

Method of Integration

Add dependency and attribute configuration to the build.gradle file under the App's directory:

dependencies { implementation 'com.sunmi:printerlibrary:1.0.13' }

Initialization

Like binding a service component, we can call the binding method through the singleton pattern.

```
boolean result = InnerPrinterManager.getInstance().bindService(context, innerPrinterCallback);
    InnerPrinterCallback innerPrinterCallback = new InnerPrinterCallback(){

            @Override
            protected void onConnected(SunmiPrinterService service){
                    // Get the interface handle of the remote service after the binding service has been successfully connected.
                    // You can call supported printing methods through service.
            }

            @Override
            protected void onDisconnected() {
                    // The method will be called back if the service is disconnected abnormally. A reconnection strategy is recommended here.

            }
    }
```

result: Bind Successfully or Fail to Bind.

Simple Call of Printing Methods

```
try{
        service.printText("Contents to be printed \n", new InnerResultCallbcak() {
```

```
                @Override public void onRunResult(boolean isSuccess) throws RemoteException
                {
                //Return the execution result (not real printing): Succeeded or Failed.
                }


                @Override
        public void onReturnString(String result)
throws RemoteException
                {
                        //Some interfaces will return inquired data asynchronously.
                }

                @Override
        public void onRaiseException(int code, String msg) throws RemoteException
                {
                        // The returned exception status when the interface failed to execute.
        }

        @Override
        public void onPrintResult(int code, String msg)
Throws RemoteException
                {
                        // The returned result of real printing in transaction printing.
        }
        });
    } catch (RemoteException e) {
        // If some interfaces can only be used for specified models, an error in calling interface
will occur. For example, the interface of a cash drawer can only be used for a desktop device.
    }
```

End Release

After a normal call, you can call the following method to disconnect the service.

```
InnerPrinterManager.getInstance().unBindService(context, innerPrinterCallback);
```

Class Descriptions

InnerPrinterManager is the management class of the print library, used to connect and disconnect printing services.

InnerPrinterCallback is the callback interface to connect remote service, used to obtain the connection result.

SunmiPrinterService is the interface class of SUNMI's printing service and it defines all printing methods. This instance type can be obtained through connecting to the service, the same as the AIDL IWoyouService interface.

InnerPrinterException is the printing exception class. An error or exception will occur when calling a method because some models are not equipped with corresponding functions (For example, a handheld device is not equipped with a cash drawer, so printing exception will occur when calling the interface of a cash drawer, and the interface will be unavailable).

InnerResultCallback is the callback interface of printing method, used to obtain its execution result.

InnerLcdCallback is a callback interface of customer display method, used to obtain its execution result.

InnerTaxCallback is the callback interface of tax control method, which is used to obtain the result of sending the tax control.

1.1.2. By AIDL

Introduction to AIDL

AIDL is the abbreviation of Android Interface Definition Language, a description language of Android's internal process communication interface, through which we can define the Communication Interface between processes. SUNMI's AIDL provides encapsulated common printing commands for your quick access to SUNMI printers.

How to Use AIDL

You can use the following five steps to establish a connection:

1. Add the AIDL files attached to the resource files in the project, and package path and name cannot be changed (some types of printers also contain Java files);
2. Implement ServiceConnection in the code class that controls printing;
3. Call ApplicationContext.bindService(), and pass it in your ServiceConnection implementation. Note: bindservice is a non-blocking call, which means that the bind does not finish immediately after the call is completed. Therefore, you must refer to the callback of serviceConnected.
4. In your implementation of ServiceConnection.onServiceConnected(), you will receive an IBinder instance (called service). Call **IWoyouService.Stub.asInterface(service)** to convert the parameter to IWoyouService type.
5. You can now print by calling the various methods defined in the IWoyouService interface.

AIDL Resources Download

⚠Note: Please use AIDL resources according to different models, for some of the interfaces are different for each model. There are four resource packages currently:

Old (V1) for SUNMI's first V1 model

—Mainly includes:

IWoyouService.aidl	Printer's interface files
ICallback.aidl	Printer's callback interface file
TransBean.aidl	
TransBean.java	

Handheld (handheld devices except for V1) for SUNMI's handheld devices, such as V1s, V2, P2, etc.

—Mainly includes:

IWoyouService.aidl	Printer's interface files
ICallback.aidl	Printer's callback interface file
TransBean.aidl	
TransBean.java	
ITax.aidl	Callback interface files of tax control

Desktop for SUNMI's desktop printers, such as T1, T2, etc.

—Mainly includes:

IWoyouService.aidl	Printer's interface files
ICallback.aidl	Printer's callback interface file
ITax.aidl	Callback interface files of tax control

Desktop+LCD (Desktop + Customer display) for SUNMI's printers with a customer display, such as T1mini、T2mini, etc.

—Mainly includes:

IWoyouService.aidl	Printer's interface files
ICallback.aidl	Printer's callback interface file
ITax.aidl	Callback interface files of tax control
ILcdCallback.aidl	Callback interface files of customer display

General Resources

WoyouConsts.java Define the class constant exposed by the printer to set the configuration.

Example of Binding Services

Implement ServiceConnection

```
private ServiceConnection connService = new ServiceConnection() {
@Override
public void onServiceDisconnected(ComponentName name) {
    woyouService = null;
}
@Override
public void onServiceConnected(ComponentName name, IBinder service) {
     woyouService = IWoyouService.Stub.asInterface(service);
}
```

Bind Printing Services

```
private void Binding(){
    Intent intent=new Intent();
    intent.setPackage("woyou.aidlservice.jiuiv5");
    intent.setAction("woyou.aidlservice.jiuiv5.IWoyouService");
    bindService(intent, connService, Context.BIND_AUTO_CREATE);
}
```

1.2 Description of Interface Definition

After establishing the connection with the inbuilt print service using the methods stated above and obtaining IWoyouService or SunmiPrinterService object, you can call the following interface to print.

1.2.1. Printer Initialization and Setting

No.	Method
1	void **printerInit( )** Initialize the printer.
2	void **printerSelfChecking(ICallback** callback**)** Run the printer self-test.

1. Initialize the printer.

Funciton: void **printerInit( )**

Note: Reset the printer's logic programs (Typesetting, Bolding, etc.), but do not clear the cached data. Therefore, unfinished print jobs will be continued after resetting.

2. Run the printer self-test.

Function： void **printerSelfChecking (ICallback** callback**)**

Parameter： callback → Result Callback

Example：

```
woyouService.printerSelfChecking(callback);
```

1.2.2. Get Device and Printer Information

No.	Method
1	String **getPrinterSerialNo( )** Get the printer's SN.
2	String **getPrinterModal( )** Get the printer's interface type.
3	String **getPrinterVersion( )** Get the firmware version of the printer.
4	Build**.MODEL**（Constant） Device name.
5	int **updatePrinterState()** Get the latest status of the printer.
6	String **getServiceVersion()** Get the printer service's version number.
7	int **getPrintedLength(ICallback** callback**)** Get the printhead's print length.
8	int **getPrinterPaper()** Get the current paper size of the printer.

1. Get the latest status of the printer.

Function: String **updatePrinterState ()**

Return Values：

1 → The printer works normally.
2 → Preparing the printer.
3 → Communication error.
4 → Out of paper.
5 → Overheated.
6 → The printer's cover is open.

7 → Cutter error.

8 → Cutter restored.

9 → No black mark has been detected.

505 → No printer has been detected.

507 → Failed to update the printer's firmware.

Description: These return values can be applied to all SUNMI devices, but some states cannot due to hardware configuration (For example, handheld models do not support cover open detection).

Note: This interface is not available for V1 devices for now. Aside from actively getting the status, you can also obtain them asynchronously by registering BroadcastReceiver. You can refer to Appendix A.

2. Get the printed length.

Function: int **getPrintedLength(ICallback** callback**)**

Description: Currently, the print length of the printer since power on can be obtained. For a desktop device and a handheld device, the ways to get the print results' return values are slightly different due to hardware difference. The handheld device gets its print length through ICallback callback interface while the desktop device gets its length directly through the return values.

3. Get the printer's current paper size.

Function: int **getPrinterPaper()**

Description: By default, handheld printers use 58 mm paper, and desktop printers use 80 mm paper, but you can add a baffle and adjust some configurations to enable a desktop printer to use 58mm paper. This interface will return the paper size currently used by the printer.

Note: Currently, desktop devices that support this interface are: T1 (v2.4.0 or above), T2 (v1.0.5 or above), and S2 (v1.0.5 or above); Other models with a version 4.1.2 or above can also use this interface to get the printer's paper size.

1.2.3. ESC/POS Commands

No.	Methods
1	void **sendRAWData (**byte[] data, **ICallback** callback**)** Print ESC/POS commands.

1. Print ESC/POS commands.

Function: void **sendRAWData(**byte[] data, **ICallback** callback**)**

Parameter:

data → ESC/POS commands

callback → Result Callback

Note: For related commands, please refer to *ESC/POS Command Set.*

```
woyouService.sendRAWData(new byte[]{0x1B,0x45,0x01}, callback); // 1B,45, 01 are commands of bolding fonts.
```

1.2.4 Description of the Interface for Print Style Setting

No.	Method
1	void **setPrinterStyle(**int key, int value**)** Set the print style.

1. **Set the print style.**

Function: void **setPrinterStyle(**int key, int value**)**

Parameter: Refer to the print style of WoyouConsts.java to set class constants.

key → Set attributes which are usually divided into **ENABLE_XXX** and **SET_XXX** according to the definition in constant interface.

value → Correspond to status or size in the attribute setting.

You need to choose between ENABLE and DISABLE when setting the ENABLE_XXX attribute.

You need a specific size when setting the SET_XXX attribute.

Note: This interface is only available for print services with a version 4.2.22 or above.

Example:

```
woyouService.setPrinterStyle( WoyouConsts.ENABLE_ILALIC,WoyouConsts.ENABLE); // Italicize the text to be printed subsequently.
woyouService.setPrinterStyle(WoyouConsts.SET_LINE_SPACING,123); // Change the leading of the text to be printed subsequently to 123 pixels.
```

1.2.5. Change the Print Mode

No.	Method / Command
1	int **getPrinterMode**() Get the print mode.
2	int **getPrinterBBMDistance**() Get the paper feed distance in current black mark mode.
3	For related mode settings, please set them in “Settings” → “Print Settings”.

1. **Get the print mode.**

Function: int **getPrinterMode**()

Return Values:

0 → Normal mode.

1 → Black mark mode.

2 → Label mode.

Note: Black mark mode is currently available for SUNMI's desktop devices T1 and T2; Label mode is currently available for SUNMI's handheld devices V2 and V2 Pro.

2. Get the Printer's Paper Feed Distance in Black Mark Mode.

Function: int **getPrinterBBMDistance**()

Return Values: Paper feed distance (pixel line)

Note: Only available for T1 and T2.

1.2.6. Print Texts

No.	Method / Command
1	void **setAlignment**(int alignment, **ICallback** callback) Set alignment mode.
2	void **setFontName**(String typeface, **ICallback** callback) Set custom fonts.
3	void **setFontSize**(float fontsize, **ICallback** callback) Set font size.
4	ESC/POS commands: Bold fonts {0x1B, 0x45, 0x1}; Remove bold fonts {0x1B, 0x45, 0x0}. Set and remove bold fonts.
5	void **printText**(String text, **ICallback** callback) Print texts.
6	void **printTextWithFont**(String text, String typeface, float fontSize, **ICallback** callback) Print texts in specified font and size.
7	void **printOriginalText** (String text, **ICallback** callback) Print vector fonts.

1. Set Alignment Mode.

Function: void **setAlignment** (int alignment, **ICallback** callback)

Parameter:

alignment → Alignment: 0→ left-aligned, 1→ centered, 2→right-aligned.

callback → Result Callback

Note: As a global method, it will affect the subsequent printing. The related settings can be disabled after initializing the printer.

Example:

```
woyouService.setAlignment(1, callback);
```

2. Set custom fonts.

Function: void **setFontName (**String typeface, **ICallback** callback**)**

Parameter:

typeface → Custom font to be used. Currently, only vector fonts are supported and they need to be pre-set in the App.

assets → Contents

callback → Result Callback

Description: This interface can extend the printer's default fonts and allow you to use custom fonts. But the leading and line width need to be adjusted due to different inner widths of each font.

Note: This interface is only available for print services with a version 4.14.0 or above.

3. Set font size.

Function: void **setFontSize (**float fontSize, **ICallback** callback**)**

Parameter:

fontSize → Font size.

callback → Result Callback

Note: As a global method, it will affect the subsequent printing. The related settings can be disabled after initializing the printer. Setting font size is not included in the standard international instruction for printing, and it will affect character width and the number of characters in a line. As a result of this, the typesetting formed by equal-width fonts may be changed.

Example:

```
woyouService.setFontSize(36, callback);
```

4. Set and remove bold fonts.

Command: Bold fonts {0x1B, 0x45, 0x1}, Remove bold fonts {0x1B, 0x45, 0x0}

Note: Please refer to **1.2.3. ESC/POS Commands**.

Example:

```
woyouService.sendRAWData(new byte[]{0x1B,0x45,0x0}, callback); // Cancel the command of bolding fonts.
```

5. Print texts.

Function: void **printText(**String text, in **ICallback** callback**)**

Parameter:

text → The text to be printed. The part of the text **exceeding one line will be automatically changed to the next line**, and code for **end-of-line "\n"** should be added to the end of a line for immediate print if the text is less or more than the line, otherwise it will be cached in the buffer.

callback → Result Callback

Note: If you want to change the print style (such as alignment, font size, bold, etc.) of the text, please set before calling printText method.

Example:

```
woyouService.setAlignment(1, callback);
woyouService.setFontSize(36, callback);
woyouService.printText("SUNMI Technology\n", callback);
```

6. Print texts in specified font and size.

Function: void **printTextWithFont(**String text, String typeface, float fontSize, **ICallback** callback**)**

Parameter:

text → The text to be printed. The part of the text **exceeding one line will be automatically changed to the next line**, and code for **end-of-line "\n"** should be added to the end of a line for immediate print if the text is less or more than the line, otherwise it will be cached in the buffer.

typeface → Custom font to be used. Currently, only vector fonts are supported and they need to be pre-set in the App's assets folder.

fontSize → Font size. Only effective for this method.

callback → Result Callback

Note: This interface is only available for print services with a version 4.14.0 or above.

Example:

```
woyouService.printTextWithFont("SUNMI\n","",36,callback);
```

7. Print vector fonts.

Function: void **printOriginalText(**String text, **ICallback** callback**)**

Parameter:

text → The text to be printed. The exceeding part **will be automatically changed to the next line**, and code for **end-of-line "\n"** should be added to the end of a line for immediate print if the text is less or more than the line, otherwise it will be cached in the buffer.

callback → Result Callback

Return Values:

Note: Output characters are **in the same width of vector fonts**, i.e., they **are not of equal width.**

Example:

```
woyouService.printOriginalText ("κρχκμνκλρκνκνμρτυφ\n", callback);
```

1.2.7. Print Tables

No.	Method / Command
1	void **printColumnsText**(String[] colsTextArr, int[] colsWidthArr, int[] colsAlign, **ICallback** callback) Print one raw in the table (Arabic letters are not supported).
2	void **printColumnsString**(String[] colsTextArr, int[] colsWidthArr, int[] colsAlign, **ICallback** callback) Print one raw in the table. You can specify the column width and alignment mode.

1. Print one raw in the table (Arabic letters are not supported).

Function: void **printColumnsText**(String[] colsTextArr, int[] colsWidthArr, int[] colsAlign, **ICallback** callback)

Parameter:

colsTextArr → Array of text strings in each column.

colsWidthArr → Array of widths of each column, calculated in English characters. Each One Chinese character takes the space of two English characters, and the width should be greater than 0.

colsAlign → Alignment of each column: 0→ left-aligned, 1→ centered, 2→right-aligned.

callback → Result Callback

Note: The array length of the three parameters should be the same. If the colsText[i] is wider than the colsWidth[i], the exceeding part will be changed to the next line. Arabic letters are not supported.

Example:

```
woyouService.printColumnsText(new String[]{"SUNMI","SUNMI","SUNMI"},
new int[]{4,4,8}, new int[]{1,1,1},callback);
```

2. Print one raw in the table. You can specify the column width and alignment mode.

Function: void **printColumnsString**(String[] colsTextArr, int[] colsWidthArr, int[] colsAlign, **ICallback** callback)

Parameter:

colsTextArr → Array of text strings in each column.

colsWidthArr → The percentage of each column is determined by its width.

colsAlign → Alignment of each column: 0→ left-aligned, 1→ centered, 2→right-aligned.

callback → Result Callback

Note: The array length of the three parameters should be the same. If the colsText[i] is wider than the colsWidth[i], the exceeding part will be changed to the next line.

Example:

```
woyouService.printColumnsString(new String[]{"SUNMI","SUNMI","SUNMI"},
new int[]{1,1,2}, new int[]{1,1,1},callback);
```

1.2.8. Print Images.

No.	Method
1	void **printBitmap**(Bitmap bitmap, **ICallback** callback) Print images.
2	void **printBitmapCustom**(Bitmap bitmap, int type, **ICallback** callback) Print images (2).

1. **Print images.**

Function: void **printBitmap** (Bitmap bitmap, **ICallback** callback)

Parameter:

bitmap → Image Bitmap object.

callback → Result Callback

Note: The image resolution should be less than 2,500,000, and the width should be set per paper size (384 pixels for 58; 576 pixels for 80). It cannot be shown if it exceeds the paper width.

Example:

```
woyouService.printBitmap(bitmap,callback);
```

2. **Print images (2).**

Function: void **printBitmapCustom** (Bitmap bitmap,int type **ICallback** callback)

Parameter:

bitmap → Image bitmap object (The maximum width is 384 pixels. Images exceeding 1M cannot be printed).

type→ There are two printing methods currently:

0 → The same method of **printBitmap**().

1 → Black-and-white images with a threshold of 200.

2 → Grayscale images.

callback → Result Callback

Note: The image resolution should be less than 2,500,000, and the width should be set per paper size (384 pixels for 58; 576 pixels for 80). It cannot be shown if it exceeds the paper width.

Supported models: P1-v3.2.0 or above, P14g-v1.2.0 or above, V1s-v3.2.0 or above, V2-v1.0.0 or above, T1-v2.4.0 or above, T2、S2-v1.0.5 or above, T1mini-v2.4.1 or above, T2mini-v1.0.0 or above.

Example:

```
woyouService.printBitmapCustom(bitmap,callback);
```

1.2.9. Print 1D and 2D barcodes

No.	Method
1	void **printBarCode(**String data, int symbology, int height, int width, int textPosition, **ICallback** callback**)** Print 1D barcodes.
2	void **printQRCode(**String data, int modulesize, int errorlevel, **ICallback** callback) Print QR codes.
3	void **print2DCode(**String data, int symbology, int modulesize, int errorlevel, **ICallback** callback) Print 2D barcodes.

1. Print 1D barcodes.

Function: void **printBarCode(**String data, int symbology, int height, int width, int textPosition, **ICallback** callback**)**

Parameter:

data → 1D barcodes

symbology → Type of barcodes (0 – 8):

0 → UPC-A
1 → UPC-E
2 → JAN13(EAN13)
3 → JAN8(EAN8)
4 → CODE39
5 → ITF
6 → CODABAR
7 → CODE93
8 → CODE128

height → Value of bar height: 1 – 255; Default: 162

width → Value of bar width: 2 – 6; Default: 2

textPosition → Text Position (0 – 3):

0 → No text printed
1 → Text above the barcode
2 → Text below the barcode
3 → Text above and below the barcode

callback → Result Callback

Note: Differences of Different Types of Barcodes:

Code	Description
code39	A maximum of 13 numbers can be printed.
code93	A maximum of 17 numbers can be printed.
ean8	Numbers are required and the valid length is 8 (The last digit is the check digit).
ean13	Numbers are required and the valid length is 13 (The last digit is the check digit).
ITF	Even numbers are required and the valid length is within 14.
Codebar	Numbers within 0-9 and 6 special characters are required. A maximum of 18 digits can be printed.
UPC-E	8-digit numbers are required (The last digit is the check digit).
UPC-A	12-digit numbers are required (The last digit is the check digit).
code128	Code128 can be divided into three types: A: Uppercase letters, numbers, punctuation, etc.; B: Uppercase and lowercase letters, numbers; C: Numbers only. It must have an even number of digits; If it has an odd one, the last digit will be omitted. By default, the interface uses B type. "{A" or "{C" should be added before the content if you need to use A or C type. For example: "{A2344A", "{C123123", "{A1A{B13B{C12".

Example:

```
woyouService.printBarCode("1234567890", 8, 162, 2, 2, callback);
```

2. **Print QR codes.**

Function: void **printQRCode (**String data, int modulesize, int errorlevel, **ICallback** callback**)**

Parameter:

data → QR codes

modulesize → the size of QR codes. Unit: dot; Value: 4-16.

errorlevel → Level of correctness(0 - 3):

0 → Level of correctness L (7%)

1 → Level of correctness M (15%)

2 → Level of correctness Q (25%)

3 → Level of correctness H (30%)

callback → Result Callback

Note: After calling this method, the content will be printed directly under normal print status. Every QR code is 4 pixels (If it is smaller than this size, the code parsing might fail), and the maximum mode supported is version 19 (93*93).

Example:

```
woyouService.printQrCode("SUNMI Technology", 4, 3, callback);
```

3. **Print 2D barcodes.**

Function: void **print2DCode(**String data, int symbology, int modulesize, int errorlevel, **ICallback** callback)

Parameter:

data → the 2D barcodes to be printed

symbology → Type of the 2D barcodes

1 Qr (same as the **printQRCode** interface)

2 PDF417

3 DataMatrix

modulesize → The valid size of 2D barcode, and it varies in accordance with different types of 2D barcodes.

QR codes	4～16（same as the **printQRCode** interface）
PDF417	1～4
DataMatrix	4～16

errorlevel → 2D barcode's level of correctness. It varies in accordance with different types of 2D barcodes.

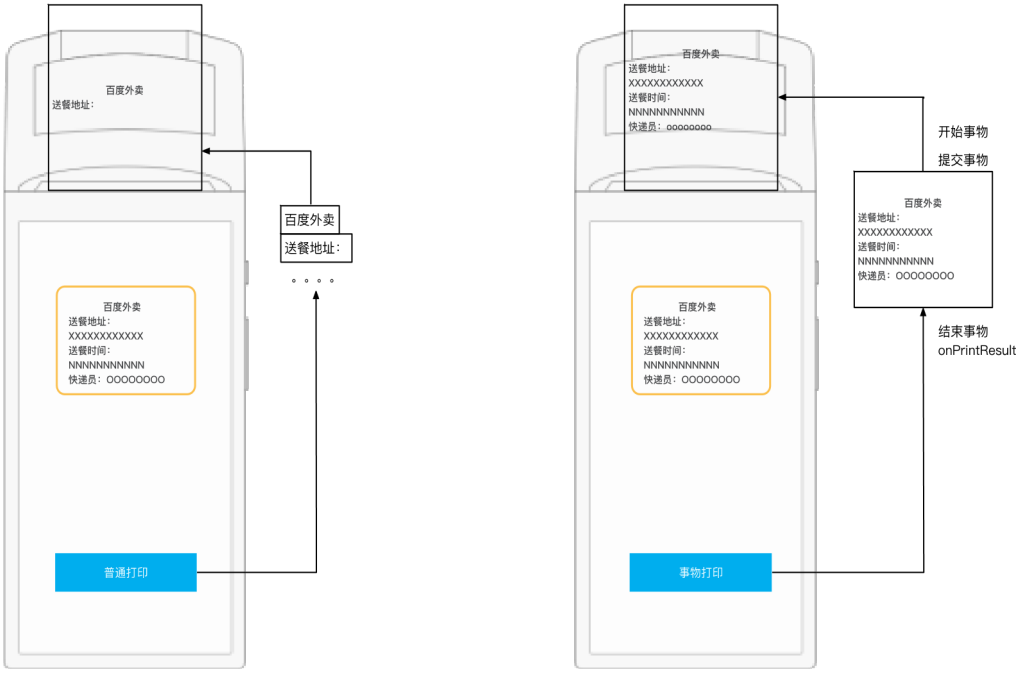
QR codes	0～3（same as the **printQRCode** interface）
PDF417	0～8
DataMatrix	Use ECC200 automatic correction by default. Setting is not supported.

callback → Result Callback

Note: After calling this method, the content will be printed directly under normal print status. This interface is available for version 4.1.2 and above.

1.2.10. Transaction Printing

Transaction printing mode is suitable for the need to control the print content and get feedback on the results of the print (whether the receipt is printed or not). Like establishing a buffer for the transaction queue, when you use this mode, a transaction queue will be opened and you can add printing methods to it. Instead of printing the content instantly, the printer will execute the printing tasks in turn after the transaction is submitted, and feedback on print results will be obtained after printing.



Notes on Transaction Printing:

1. When printing transactions in the buffer, the printing results will be returned after printing successfully. However, **all print jobs submitted this time will lose** if **errors including out of paper, overheated occurred**, and **error feedback will be given**. That means if an error occurs before or during a print job, the job will not be printed.
2. If you alternately using transaction printing in the buffer and command printing, the print jobs of command printing will not lose if an error occurs.
3. In transaction printing mode, you will also use other interfaces in 1.2.x. to output the print content. However, the content **will not** be printed instantly but saved in the buffer, and will only be printed after **exitPrinterBuffer() or commitPrinterBuffer()** methods being called.
4. Transaction printing results will be called back in onPrintResult (int code, String msg) of ICallback method (It may take some time to wait for **the paper to be printed out**. It is not recommended to use transaction printing **frequently for a single line**, for **print speed will be reduced**. Instead, transaction printing is suitable for printing **a whole receipt**). The corresponding return codes are:

   a) 0 → Printed successfully. The msg is “Transaction print successful!”;

   b) 1 → Failed to print. The msg is “Transaction print failed!”.
5. The following is the complete pseudocode of transaction printing:

enterPrinterBuffer(true) —— You are using transaction printing mode. The subsequent commands will not be outputted instantly.
printText(/*something*/)
printBitmap(/*bitmap resource*/)
…… Other print-related methods——Print something
commitPrinterBuffer()/commitPrinterBufferWithCallback(callback)——The printer will start printing when a transaction is submitted. The result (succeeded or failed) will be returned in callback.
……Wait for the return of last transaction.
printText(/*something*/)
printBitmap(/*bitmap resource*/)
……Other print-related methods——You can choose to wait for the result of the last transaction printing or to continue to print.
commitPrinterBuffer()/commitPrinterBufferWithCallback(callback)——Continue to submit the next transaction, and the printer will continue to print.
exitPrinterBuffer(true)/exitPrinterBufferWithCallback(true, callback)——Call it when exiting transaction printing. Printing will be continued if new data has been entered after the last submission, otherwise it does not print.

6. Description of Detailed Methods

No.	Method
1	void **commitPrint**(TransBean[] tranBean, **ICallback** callback) The interface specially for transaction printing of lib package.
2	void **enterPrinterBuffer**(boolean clean) Use transaction printing mode.
3	void **exitPrinterBuffer**(boolean commit) Exit transaction printing mode.
4	void **exitPrinterBufferWithCallback**(boolean commit, **ICallback** callback) Exit transaction printing mode and return the result from callback.
5	void **commitPrinterBuffer**() Submit transaction printing.
6	void **commitPrinterBufferWithCallback**(**ICallback** callback) Submit transaction printing and return the result from callback.

1. The interface specially for transaction printing of lib package.

Function: void **commitPrint** (TranBean[] tranBean, **ICallback** callback)

Parameter:

tranBean → Print job list.

callback → Result Callback

Example:

```
woyouService.commitPrint(tranBean, callback);
```

2. **Use transaction printing mode.**

Function: void **enterPrinterBuffer(**Boolean clear**)**

Parameters:

clear → Whether to clear the cached data in the buffer:

true → Clear the unsubmitted data in the last transaction printing;

false →Not clear the unsubmitted data in the last transaction printing, and it will be included in next submission.

Note:

1. Use the transaction printing mode, where data will not be printed instantly until submitting a transaction or exiting the submission of a transaction.
2. Versions supported: all SUNMI devices except V1.

Example:

```
woyouService.enterPrinterBuffer(false);
```

3. **Exit transaction printing mode.**

Function: void **exitPrinterBuffer(**Boolean commit**)**

Parameter:

commit → Whether to print the cached data in the buffer:

true → Print the data in the transaction queue;

false →Not print the data in the transaction queue, and it will be saved until next submission.

Note: Versions supported: all SUNMI devices except V1.

Example:

```
woyouService.exitPrinterBuffer(true);
```

4. **Exit transaction printing mode and return the result from callback.**

Function: void **exitPrinterBuffer(**Boolean commit, **ICallback** callback**)**

Parameter:

commit → Whether to print the cached data in the buffer:

true → Print the data in the transaction queue;

false →Not print the data in the transaction queue, and it will be saved until next submission.

callback → Result Callback

Note: Versions supported: all SUNMI devices except V1.

Example:

```
woyouService.exitPrinterBuffer(true);
```

5. **Submit transaction printing.**

Function: void **commitPrinterBuffer()**

Note:

1. submit and print a transaction queue, and continue to use transaction printing mode.
2. Versions supported: all SUNMI devices except V1.

Example:

```
woyouService.commitPrinterBuffer();
```

6. **Submit transaction printing and return the result from callback.**

Function: void **commitPrinterBufferWithCallbacka(ICallback** callback**)**

Parameter:

callback → Result Callback

Note: Versions supported: all SUNMI devices except V1.

Example:

```
woyouService.commitPrinterBufferWithCallback(callback);
```

1.2.11. Paper Feeding

No.	Method
1	void **lineWrap**(int n, **ICallback** callback) Feed paper n lines.

1. **Feed paper n lines.**

Function: void **lineWrap**(int n, **ICallback** callback)

Parameter:

n → The number of lines

callback → Result Callback

Note: Forced line break. After ending the previous printing, feed paper **n** lines.

Example:

```
woyouService.printBitmap(3, callback);
```

1.2.12. Cut Paper

No.	Method
1	void **cutPaper(ICallback** callback**)** Cut paper.
2	int **getCutPagerTimes()** Get the number of times a cutter has been used.

1. **Cut paper.**

Function: void **cutPaper (ICallback** callback**)**

Parameter: callback → Result Callback

Note: The gap between the printhead and the paper cutter will be automatically filled up by calling the interface;

It is only available for the desktop devices with a cutter.

Example:

```
woyouService.cutPager(callback);
```

2. **Get the number of times a cutter has been used.**

Function: int **getCutPaperTimes ()**

Return Values: The number of times a cutter has been used

Note: It is only available for the desktop devices with a cutter.

1.2.13. Cash Drawer

No.	Method
1	void **openDrawer(ICallback** callback**)** Open the cash drawer.
2	int **getOpenDrawerTimes()** Get the number of times a cash drawer has been opened.
3	int **getDrawerStatus()** Get the current status of a cash drawer.

1. **Open the cash drawer.**

Function: void **openDrawer (ICallback** callback**)**

Parameter:

callback → Return Callback

Note: It is only available for the desktop devices with a cash drawer.

Example:

```
woyouService.openDrawer(callback);
```

2. **Get the number of times a cash drawer has been opened.**

   Function: int **getCutPaperTimes ()**

   Return Values: The number of times a cash drawer has been opened.

   Note: It is only available for the desktop devices with a cash drawer.

3. **Get the current status of a cash drawer.**

   Function: int **getDrawerStatus()**

   Description: You can get the status of the cash drawer's switch through this interface on some models that can connect the cash drawer.

   Note: It is only available for S2, T2, and T2mini devices with a version 4.0.0 or above.

1.2.14. Get the Global Attribute Setting.

You can override the style set previously by configuring some print styles. The following interfaces can be used to obtain the enabled configuration status.

No.	Method
1	int **getForcedDouble()** Get the status of global fonts in double height and double width.
2	boolean **isForcedAntiWhite()** Get the white to black style of global fonts.
3	boolean **isForcedBold()** Get the bold style of global fonts.
4	boolean **isForcedUnderline()** Get the underline style of global fonts.
5	int **getForcedRowHeight**() Get the set value of global line height.
6	int **getFontName**() Get the font currently being used.
7	int **getPrinterDensity**() Get the print density.

Note: Currently, all interfaces, except the interface to get the print density, are supported only in the handheld devices V1, V1s, P1 with a version 3.2.0 or above and P14g with a version 1.2.0 or above.

1.2.15. Description of the Customer Display Interface.

Devices of mini-series have the function of customer display. You can use according to the following methods:

No.	Method
1	void **sendLCDCommand**(int flag) Send control command.
2	void **sendLCDString**(String string, ILcdCallback callback) Send a single-line text.
3	void **sendLCDDoubleString**(String topText, String bottomText, ILcdCallback callback) Send a double-line text.
4	void **sendLCDMultiString**(String[] text, int[] align, ILcdCallback callback) Send a multi-line text. The size of each line will be adjusted according to the content weight.
5	void **sendLCDFillString**(String string, int size, boolean fill, ILcdCallback callback) Send a single-line text. The filling method and font size can be set.
6	void **sendLCDBitmap**(Bitmap bitmap, ILcdCallback callback) Send and show bitmap images.

1. **Send control command.**

   Function: void **sendLCDCommand**(int flag)

   Parameter: flag → 1 Initialization; 2 Wake LCD; 3 Hibernate LCD; 4 Clear the screen.

   Note: It is only available for desktop devices of mini-series with a customer display.

2. **Send a Single-Line Text.**

   Function: void **sendLCDString**(String string, ILcdCallback callback)

   Parameter: string → Text being shown.

   callback → Asynchronous callback of results.

   Note: It is only available for desktop devices of mini-series with a customer display. Text cannot be displayed if it is too long.

3. **Send a Double-Line Text.**

   Function: void **sendLCDDoubleString**(String topText， String bottomText, ILcdCallback callback)

   Parameter: topText → Show the top half text.

   bottomText → Show the bottom half text.

   callback → Asynchronous callback of results.

   Note: It is only available for desktop devices of mini-series with a customer display. Text cannot be displayed if it is too long.

4. **Send a Multi-Line Text.**

Function: void **sendLCDMultiString**(String[] text, int[] align, ILcdCallback callback)

Parameter: text → Show the array of each line, and determine the number of lines based on it. No content will be shown if the line is blank.

align → The weight ratio of the area occupied by each line, the array of which must be the consistent as the text array.

callback → Asynchronous callback of results.

Note: It is only available for desktop devices of mini-series with a customer display. Text cannot be displayed if it is too long.

5. **Send a single-line text in a customized size.**

Function: void **sendLCDFillString**(String string, int size, boolean fill, ILcdCallback callback)

Parameter: string → Text to be shown.

size → The font size of the text. Default: centered.

fill → Whether to stretch the text to fill the display area

callback → Asynchronous callback of results.

Note: It is only available for desktop devices of mini-series with a customer display. Text cannot be displayed if it is too long.

6. **Send and show bitmap images.**

Function: void **sendLCDBitmap**(Bitmap bitmap, ILcdCallback callback)

Parameter: bitmap → Images to be shown.

callback → Asynchronous callback of results.

Note: It is only available for desktop devices of mini-series with a customer display. The maximum pixels that can be displayed is 128*40.

1.2.16. Description of Label Printing

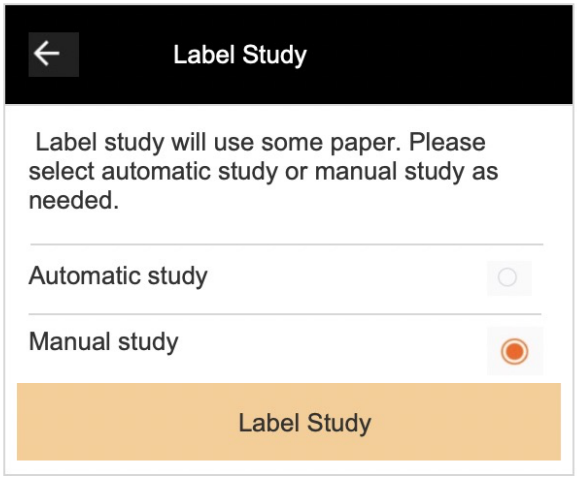
SUNMI mobile V series devices currently support label printing (such as V2pro, V2s). You can refer to the following descriptions:

Precondition:

1. Set mode——The printers of SUNMI devices are used to print thermal receipts by default. If you want to use the label function, please enter Print Setting → Built-in Setting (or SunmiPrinter) → Select Printer Mode → Label Mode. Your selection will be recorded and the device will be used as a label printer. Please see the picture below (Update your system to the latest version if you cannot find it.):

You can check the current print mode through the interface getPrinterMode().

2. Label study——After selecting the Label Mode, you can see the new Label Study item. If you use it for the first time or change a different type of paper, please click Label Study. Please see the picture below:



This will initiate label paper study process. Please ensure there are at least 3 pieces of labels. A pop-up window will indicate the result of the study when the process ends.

You are suggested to use labels above **50mm** wide and **30mm** high and the gap between labels should be at least **2mm** to reach the best positioning effect.

Relevant interfaces of the label function:

No.	Method
1	void **labelLocate**() Position the next label.
2	void **labelOutput**() Output a label to the cutter position.

Print a label:

Label-related interfaces only provide positioning of a label, and you should design the label content according to your needs, just like the thermal printing. One thing to note is to control the height of the content within the height of the label, otherwise the content may exceed to the next label or lead to an inaccurate label positioning.

If you use a 30mm-high label paper, the content with a height of 240 (30mm x8) pixel lines can be printed. The default print leading is 32, so the 8-line content or a 384x240 high image can be printed.

For example—How to design a simple label content:

```
labelLocate()
// Positioning the label every time before sending the print content

Design the content.
setPrinterStyle(WoyouConsts.ENABLE_BOLD, WoyouConsts.ENABLE)
//Bold fonts.
linewrap(1, null)
// Leave a blank line at the head.
setAlignment(0, null)
// Set the content to be left-aligned.
printText("Item：        Soy Milk\n", null)
// Print the content. \n is needed.
printText("Expiration time:      12-13 14:00 \n", null)
// Print the content. \n is needed.
printBarCode("{C1234567890123456", 8, 90, 2, 2, null)
```

// Print the barcode. Here the barcode printed is a code128c type barcode with a height of 90, which occupies 154 pixel lines (A blank line above and below the barcode will be left).

linewrap(1, null)

// Leave a blank line or not according to the case.

The content basically takes all the area of a label will be outputted, please see the image below:

After the content is printed, you can choose whether to continue with labelLocate and print the content repeatedly.

If printing is not required,labelOutput() will be implemented, which will allow the label to be output to the cutter for easy peeling.You can add other APIs to design needed label content.

Print more than one label:

If you only print one label, you can perform:

labelLocate -> Print label（-> labelOutput）

If you print more than one label, instead of implementing labelOutput to output the labels after sending the label content, you only need to:

labelLocate->Print label 1->labelLocate->Print label 2->labelLocate->Print label 3……

->labelLocate->Print label n->labelOutput

Note: Currently labels can only be used through interface, and calling command sets is not supported.

1.3. Description of Interface Return

Sine the AIDL callback interface is often instantiated incorrectly by developers, the InnerResultCallback is used to replace the original callback interface for remote import library.

1.3.1. Description of InnerResultCallback Interface Method.

No.	Method
1	void **onRunResult**(boolean isSuccess) Results of conducting commands.

2	void **onReturnString**(String result) Returned result string of conducting commands.
3	void **onRaiseException**(int code, String msg) Returned error information.
4	void **onPrinterResult**(int code, String msg) Feedback on transaction printing.

1. **Results of conducting commands.**

   Function: void **onRunResult** (boolean isSuccess)

   Parameter:

   isSuccess → Implementation results:

   true → Implemented successfully.

   false → Failed to implement.

   Note: This interface returned result indicates the result of implementing a command, instead of the result of printing paper.

2. **Returned result string of conducting commands.**

   Function: void **onReturnString** (String result)

   Parameter:

   result → Implementation results

3. **Returned error information.**

   Function: void **onRaiseException** (int code, String msg)

   Parameter:

   code → Error code.

   msg → Error description.

   Note: Error Codes.

4. **Feedback on transaction printing.**

   Function: void **onPrintResult** (int code, String msg)

   Parameter:

   code → Status code:

   0 → Succeeded.

   1 → Failed.

   msg → Succeeded: null; Failed: error descriptions.

   Note: This interface returned result indicates the results of implementing command and printing (it may **take some time for ejecting paper**).

1.3.2. Examples of Callback Object.

```
new InnerResultCallback{
        @Override
        public void onRunResult(boolean b) throws RemoteException {
            // Results of conducting commands.
        }
        @Override
        public void onReturnString(String s) throws RemoteException {
            // Returned result string of conducting commands.
        }
        @Override
        public void onRaiseException(int i, String s) throws RemoteException {
            // Returned error information.
        }
        @Override
        public void onPrintResult(int i, String s) throws RemoteException {
             // Feedback on transaction printing.
        }
    }
```

1.3.3. Error Codes

code	msg
-1	"command is not support,index #"; //# Indicates that the # th byte went wrong.
-2	"# encoding is not support"; //# Indicates that the # th byte went wrong.
-3	"oops,add task failed (the buffer of the task queue is 10M),please try later";
-4	"create command failed";
-5	"Illegal parameter";
-6	"param found null pointer"

2. Call the Printer via the Built-in Virtual Bluetooth.

2.1. Introduction to Virtual Bluthtooth

You can find a Bluetooth device “InnerPrinter” that has already been paired and always exists in the list. It is a virtual printer invented by the OS and supports SUNMI ESC/POS commands.

Some special commands are SUNMI customized. For example:

Function	Command
Open the cash drawer.	byte［5］： 0x10 0x14 0x00 0x00 0x00
Cutter – Cut the receipt paper.	byte［4］： 0x1d 0x56 0x42 0x00
Cutter – cut the receipt paper (Leave a little on the left side).	byte［4］： 0x1d 0x56 0x41 0x00

2.2. Use the Virtual Bluetooth.

1. Connect to the Bluetooth device.
2. Combine and transcode commands and texts into Bytes.
3. Send the Bytes to InnerPrinter.
4. Drive the printer with the underlying print service to complete printing.

Note: **BluetoothUtil** is a Bluetooth tool class, used to connect the virtual Bluetooth device **InnerPrinter**.

2.2.1. The Tool Class BluetoothUtil, a Standard Tool for Bluetooth Connection.

```
public class BluetoothUtil {

    private static final UUID PRINTER_UUID =
                UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");

    private static final String Innerprinter_Address = "00:11:22:33:44:55";

    public static BluetoothAdapter getBTAdapter() {
        return BluetoothAdapter.getDefaultAdapter();
    }

    public static BluetoothDevice getDevice(BluetoothAdapter bluetoothAdapter) {
        BluetoothDevice innerprinter_device = null;
        Set<BluetoothDevice> devices = bluetoothAdapter.getBondedDevices();
        for (BluetoothDevice device : devices) {
            if (device.getAddress().equals(Innerprinter_Address)) {
                innerprinter_device = device;
                break;
            }
        }
        return innerprinter_device;
    }

    public static BluetoothSocket getSocket(BluetoothDevice device) throws IOException
{
        BluetoothSocket socket =
device.createRfcommSocketToServiceRecord(PRINTER_UUID);
        socket.connect();
        return socket;
    }

    public static void sendData(byte[] bytes, BluetoothSocket socket) throws IOException
{
        OutputStream out = socket.getOutputStream();
        out.write(bytes, 0, bytes.length);
        out.close();
    }
}
```

2.2.2. Examples of Connecting Print Service via Bluetooth.

1: Get BluetoothAdapter

```
BluetoothAdapter btAdapter = BluetoothUtil.getBTAdapter();

if (btAdapter == null) {
   Toast.makeText(getBaseContext(),"Please Open Bluetooth!",
Toast.LENGTH_LONG).show();
   return;
}
```

2: Get Sunmi's InnerPrinter BluetoothDevice

```
BluetoothDevice device = BluetoothUtil.getDevice(btAdapter);
if (device == null) {
     Toast.makeText(getBaseContext(),"Please Make Sure Bluetooth have
     InnterPrinter!", Toast.LENGTH_LONG).show();
      return;
}
```

3: Generate a order data , user add data here

```
byte[] data = null;
```

4: Using InnerPrinter print data

```
BluetoothSocket socket = null; socket = BluetoothUtil.getSocket(device);
BluetoothUtil.sendData(data, socket);
```

2.2.3. Notes

A Bluetooth device can only be used by adding Bluetooth Permission Declaration to the project in the App:

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

Appendix A Print Service Broadcast

Through broadcast: users need to create a broadcast receiver to monitor broadcasts.

Function	Action
Preparing the printer.	"woyou.aidlservice.jiuv5.INIT_ACTION"
Updating the printer.	"woyou.aidlservice.jiuv5.FIRMWARE_UPDATING_ACITON"
Ready to print.	"woyou.aidlservice.jiuv5.NORMAL_ACTION"
Print errors.	"woyou.aidlservice.jiuv5.ERROR_ACTION"
Out of paper.	"woyou.aidlservice.jiuv5.OUT_OF_PAPER_ACTION"
Overheated.	"woyou.aidlservice.jiuv5.OVER_HEATING_ACITON"
Printhead's temperature returns to normal.	"woyou.aidlservice.jiuv5.NORMAL_HEATING_ACITON"
Open the cover.	"woyou.aidlservice.jiuv5.COVER_OPEN_ACTION"
Error occurs when closing the cover.	"woyou.aidlservice.jiuv5.COVER_ERROR_ACTION"
Cutter error 1— Blocked.	"woyou.aidlservice.jiuv5.KNIFE_ERROR_ACTION_1"
Cutter error 2— Fixed.	"woyou.aidlservice.jiuv5.KNIFE_ERROR_ACTION_2"
The printer's firmware starts upgrading.	"woyou.aidlservice.jiuv5.FIRMWARE_UPDATING_ACITON"
Failed to upgrade the printer's firmware.	"woyou.aidlservice.jiuv5.FIRMWARE_FAILURE_ACITON"
No printer has been found.	"woyou.aidlservice.jiuv5.PRINTER_NON_EXISTENT_ACITON"
No black mark has been detected.	"woyou.aidlservice.jiuv5.BLACKLABEL_NON_EXISTENT_ACITON"

Appendix B FAQs of Print Service

1. Description of the print paper's size.

Print paper width is 58 | Valid print width is 48mm | Valid print pixel is 384

Note: SUNMI printers support the print paper of 58mm and 80mm width. This document takes the 58mm print paper as an example to illustrate the printer's supported parameters. 80mm paper has similar specifications.

The width of the 58 print paper is 58mm, and the valid print width is 48mm, with 384 pixels in one line. The depth of V1 paper slot is 40mm which can accommodate the 40mm paper at most.

2. What is the resolution of SUNMI's printers?

The printer's resolution is 205DPI. The calculation formula is:
DPI=384dots/48mm=8dots/1mm=205dots/in=205

3. How to find whether there is a Pprinter?

Desktop devices can be divided into two types: one with printer hardware and one without printer hardware; users can find whether there is a printer from the software through the query interface.

Query interface:

Function: Settings.Global.getInt(getContentResolver(), "sunmi_printer", 0);

Parameter: Fixed value.

Return values

0 → There is a printer.

1 → There is no printer.

-1 → Query in process.

4. Why are the images not printed out when I conduct printing?

First of all, the printer is a line buffer. Currently, devices for the command interface (except QR codes) print out the content filling at least one line, and the content that cannot fill a line will be saved in the buffer. Therefore, if you find the image is not printed out after calling the printBitmap interface, the reason might be that the image width is smaller than the paper width. You can call the linewrap interface to print the buffered image.

If the method stated above failed, you need to check the callback of the interface to see whether there is an error. Generally, an oversized image might be the cause of failing to call the interface. SUNMI printers support images with a resolution smaller than 2M (not the actual size of an image), and the maximum displayable width depends on the paper size. Therefore, you need to adjust the image size to print a suitable image.

Additionally, if you send a raster or bitmap image through Bluetooth hexadecimal command, you need to check whether there is an error in the command, for a byte data error might affect the realization of the whole command.

5. My barcode is too long to fit in a receipt. Which barcode should I use?

Due to the width limit of the print paper (handheld device – 384 pixels wide; desktop device – 576 pixels wide), the barcode with pixels exceeding the limit cannot be fully displayed. The width of the barcode should be adjusted first to see whether this problem can be solved.

If the barcode exceeds 20 digits, code 128 is recommended to use. Please call **printBarCode** interface and select code128. This interface has realized mixed encoding, which means you can add {A, {B, {C to dynamically change encoding types (A: numbers, uppercase letters, and control characters, etc.; B: numbers, uppercase and lowercase letters, and some characters; C: numbers with even number of digits). C type will be used when it is a number, and A/B type will be used when it is another character, and then you can print a long barcode. For example, to print ABab1212: pass "{BABab{C1212".

Mixed encoding is only available for print services with a version 4.0.0 or above. If it is not available for you, or your print via Bluetooth, an array of Epson commands needs to be obtained by introducing the following method, and then send the returned array through sendRawData interface or Bluetooth.

Among it, data is the barcode content to be printed for direct pass; For width, while the default minimum width is 2 pixels, and a width smaller than it will drastically reduce the scanning speed, the width can be set as 1 to print barcodes exceeding 25 or more.

```
public static byte[] getPrintBarCode(String data, int height, int width, int textposition){
    if(width < 1 || width > 6){width = 2;}
    if(textposition < 0 || textposition > 3){textposition = 0;}
    if(height < 1 || height>255){height = 162;}
    ByteArrayOutputStream buffer = new ByteArrayOutputStream();
    try{
        buffer.write(new byte[]{0x1D,0x66,0x01,0x1D,0x48,(byte)textposition,
                0x1D,0x77,(byte)width,0x1D,0x68,(byte)height,0x0A});
       byte[] barcode = checkCode128Auto(data.getBytes());
        buffer.write(new byte[]{0x1D,0x6B,0x49,(byte)(barcode.length)});
        buffer.write(barcode);
    }catch(Exception e){
        e.printStackTrace();
    }
    return buffer.toByteArray();
}
```

```
public static byte[] checkCode128Auto(byte[] data){
    ByteArrayOutputStream temp = new ByteArrayOutputStream();
    int pos = 0;
    boolean mode_C = true;
    temp.write(0x7b);
    temp.write(0x43);
    while(pos < data.length){
        if(data[pos] == '{' && pos + 1 != data.length){
            switch (data[pos + 1]){
                case 'A':
                case 'B':
                    if(mode_C){mode_C =
false;temp.write(data[pos++]);temp.write(data[pos++]);
                    }else{pos++;pos++;}
                    break;
                case 'C':
                    if(!mode_C){mode_C =
true;temp.write(data[pos++]);temp.write(data[pos++]);
                    }else{pos++;pos++;}
                    break;
                default:
                    break;}
        }else if(pos + 1 == data.length){
            if(mode_C){temp.write(0x7b);temp.write(0x42);mode_C = false;}
            temp.write(data[pos++]);
        }else if(data[pos] < '0' || data[pos] > '9'){
            if(mode_C){temp.write(0x7b);temp.write(0x42);mode_C = false;}
            temp.write(data[pos++]);
        } else if(data[pos+1] < '0' || data[pos+1] > '9'){
            if(mode_C){temp.write(0x7b);temp.write(0x42);mode_C = false;}
            temp.write(data[pos++]);
            temp.write(data[pos++]);
        }else{
            if(!mode_C){temp.write(0x7b);temp.write(0x43);mode_C = true;}
            int left = data[pos] - '0';
            int right = data[pos + 1] - '0';
            int num = left*10 + right;
            if(num < 0 || num > 99){
                return null;
            }else{
                temp.write(num);
```

6. Why can't I receive the callback results?

AIDL resource documents – P series, V1S and V2; AIDL resource documents - T series and S series;

AIDL resource documents – Mini-series; AIDL resource document - series:

First of all, if you access the interface via AIDL method, please note whether the suitable AIDL resource has been used.

On the premise of using the suitable resource document, please note that you should introduce the static nested class Stub! in callback class generated by AIDL when you. Create the callback object.

~~new ICallback(…)~~ —> new ICallback.Stub()

This problem will not occur if you use remote imported library and follow the document to use the encapsulated InnerPrinterCallback class, etc.

7. Select and set the character set.

Background: The built-in printers are compatible with the transmission in binary byte stream form, so a character set must be selected and set for the text sent in the form of byte code. Multi-byte, GB18030 character encoding is the default setting.

The single-byte encoding types supported by the built-in printers are:

```
[Parameter] [Encoding] [Country]
0 "CP437"; American, European standard.
2 "CP850"; Multiple languages.
3 "CP860"; Portuguese.
4 “CP863"; Canada – French.
5 “CP865"; Northern Europe.
13 "CP857"; Turkish.
14 "CP737"; Greek.
15 "CP928";
16 "Windows-1252";
17 "CP866"; Cyrillic.
18 "CP852"; Latin – Central European.
19 "CP858";
21 "CP874";
33 "Windows-775"; Balticc.
34 "CP855"; Cyrillic.
36 "CP862"; Hebrew.
37 "CP864";
254 "CP855"; Cyrillic.
```

The multi-byte encoding types supported by the built-in printers are:

```
[Parameter] [Encoding]
0x00 || 0x48 "GB18030";
0x01 || 0x49 "BIG5";
0xFF  “utf-8”;
```

The device settings can be adjusted in accordance with the needs of different countries or other requirements to enable the printers to identify the data stream of the print content. To print CP437, please send:

0x1C 0x2E ——Set it as the single-byte encoding type.

0x1B 0x74 0x00 ——Set it as the CP437 of the single-byte code page.

To print CP866, please send:

0x1C 0x2E ——Set it as the single-byte encoding type.

0x1B 0x74 0x11 ——Set it as the CP866 of the single-byte code page.

To print the content in traditional Chinese, please send:

0x1C 0x26 ——Set it as the multi-byte encoding type.

0x1C 0x43 0x01—— Set it as the BIG5 encoding of the multi-byte code page.

To print UTF-8 encoding content (all Unicode character sets are supported if using UTF-8 encoding, and all the contents can be printed), please send:

0x1C 0x26 ——Set it as the multi-byte encoding type.

0x1C 0x43 0xFF—— Set it as the UTF-8 encoding of the multi-type code page.

8. Description of black mark mode.

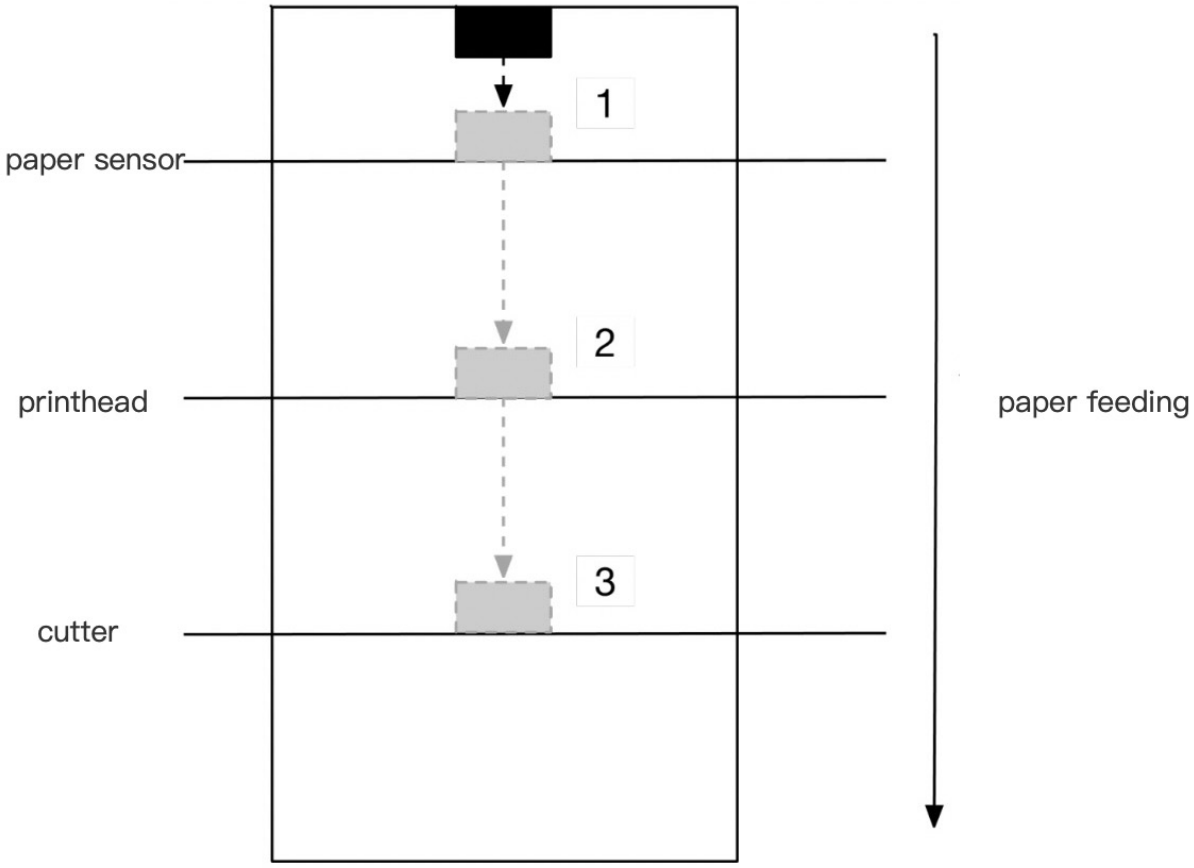
SUNMI T1 printers can print using specification-compliant print papers with black marks.

Requirements on the print paper with black marks: Different from other black mark printers, SUNMI T1 printer has no sensor to detect black marks, but provide a function similar to black mark printing by using the principle that the sensor considers materials with a reflectivity not bigger than 6% (Infrared wavelength range of 700-1000nm) as no paper. Due to different principles, SUNMI T1 printers in black mark print mode cannot accurately locate the black mark as other black mark printers and it may have some errors.

Requirement on the location of the black mark: The black mark should be in the horizontal center for the paper sensor to detect.

The principle SUNMI T1 uses to realize black mark printing:

The paper sensor and the cutter are not located in the same horizontal line. The black mark on the paper will first run through the paper sensor (the location of 1) and then run through the printhead (the location of 2) and finally arrive the location of the cutter (the location of 3).



In black mark printing mode, if the sensor detects that there is no paper, it will automatically feed paper 7mm. If still no paper is detected, the printer will process it as no paper; and if paper is detected after that, the printer will process it as a black mark.

If the print content covers the black mark, printing will be continued, which means if the black mark arrives at the location of 2, there is still a print job, and it will continue until it is completed.

Command sequence in the black mark printing mode:

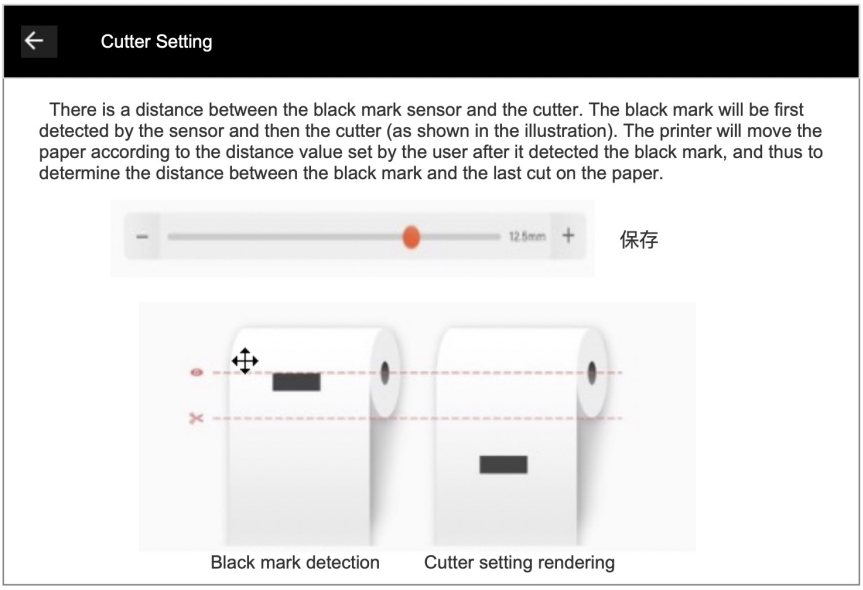
1. Send the content to be printed.

2. Enter the command of feeding paper until the black mark is detected {0x1c, 0x28, 0x4c, 0x02, 0x00, 0x42, 0x31}.

3. Enter the command of using the cutter {0x1d, 0x56, 0x00}.

Then the printer will automatically detect next black mark after printing the content, and use the cutter in a specific location.

Users can modify the black mark printing mode and the location of cutter in Setting->Print->Built-in print management.



9. How to print special symbols?

If there is a need to print some special symbols such as currency: $, ¥, €, £, ₣, etc., the following methods can be used:

The easiest way is to directly call the print interface we provide, printText, to print.

printText("$ ¥ € £ ₣ \n", null)

The character set for printing special symbols may not be the same as different currency symbols correspond to page numbers of different countries. So if you are printing via Bluetooth or ESC commands, the utf-8 characters we specify (The default is GB18030) are recommended. You can refer to FAQ7 for the character set settings.

For example, if you want to print currency symbols like $, ¥, €, £, and ₣:

You need to send the command byte[] data = new byte[]{0x1C, 0x26, 0x1C, 0x43, 0xFF}, set the printer to receive utf-8 character encoding;

After that, send the symbol data "$¥€£₣".getBytes("utf-8"), i.e.:

sendData(new byte[]{0x1C, 0x26, 0x1C, 0x43, 0xFF}, null)

sendData("$¥€£₣".getBytes("utf-8"))