SUNMI Inbuilt Printer
Developer Documentation

ntroduction

There are inbuilt thermal printers equipped in SUNMI terminals, which allows you to print thermal receipt directly through SDK from your APP. The terminals with inbuilt thermal printers are:
Mobile terminals: V1, V1s, V2 Pro, etc.
Payment terminals: P1, P1-4g, etc.
Desktop terminals: T1, T2, T1mini, T2mini, etc.
Desktop scale terminals: S2, etc.

2 inbuilt printer specs are available:
- 80mm paper width, cutter adopted and compatible to 58mm-wide paper, such as the inbuilt printer of T1 model;
- 58mm paper width, cutter not adopted, such as the inbuilt printer of V1 model.

You can call the inbuilt thermal printer by the following methods:
- **Call the printer through the inbuilt printing service interface:** this method suits you best if it is the first time for you to develop the printing-related app and do not know much about Epson commands, and want to achieve the desired printing effect through multiple printing interfaces provided by our printing service.
- **Call the printer through the inbuilt virtual Bluetooth:** this method suits you best if you have developed Bluetooth or USB printers before or your app has realized Bluetooth printing and can achieve printing effect by changing some codes only.

1. Call the printer through the inbuilt printing service interface

1.1 Connect the inbuilt printing service

1.1.1 By remote dependency

To facilitate the calling of inbuilt printing service, we recommend you to use remote dependency library if you use gradle configuration, and since our dependency library is adaptive to several terminal models, the interfaces of different models will be automatically distinguished by using remote dependency mode, thus there is no need to configure different AIDL files for different models, and an exception message will be returned if the wrong interface is used.

Integration ways

You have to confirm that the central repository has been configured in the build.gradle file in the root directory: mavenCtral
Then you have to add dependency and attribute configuration in the build.gradle file in the app directory:

```
dependencies { implementation ‘com.sunmi:printerlibrary:1.0.15’ }
```

For V2s and V2s Plus models, due to the requirements of Android version, you need to add package visibility declaration before connecting to the service, and add the following commands in AndroidManifest.xml:

```
<queries>
        <package android:name=“woyou.aidlservice.jiuiv5” />
</queries>
```

Initialization

Like binding a service component, here we call the binding method with singleton pattern:

```
boolean result = InnerPrinterManager.getInstance().bindService(context,
```

```
innerPrinterCallback);
 InnerPrinterCallback innerPrinterCallback = new InnerPrinterCallback(){

 @Override
 protected void onConnected(SunmiPrinterService service){
//This service is the handle of remote service API
//You can call the supported printing method by service
}

 @Override
 protected void onDisconnected() {
//This method will be called back when the service is disconnected abnormally. We advise to reconnect here.
}
 }
```

Result: indicating binding succeeded or failed

Simple calling of printing method

```
try{
 service.printText(“the content to be printed\n”, new InnerResultCallbcak() {

@Override public void onRunResult(boolean isSuccess) throws
RemoteException
 {
//Return the result of interface implementation (not a real printing): succeeded or failed
}

@Override
 public void onReturnString(String result)
throws RemoteException
 {
 //Queried data will be returned asynchronously for some interfaces
 }
@Override
 public void onRaiseException(int code, String msg) throws
RemoteException
 {
//Description of the abnormal status returned when the interface failed to implement
}
 @Override
 public void onPrintResult(int code, String msg)
Throws RemoteException
 {
//Return the real printing result under transaction mode
}
 });
 } catch (RemoteException e) {
 //Some interfaces can only be used for specified models, so interface calling exceptions may occur. For example, the cash drawer interface can only be used for the desktop terminals.
}
```

Completion & release

The following method can be called after a call completes to disconnect the service.
InnerPrinterManager.getInstance().unBindService(context,
innerPrinterCallback);

Class description

InnerPrinterManager: management class of print library, which is mainly used to realize the connection and disconnection of printing service.
InnerPrinterCallback: callback interface connecting to the remote service which is used to get the connection result of remote service.
SunmiPrinterService: SUNMI printing service interfaces define all the printing methods. You can get such instance after connecting to service, which is the same as AIDL IWoyouService interface.
InnerPrinterException: printing exception class; an error or exception occurs when calling a method; some models don't have specific methods or functions. For example, handheld terminals don't have the cash drawer function, and printing exception will be returned when calling cash drawer opening interface, which leads to the failure of corresponding interface use.
InnerResultCallback: callback interface of printing method, which is used to get the implementation result of interface method.
InnerLcdCallback: callback interface of customer display method, which is used to get the implementation result of customer display method.
InnerTaxCallback: callback interface of tax control method, which is used to get the result of tax control sending.

1.1.2 By using AIDL

AIDL introduction

AIDL, or Android Interface Definition Language, allows us to define the programming interfaces for interprocess communication. We provide AIDL with common print commands encapsulated for you to quickly integrate SUNMI printers.

How to use AIDL

Five steps to complete the connection:
Step 1: Add AIDL file enclosed in the resource document in the program, and package path and package name can't be modified (java files included for some device models);
Step 2: Implement ServiceConnection in the code class for print control;
Step 3: Call ApplicationContext.bindService(), and pass it during the implementation of ServiceConnection.
Note: bindservice is not a blocking call, which means that the binding has not been completed after calling, and please refer to the callback of serviceConnected;
Step 4: During the implementation of ServiceConnection.onServiceConnected(), you will receive an IBinder instance (the Service called). Convert the parameter into IWoyouService type by calling IWoyouService.Stub.asInterface(service);
Step 5: Now you can call all kinds of methods defined in IWoyouService interface to print.

AIDL resource download

!Note: the interfaces of models may vary depending on different models. Please pay attention to models for corresponding AIDL resource packages. Currently four resource packages are available:

old (V1) for SUNMI first V1 model, including:
IWoyouService.aidl: interface files of printer
ICallback.aidl: callback interface files of printer
TransBean.aidl
TransBean.java: page print file

handheld (handheld terminals except V1 model) for SUNMI handheld terminals, such as V1s, V2, P2, etc., including:
IWoyouService.aidl: interface files of printer
ICallback.aidl: callback interface files of printer
TransBean.aidl
TransBean.java: page print file
ITax.aidl: tax control callback interface file

desktop (desktop terminals) for SUNMI desktop printers, such as T1, T2, etc., including:
IWoyouService.aidl: interface files of printer
ICallback.aidl: callback interface files of printer
ITax.aidl: tax control callback interface file

desktop + lcd (desktop + customer display) for SUNMI printers with customer display, such as T1mini, T2mini, etc., including:
IWoyouService.aidl: interface files of printer
ICallback.aidl: callback interface files of printer
ITax.aidl: tax control callback interface file
ILcdCallback.aidl: customer display callback interface file

General package:
WoyouConsts.java: it defines the const class exposed by the printer and is used to set the configuration.

Binding service examples:

Realize 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);
}
```

Binding printing service

```
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 Interface definition description

After connecting the inbuilt printing service through the above methods and getting IWoyouService or SunmiPrinterService objects, you can print by calling the following interfaces.

1.2.1 Printer initialization and setting

No.	Methods
1	void printerInit( ) Printer initialization
2	void printerSelfChecking(ICallback callback) Printer self-checking

1. Printer initialization

Function: void printerInit( )

Note: reset the printer's logical program, such as typography, bold, etc., without emptying the cached data, then the unfinished print tasks will continue after resetting.

2. Printer self-checking

Function: void printerSelfChecking (ICallback callback)

Parameter: callback → result callback

Example:

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

1.2.2 Get device and printer information

No.	Methods
1	String getPrinterSerialNo( ) Get the printer’ s serial no.
2	String getPrinterModal( ) Get the interface of printer model
3	String getPrinterVersion( ) Get the printer’ s firmware version no.
4	Build.MODEL (const class) Device name
5	int updatePrinterState() Get the printer’ s latest status
6	String getServiceVersion() Get the printing service version no.
7	int getPrintedLength(ICallback callback) Get the print length of the printhead
8	int getPrinterPaper() Get the printer’ s current paper specification

1. Get the printer’ s latest status

Function: String updatePrinterState ()

Return value:

1 → Printer is under normal operation

2 → Printer is under preparation
3 → Communication is abnormal
4 → Out of paper
5 → Overheated
6 → Cover is open
7 → Cutter error
8 → Cutter recovered
9 → Black mark not detected
505 → Printer not detected
507 → Printer firmware update failed
Note 1: these return values are applicable to all SUNMI devices, but some status can't be obtained due to hardware configuration. For example, cover open detection is not applicable to handheld devices.

Note 2: V1 devices currently can't support this interface; you can also get it asynchronously by registering broadcast apart from getting status proactively, please see Appendix A.

2. Get the printer's print length
Function: int getPrintedLength(ICallback callback)
Note: currently you can get the printed length since powering on the device. Due to the hardware differences between desktop devices and handheld devices, the returns of printing results may vary. In other words, you can get the printed length through ICallback callback interface for handheld devices, and get the printed length directly through the return values for desktop devices.

3. Get the printer's current paper specification
Function: int getPrinterPaper()
Note 1: by default, 58mm paper specification is adopted for handheld printers, and 80mm paper specification for desktop printers but you can add a fixator and set the printer to use a 58mm paper roll by configuration, and the interface will return the current paper specification set for the printer.
Note 2: currently, desktop device T1 model with versions above v2.4.0 supports this interface; T2 and S2 models with versions above v1.0.5 support this interface; for other device models with versions above v4.1.2, they all support this interface for getting paper specification.

1.2.3 ESC/POS commands

No.	Method
1	void sendRAWData (byte[] data, ICallback callback) Commands of printing ESC/POS format

1. Commands of printing ESC/POS format
Function: void sendRAWData(byte[] data, ICallback callback)
Parameter:
data → ESC/POS commands
callback → result callback
Note: for related commands, please see *ESC/POS Command Set*.
Example:
```
woyouService.sendRAWData(new byte[]{0x1B,0x45,0x01},
callback); // 1B,45, 01 is the command for bold font
```

1.2.4 Instruction for printer style setting interface

No.	Method
1	void setPrinterStyle(int key, int value)

	Set printer style

1. Set printer style

Function: void setPrinterStyle(int key, int value)

Parameter: set the const class by referring to WoyouConsts.java printer style.

key → set different attributes according to the definition in the const class interface, usually being ENABLE_XXX and SET_XXX.

value → set status or size corresponding to the attributes

Select ENABLE or DISABLE for ENABLE XXX attribute.

Set the detailed size for SET_XXX attribute.

Note: this interface is available for the printing service with versions above v4.2.22.

Example:

```
woyouService.setPrinterStyle( WoyouConsts.ENABLE_ILALIC,
WoyouConsts.ENABLE); // Italic font for the subsequent text to be printed
woyouService.setPrinterStyle(WoyouConsts.SET_LINE_SPACIN
G,123); // Change line spacing to 123 pixels for the subsequent text to be printed
```

1.2.5 Change print mode

No.	Methods/Commands
1	int getPrinterMode() Get printer mode
2	int getPrinterBBMDistance() Get paper moving distance set under black mark mode
3	Related mode setting, please set in the system ‘Setting’ → ‘Print Settings’.

1. Get printer mode

Function: int getPrinterMode()

Return value:

0 → General mode

1 → Black mark mode

2 → Label mode

Note: black mark mode currently is available for SUNMI T1, T2 desktop terminals;

Label mode currently is available for SUNMI V2, V2 Pro handheld terminals.

2. Get the printer’s auto paper moving distance under black mark mode

Function: int getPrinterBBMDistance()

Return value: paper moving distance (number of pixels)

Note: only available for T1, T2 devices.

1.2.6 Text printing

No.	Methods/Commands
1	void setAlignment(int alignment, ICallback callback) Set alignment mode.
2	void setFontName(String typeface, ICallback callback) Set custom print font.
3	void setFontSize(float fontsize, ICallback callback) Set font size.
4	esc/pos commands: bold font {0x1B, 0x45, 0x1}, cancel bold {0x1B, 0x45, 0x0} Set or cancel bold.
5	void printText(String text, ICallback callback)

	Print text.
6	void printTextWithFont(String text, String typeface, float fontSize, ICallback callback) Print text with designated font and size.
7	void printOriginalText (String text, ICallback callback) Print vector font.

1. Set alignment mode

Function: void setAlignment (int alignment, ICallback callback)

Parameter:

alignment → alignment mode: 0 → left, 1 → center, 2 → right

callback → result return

Note: global method may influence the subsequent print implementation, and you can cancel the related settings when initializing the printer.

Example:

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

2. Set custom print font

Function: void setFontName (String typeface, ICallback callback)

Parameter:

typeface → specify the custom font name you're about to use. Currently only vector font supported, and the font needs to be preset in the app assets directory.

callback → result return

Note 1: this interface can expand the default font type in the printer for you to use a custom font. However, you have to adjust the line spacing and line width due to the inconsistent inner width of each font.

Note 2: this interface is available for the printing service with versions above v4.14.0.

3. Set font size

Function: void setFontSize (float fontSize, ICallback callback)

Parameter:

fontSize → font size

callback → result return

Note: global method may influence the subsequent printing, and you can cancel this setting using printer initialization. The font size adopts the printing method exceeding the standard international commands, and the adjustment of font size may affect the character width, which leads to the change to the number of each line's characters. So typography of monospaced fonts can be confusing some time.

Example:

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

4. Set and cancel bold

Commands: bold font {0x1B, 0x45, 0x1}, cancel bold {0x1B, 0x45, 0x0}

Note: please see 1.2.3 ESC/POS Commands in this document.

Example:

```
woyouService.sendRAWData(new byte[]{0x1B,0x45,0x0},
callback); // cancel bold font command
```

5. Print text

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

Parameter:

text → text to be printed; **automatic linefeed** will be implemented when the text width exceeds one line, and the forced newline character "\n" shall be added at the end of one line when the text width is less than one line or exceeds one line but not meeting one line before being printed, or it will be cached.

callback → result return

Note: if you need to change the style of print text, such as alignment mode, font size, bold, etc., please set it before calling printText method.

Example:

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

6. Print text with specified font and size

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

Parameter:

text → text to be printed; **automatic linefeed** will be implemented when the text width exceeds one line, and the forced newline character "\n" shall be added at the end of one line when the text width is less than one line or exceeds one line but not meeting one line before being printed, or it will be cached.

Typeface → the custom font name. Currently only vector fonts are supported, and the font needs to be preset in the app assets directory.

fontSize → font size; only available for this method.

Callback → result return

Note: the font setting of this interface is available for the printing service with versions above v4.14.0.

Example:

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

7. Print vector text

Function: void printOriginalText(String text, ICallback callback)

Parameter:

text → text to be printed; **automatic linefeed** will be implemented when the text width exceeds one line, and the forced newline character "\n" shall be added at the end of one line when the text width is less than one line or exceeds one line but not meeting one line before being printed, or it will be cached.

callback → result return

Return value:

Note: the text is printed according to the vector text width, which means that each character is not monospaced.

Example:

```
woyouService.printOriginalText ("κ ρ χ κ μ ν κ λ ρ κ ν κ ν μ ρ τ υ ϕ\n",
callback);
```

1.2.7 Print a table

No.	Methods/Commands
1	void printColumnsText(String[] colsTextArr, int[] colsWidthArr, int[] colsAlign, ICallback callback) Print a column of a table (Arabic characters are not supported)
2	void printColumnsString(String[] colsTextArr, int[] colsWidthArr, int[] colsAlign, ICallback callback) Print a column of a table, and you can specify the column width and alignment mode.

1. Print a column of a form (Arabic characters are not supported)

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

Parameter:

colsTextArr → Array of column text strings.

colsWidthArr → Array of each column width, calculated in English characters, and each Chinese character is equal to two English characters, with each width > 0.

colsAlign → Alignment mode of each column: 0: left; 1: center; 2: right.
callback → result return
Note: the array length of the above three parameters should be consistent. If the width of colsText[i] is larger than that of colsWidth[i], the text will be changed to another line, and Arabic characters are not supported.
Example:
woyouService.printColumnsText(new String[]{"SUNMI","SUNMI","SUNMI"}, new int[]{4,4,8}, new int[]{1,1,1},callback);

2. Print a column of a form, and you can specify the column width and alignment mode
Function: void printColumnsString(String[] colsTextArr, int[] colsWidthArr, int[] colsAlign, ICallback callback)
Parameter:
colsTextArr → Array of column text strings.
colsWidthArr → The width weight of each column is the proportion of each column.
colsAlign → Alignment mode of each column: 0: left; 1: center; 2: right.
callback → result return
Note: the array length of the above three parameters should be consistent. If the width of colsText[i] is larger than that of colsWidth[i], the text will be changed to another line.
Example:
woyouService.printColumnsString(new String[]{"SUNMI","SUNMI","SUNMI"}, new int[]{1,1,2}, new int[]{1,1,1},callback);

1.2.8 Print an image

No.	Methods
1	void printBitmap(Bitmap bitmap, ICallback callback) Print an image
2	void printBitmapCustom(Bitmap bitmap, int type, ICallback callback) Print an image (2)

1. Print an image
Function: void printBitmap (Bitmap bitmap, ICallback callback)
Parameter:
bitmap → image: Bitmap object;
callback → result return
Note: the maximum pixel size of the image should be less than 2.5 million pixels of width x height, and the width should be set according to the size of paper specification (384 pixels for 58mm paper, and 576 pixels for 80mm paper). If it exceeds the width of the paper, it will not be displayed.
Example:
woyouService.printBitmap(bitmap,callback);

2. Print an image (2)
Function: void printBitmapCustom (Bitmap bitmap,int type ICallback callback)
Parameter:
bitmap → image: bitmap object; the biggest width is 384 pixels, and the image can’ t be printed when exceeding 1M.
type → currently two printing methods are available:
0 → the same as printBitmap();
1 → black and white image with a threshold of 200
2 → grayscale image
callback → result return
Note: the image resolution is less than 2 million pixels, and the width should be set according to the size of paper specification (384 pixels for 58mm paper, and 576 pixels for 80mm paper). If it exceeds the width of the paper, it will not be displayed.

Versions supported: v3.2.0 above for P1; v1.2.0 above for P14g; v3.2.0 above for V1s; v1.0.0 above for V2; v2.4.0 above for T1; v1.0.5 above for T2, S2; v2.4.1 above for T1mini; v1.0.0 above for T2mini.

Example:
woyouService.printBitmapCustom(bitmap,callback);

1.2.9 Print a 1D/2D barcode

No.	Methods
1	void printBarCode(String data, int symbology, int height, int width, int textPosition, ICallback callback) Print a 1D barcode.
2	void printQRCode(String data, int modulesize, int errorlevel, ICallback callback) Print a QR code.
3	void print2DCode(String data, int symbology, int modulesize, int errorlevel, ICallback callback) Print a 2D barcode.

1. Print a 1D barcode

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

Parameter:

data → 1D barcode content

symbology → barcode type (0 – 8):

0 → UPC-A

1 → UPC-E

2 → JAN13 (EAN13)

3 → JAN8 (EAN8)

4 → CODE39

5 → ITF

6 → CODABAR

7 → CODE 93

8 → CODE128

height → barcode height from 1 – 255, with default value of 162

width → barcode width from 2 – 6, with default value of 2

textPosition → text position (0 -3)

0 → not to print text

1 → text is above barcode

2 → text is below barcode

3 → print text above and below the barcode

Callback → result return

Note: different barcode types have the following differences:

Code	Description
code39	A maximum of 13 digits can be printed
code93	A maximum of 17 digits can be printed
ean8	8 digits required, with the last one as a check digit, and the valid length of 8 digits
ean13	The valid length consists of 13 digits, the last of which is a check digit
ITF	Require to enter number, with valid length less than 14 digits, and must be an even number
Codebar	Require to enter numbers from 0 – 9, with 6 special characters, and a maximum of 18 digits can be printed
UPC-E	8 digits required, with the last one as a check digit
UPC-A	12 digits required, with the last one as a check digit

code128	Three types for Code128: Type A: including uppercase letters, numbers, and punctuation marks; Type B: uppercase letters, lowercase letters, and numbers; Type C: pure digits, plural characters, and the last digit is ignored if it is a singular digit; The interface adopts type B code by default. To use type A and C codes, you need to add “{A” and “{C” before the content, for example, “{A2344A”, ”{C123123”, ”{A1A{B13B{C12”.

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

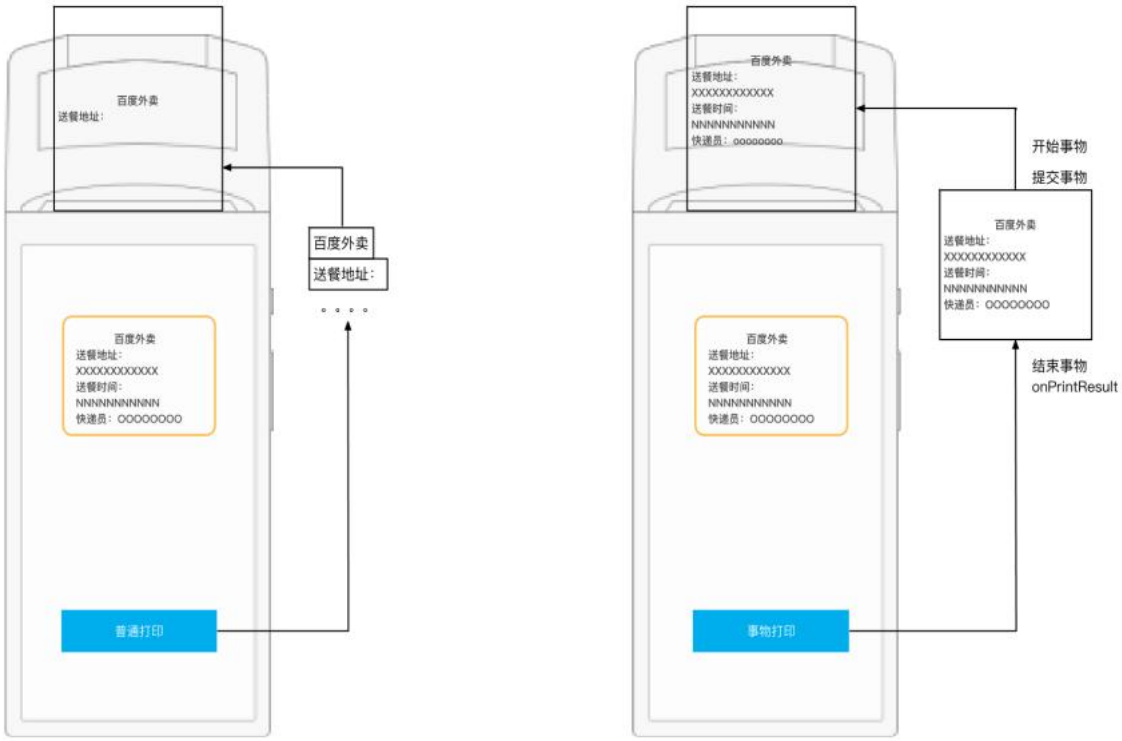
2. Print a QR code
Function: void printQRCode (String data, int modulesize, int errorlevel, ICallback callback)
Parameter:
data → QR barcode content
modulesize → the size of QR barcode; unit: pixel; value from 4 to 16;
errorlevel → barcode error correction level (0-3):
0 → error correction level L (7%)
1 → error correction level M (15%)
2 → error correction level Q (25%)
3 → error correction level H (30%)
callback → result return
Note: the printing content will be directly output after calling this method under normal printing status, and each barcode has 4 pixels (if the pixels are less than 4, the barcode scanning may fail). A maximum version19 (93*93) mode is supported.
Example:
woyouService.printQrCode("SUNMI TECHNOLOGY", 4, 3, callback);

3. Print a 2D barcode
Function: void print2DCode(String data, int symbology, int modulesize, int errorlevel, ICallback callback)
Parameter:
data → 2D barcode content
symbology → 2D barcode type
1 Qr (same as printQRCode interface)
2 PDF417
3 DataMatrix
modulesize → the size of effective 2D barcode; the supported optimum barcode size varies according to different barcode types.
Qr barcode: 4 – 16 (the same as printQRCode interface)
PDF417: 1 – 4
DataMatrix: 4 – 16
errorlevel → 2D barcode error correction level; the supported level varies according to different barcode types.
Qr barcode: 0 – 3 (same as printQRCode interface)
PDF417: 0 – 8
DataMatrix: ECC200 auto error correction is adopted by default, and it cannot be modified.
callback → result return
Note: the printing content will be directly output after calling this method under normal printing status; this interface is available for versions above v4.1.2.

1.2.10 Transaction printing

Transaction printing mode is applicable to requests needing to control printing content and get printing result feedback (whether to print a receipt), which is equal to set up a transaction queue buffer area. When you implement transaction printing mode, you’ll enable a transaction queue, and you can add print methods to it, the printer won’t start printing

immediately. After submitting transactions, the printer will start to implement the printing tasks in the queue, and the result feedback can be obtained after the implementation.



Notices of transaction printing:

1. After implementing buffer (transaction) printing, there will be a succeeded result returned if you submitted the print job successfully. However, **all the commands in this transaction will be discarded and exceptions feedback will be given when the printer encounters exceptions such as out of paper, overheated, etc.**, that is, this print job won't be output when the printer encounters exceptions before implementation or under implementation.
2. When commands printing and buffer (transaction) printing are implemented interactively, the content of commands printing won't be cleared if the printer encounters exceptions.
3. After enabling transaction printing mode, you can use the methods for other interfaces in 1.2.x to output the content, but the content won't be printed out immediately, and will be cached in the cache area. You can implement the printing by calling **exitPrinterBuffer()** or **commitPrinterBuffer()**.
4. The transaction printing result callback can be implemented by **onPrintResult(int code, String msg)** method in **ICallback** method (it may take some time since you have to wait for the printer to move the paper, so we recommend not to implement transaction printing too frequently for a single line, which may affect the print speed. We recommend you to implement transaction printing when printing a whole receipt). The corresponding return codes are as follows:

a) 0 → printing succeeded, the msg is "Transaction print successful!"

b) 1 → printing failed, the msg is "Transaction print failed!"

5. The pseudocode example of the whole transaction printing is as follows:

```
enterPrinterBuffer(true) - - enable transaction mode, and all the subsequent commands won't be output immediately
printText(/*something*/)
printBitmap(/*bitmap resource*/)
…… other related print methods - - print some content
commitPrinterBuffer()/commitPrinterBufferWithCallback(callback) - - submit a transaction, and the printer will start printing. A result (succeeded or failed) will be returned in callback.
……wait for the return of last transaction
printText(/*something*/)
printBitmap(/*bitmap resource*/)
```

…… other related print methods – – you can choose to wait for or not wait for the return of last transaction, and 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 when disabling the transaction mode, and the printer will continue to print if you input new data after last submission, or it will stop printing

6. Descriptions of specific methods

No.	Methods
1	void commitPrint(TransBean[] tranBean, ICallback callback) Specific interface for lib package transaction printing
2	void enterPrinterBuffer(boolean clean) Enable transaction printing mode
3	void exitPrinterBuffer(boolean commit) Disable transaction printing mode
4	void exitPrinterBufferWithCallback(boolean commit, ICallback callback) Disable transaction printing mode and call back result
5	void commitPrinterBuffer() Submit transaction printing
6	void commitPrinterBufferWithCallback(ICallback callback) Submit transaction printing and call back result

1. Specific interface for lib package transaction printing

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

Parameter:

tranBean → printing task list

callback → result callback

example:

woyouService.commitPrint(tranBean, callback);

2. Enable transaction printing mode

Function: void enterPrinterBuffer(Boolean clear)

Parameter:

clear → whether to clear the content in the buffer area

true → clear the content unsubmitted in the last transaction printing

false → not clear the content unsubmitted in the last transaction printing, which will be included in the next submission

Note:

1. After enabling transaction printing mode, the printer won't print the data immediately under this mode, until you submit transaction or disable submitting transaction;
2. Transaction printing mode is available for all devices except for V1 model.

Example:

woyouService.enterPrinterBuffer(false);

3. Disable transaction printing mode

Function: void exitPrinterBuffer(Boolean commit)

Parameter:

commit → whether to print the content in the buffer area;

true → the printer will print all the contents in the transaction queue;

false → the printer will not print all the contents in the transaction queue, which will be saved for the next submission

Note: transaction printing mode is available for all devices except for V1 model.
Example:
woyouService.exitPrinterBuffer(true);

4. Disable transaction printing mode and call back result
Function: void exitPrinterBuffer(Boolean commit, ICallback callback)
Parameter:
commit → whether to print the content in the buffer area;
true → the printer will print all the contents in the transaction queue;
false → the printer will not print all the contents in the transaction queue, which will be saved for the next submission
callback → result callback
Note: transaction printing mode is available for all devices except for V1 model.
Example:
woyouService.exitPrinterBuffer(true);

5. Submit transaction printing
Function: void commitPrinterBuffer()
Note:
1. Submit all the contents in the transaction queue and print all. The printer remains in the transaction printing mode;
2. Transaction printing mode is available for all devices except for V1 model.
Example:
woyouService.commitPrinterBuffer();

6. Submit transaction printing and call back result
Function: void commitPrinterBufferWithCallbacka(ICallback callback)
Parameter:
callback → result callback
Note: transaction printing mode is available for all devices except for V1 model.
Example:
woyouService.commitPrinterBufferWithCallback(callback);

1.2.11 Paper moving related

No.	Method
1	void lineWrap(int n, ICallback callback) The printer moves the paper for n lines

1. The printer moves the paper for n lines
Function: void lineWrap(int n, ICallback callback)
Parameter:
n → the number of lines moved
callback → result callback
Note: forced linefeed is adopted, and the printer will move the paper for n lines after completing the previous printing.
Example:
woyouService.printBitmap(3, callback);

1.2.12 Cutter (paper cutting) related

No.	Methods
1	void cutPaper(ICallback callback)

	Paper cutting
2	int getCutPagerTimes() Get the cutter’s cumulative cutting times

1. Paper cutting

Function: void cutPaper (ICallback callback)

Parameter: callback → **result callback**

Note: there’s some distance between printhead and cutter, which will be automatically complemented by calling the interface.

Only available for desktop terminals with cutter function.

Example:

woyouService.cutPager(callback);

2. Get the cutter’s cumulative cutting times

Function: int getCutPaperTimes ()

Return value: the cutter’s cutting times

Note: Only available for desktop terminals with cutter function.

1.2.13 Cash drawer related

No.	Methods
1	void openDrawer(ICallback callback) Open the cash drawer
2	int getOpenDrawerTimes() Get the cumulative open times of the cash drawer
3	int getDrawerStatus() Get the current status of the cash drawer

1. Open the cash drawer

Function: void openDrawer (ICallback callback)

Parameter:

callback → **result callback**

Note: Only available for desktop terminals with cash drawer function.

Example:

woyouService.openDrawer(callback);

2. Get the cumulative open times of cash drawer

Function: int getCutPaperTimes ()

Return value: the cumulative open times of the cash drawer

Note: Only available for desktop terminals with cash drawer function.

3. Get the current status of cash drawer

Function: int getDrawerStatus()

Note 1: you can get the cash drawer status of some models with cash drawer function by calling this interface.

Note 2: this interface is only available for device models of S2, T2, and T2mini with versions above v4.0.0.

1.2.14 Get global attributes

You can override the custom styles by configuring some printing styles in SUNMI devices, and the following interfaces can help you get the current enabled configuration status.

No.	Methods
1	int getForcedDouble() Get global font height and width status

2	boolean isForcedAntiWhite() Get global font anti-white style enabled
3	boolean isForcedBold() Get global font bold style enabled
4	boolean isForcedUnderline() Get global font underline style enabled
5	int getForcedRowHeight() Get global line height set value
6	int getFontName() Get current font used
7	int getPrinterDensity() Get print density

Note: currently, these interfaces, except 'get print density' interface, are only available for handheld terminals of models V1, V1s, and P1 with versions above v3.2.0, and model P14g with versions above v1.2.0.

1.2.15 Customer display interface description

Mini-series devices have customer display functions, which can be used by calling the following interfaces:

No.	Methods
1	void sendLCDCommand(int flag) Send control commands
2	void sendLCDString(String string, ILcdCallback callback) Send single line text
3	void sendLCDDoubleString(String topText, String bottomText, ILcdCallback callback) Send double lines text
4	void sendLCDMultiString(String[] text, int[] align, ILcdCallback callback) Send multiple lines text, and each line's content will be automatically sized based on its weight
5	void sendLCDFillString(String string, int size, boolean fill, ILcdCallback callback) Send single line text (font size and filling method can be customized for the text)
6	void sendLCDBitmap(Bitmap bitmap, ILcdCallback callback) Send bitmap image

1. Send control commands

Function: void sendLCDCommand(int flag)

parameter: flag → 1. Initialization; 2. Wake up LCD; 3. LCD hibernation; 4. Clear screen

Note: only available for desktop terminals of mini series with customer display function.

2. Send single line text

Function: void sendLCDString(String string, ILcdCallback callback)

Parameter:

string → the text to be displayed

callback → asynchronous callback on implementation result

Note: only available for desktop terminals of mini series with customer display function, and it won't be displayed if the text is too long.

3. Send double lines text

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

Parameter:

topText → display the top text

bottomText → display the bottom text

callback → asynchronous callback on implementation result

Note: only available for desktop terminals of mini series with customer display function, and it won't be displayed if the text is too long.

4. Send multiple lines text

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

Parameter:

text → arrays displaying the text of each line. You need to confirm the line number according to array element. If a line is empty, no text will be displayed

align → The weight ratio of the area occupied by each line of text, and the size of array elements must be the same size as the text array

callback → asynchronous callback on implementation result

Note: only available for desktop terminals of mini series with customer display function, with versions above v4.0.0. It won't be displayed if the text is too long.

5. Send single line text with custom size

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

Parameter:

string → text to be displayed

size → the font size of the text to be displayed, center by default

fill → whether to magnify the font to fill the display area

callback → implementation result of asynchronous callback

Note: only available for desktop terminals of mini series with customer display function, with versions above v4.0.0. It won't be displayed if the text is too long.

6. Send bitmap image

Function: void sendLCDBitmap(Bitmap bitmap, ILcdCallback callback)

Parameter:

bitmap → the image to be displayed

callback → asynchronous callback on implementation result

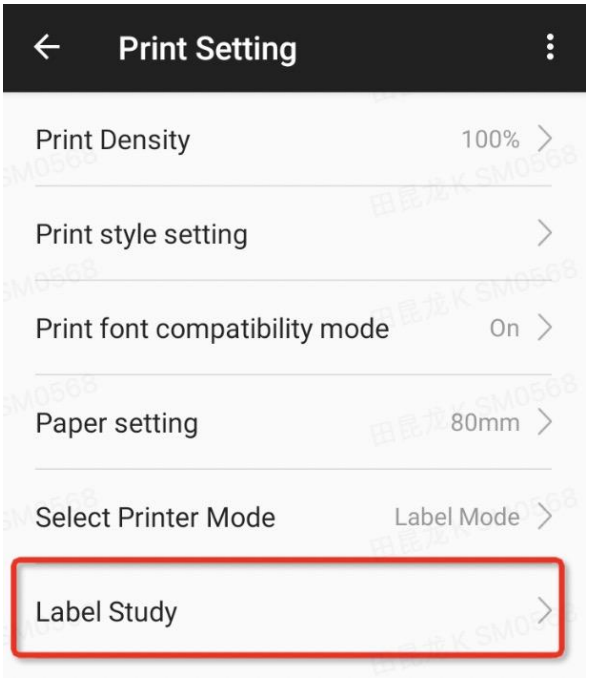
Note: only available for desktop terminals of mini series with customer display function, and the customer display can only display the image with the maximum pixel of 128*40.

1.2.16 Label printing instructions

Our mobile terminals of V series models now support label printing function, such as V2pro, V2s, etc. Please see the following instructions for detailed usage.

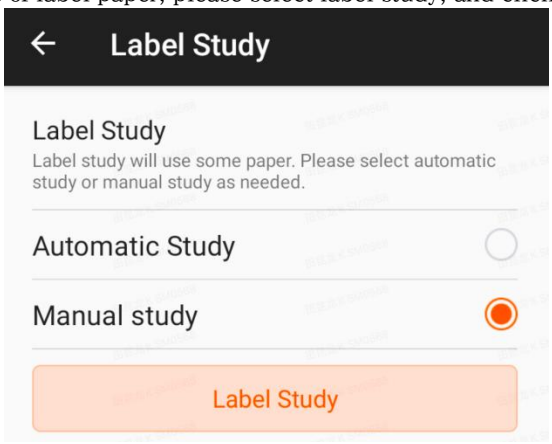
Preconditions:

1. Mode setting: the printer of SUNMI device prints thermal receipts by default. If you want to use the label printing function, please first set the print mode. Find inbuilt printing (or SUNMI printing) option, select the printer mode, then the label printing function is enabled. Please see the following figure: (if you can't find the option, please update the system to the latest version)



You can get the current printing mode by calling the interface getPrinterMode().

2. Label study: when you select thermal printing mode, you can see a newly-added option of label study. If you use it for the first time or change other types of label paper, please select label study, and click the option as follows:



After clicking the option, you will enable label study procedure. Please ensure there are at least three labels. After studying, a window will pop up to show you the studying result.
We recommend you to use the label paper with the specification of 50mm width, over 30mm height, and over 2mm label gap, to achieve the optimum positioning effect.

Interfaces for label printing function:

No.	Methods
1	void labelLocate() Locate the next label
2	void labelOutput() Output the label to the paper cutting position

Print label:
You can only implement the location output operation specific to the label by calling the label interfaces, while the label content needs to be customized by yourself according to your own needs, like the operation of thermal printing.

Please note that the height of printing content shall be within the label paper, or the printing content may exceed the label paper, which can result in printing to the next label or inaccurate location of the next label.
If you use a label paper with 30mm height, you can print the content with 240 pixels (30mm x 8 pixels), and the printer defaults to 32 pixels of line spacing, which allows you to print about 8 lines of text or a 384x240 image.
For example, if you want to implement a simple label as follows:

```
labelLocate()
//You need to locate the label position every time before sending the printing content

Set printing content
setPrinterStyle(WoyouConsts.ENABLE_BOLD, WoyouConsts.ENABLE)
//Set bold font
linewrap(1, null)
//Leave a blank line in the header
setAlignment(0, null)
//Content is aligned to the left
printText(“商品： 豆浆\n”, null)
//Print content; please feed a line (using \n)
printText(“到期时间： 12-13 14 时\n”, null)
//Print content; please feed a line (using \n)
printBarCode(“{C1234567890123456”, 8, 90, 2, 2, null)
//Print the barcode, which is of code128c type with a height of 90 occupying 154 pixels lines (a blank line will be reserved both for the upper and bottom lines for the barcode)
linewrap(1, null)
//Whether to leave blank according to the actual situation
```

Then a label will be output like the following figure:

After printing the content, you can choose to implement labelLocate and print the content circularly according to your own needs.
If you don’t need to print any more, you can implement labelOutput(), which can output the label to the paper cutting position, facilitating you to add other APIs according to needs to design your own label content.

Print multiple labels:

If you just need to print one label, you only need to implement the following operation:
labelLocate -> print label （-> labelOutput）
If you need to print multiple labels, you don’t need to implement labelOutput every time after sending label content, all you need to do is to implement like the following:
labelLocate->print label 1->labelLocate-> print label 2->labelLocate-> print label 3……
->labelLocate-> print label n （->labelOutput）

Note: currently, label printing can only be implemented by interface mode, and calling by instruction set is not supported.

1.3 Interface return description

Since instantiation errors often occur for AIDL callback interface, we use InnerResultCallback to replace the original callback interface specific to remote import library.

1.3.1 Interface methods description of InnerResultCallback

No.	Methods
1	void onRunResult(boolean isSuccess) Implementation results of commands
2	void onReturnString(String result) The return String on commands implementation result
3	void onRaiseException(int code, String msg) Return exceptions
4	void onPrinterResult(int code, String msg) Feedback specific to transaction printing

1. Implementation results of commands

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

Parameter:

isSuccess → implementation result:

true → implementation succeeded

false → implementation failed

Note: the interface return result means the implementation result of commands, not the result of printing.

2. Commands implementation result return String

Function: void onReturnString (String result)

Parameter:

result → implementation result

3. Return exceptions

Function: void onRaiseException (int code, String msg)

Parameter:

code → exception code

msg → exception description

Note: please see Exceptions List

4. Feedback specific to transaction printing

Function: void onPrintResult (int code, String msg)

Parameter:

code → status code:

0 → succeeded

1 → failed

msg → null if succeeded, and exception description if failed

Note: the return result of this interface means the result of commands processing and actual printing output. It may take some time, since you have to wait for the printer to move the paper.

1.3.2 Callback object example

```
new InnerResultCallback{
 @Override
 public void onRunResult(boolean b) throws RemoteException {
 //commands implementation result
 }
```

```
    @Override
    public void onReturnString(String s) throws RemoteException {
    //The return String on commands implementation result
    }
    @Override
    public void onRaiseException(int i, String s) throws RemoteException {
    //Exception return
    }
    @Override
    public void onPrintResult(int i, String s) throws RemoteException {
    //feedback specific to transaction printing
    }
}
```

1.3.3 Exceptions list

code	msg
- 1	"command is not support,index #"; // # represents No. # Byte is wrong
- 2	"# encoding is not support"; // # represents No. # Byte is 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 through inbuilt virtual Bluetooth

2.1 Introduction to virtual Bluetooth

You can see a paired Bluetooth "InnerPrinter" already exists in the Bluetooth list, which is a virtual printer device in the operating system, not actually exists. The virtual Bluetooth supports SUNMI <esc/pos> commands.
Some special commands are our custom commands, such as the following:

Function	Commands
Command of cash drawer opening	byte [5] : 0x10 0x14 0x00 0x00 0x00
Command of cutter fully cutting	byte [4] : 0x1d 0x56 0x42 0x00
Command of cutter cutting (leaving some space in the left without being cut)	byte [4] : 0x1d 0x56 0x41 0x00

2.2 Virtual Bluetooth use

1. Connect to the virtual Bluetooth;
2. Splice commands and text content into Bytes;
3. Send to InnerPrinter;
4. The underlying printing service drives the printer to print.

Note: **BluetoothUtil** is a Bluetooth tool, which can be used to connect virtual Bluetooth InnerPrinter.

2.2.1 Standard Bluetooth connection tool BluetoothUtil

```
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 Example of printing service via Bluetooth connection

```
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 an 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 Notices

Please add a Bluetooth permission statement in the App program to use the Bluetooth.

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

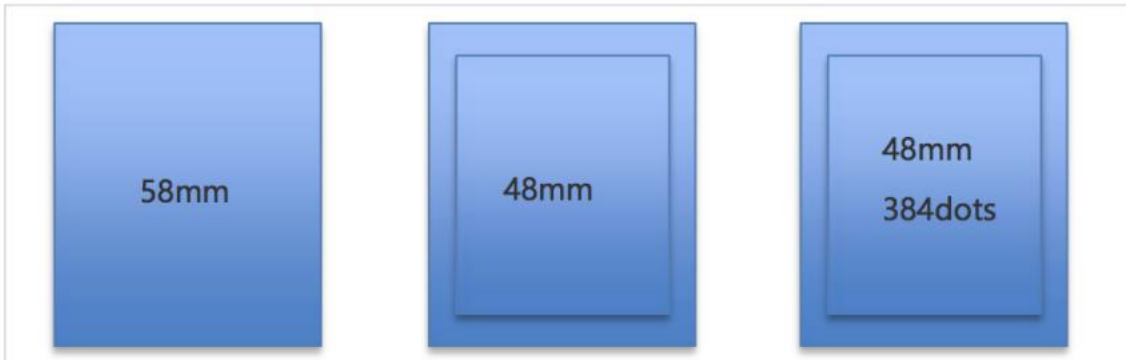
Appendix A: Printing service broadcast

Please build a broadcast receiver to monitor the broadcast when adopting the broadcast method.

Function	Action
Printer is under preparation	"woyou.aidlservice.jiuv5.INIT_ACTION"
Printer is updating	"woyou.aidlservice.jiuv5.FIRMWARE_UPDATING_ACITON"
Printing is ready	"woyou.aidlservice.jiuv5.NORMAL_ACTION"
Printing error	"woyou.aidlservice.jiuv5.ERROR_ACTION"
Out of paper	"woyou.aidlservice.jiuv5.OUT_OF_PAPER_ACTION"
Printhead is overheated	"woyou.aidlservice.jiuv5.OVER_HEATING_ACITON"
Printhead temperature back to normal	"woyou.aidlservice.jiuv5.NORMAL_HEATING_ACITON"
Cover open	"woyou.aidlservice.jiuv5.COVER_OPEN_ACTION"
Cover closing exception	"woyou.aidlservice.jiuv5.COVER_ERROR_ACTION"
Cutter exception 1 – cutter stuck	"woyou.aidlservice.jiuv5.KNIFE_ERROR_ACTION_1"
Cutter exception 2 – cutter back to normal	"woyou.aidlservice.jiuv5.KNIFE_ERROR_ACTION_2"
Printer firmware updating	"woyou.aidlservice.jiuv5.FIRMWARE_UPDATING_ACITON"
Printer firmware updating failed	"woyou.aidlservice.jiuv5.FIRMWARE_FAILURE_ACITON"
Printer not detected	"woyou.aidlservice.jiuv5.PRINTER_NON_EXISTENT_ACITON"
Black mark not detected	"woyou.aidlservice.jiuv5.BLACKLABEL_NON_EXISTENT_ACITON"

Appendix B: Printing service F&Q

1. Instructions of printing paper specification



Printing paper of 58mm width Effective print width of 48mm Effective print pixel of 384 pixels

Note: Our printer supports 58mm and 80mm printing papers. We take 58mm printing paper as an example in this documentation to explain the printer’ s supporting specs. The specs of 80mm printing paper are almost the same.

A 58mm printing paper is 58mm wide and its effective print width is 48mm or 384 dots per line. For V1 model, a paper roll as big as 40mm OD can be accommodated.

2. What's the resolution of SUNMI printer?

Our printer's resolution is 205DPI, with the calculation method as follows:
DPI=384dots/48mm=8dots/1mm=205dots/in=205

3. How to get the printer's status?

You can get the printer's status by calling the interface updatePrinterState. When the result returned is 505, it indicates that the printer is offline and can't be used.

4. Why there is no printout when I print an image?

Our printer will buffer content that is less than one line and print out when the line is full for all command interfaces except 2D barcodes. Therefore, when the image is not printed after calling printBitmap interface, it is likely that the width of the image cannot reach the width of the paper, so there is no output. You can print the cached image just by calling linewrap interface.
If you still can't print out the image after using the method above, please check the interface callback, to see whether there exist exceptions. Usually, the failure is caused by oversized images. Our printer can only support the image with the resolution of 2M (not the actual size of the image), and the largest displayable width depends on the paper specification. Please scale the image if you want to print the proper image.
In addition, if you send raster bitmap through Bluetooth hexadecimal command, which leads to the failure of image printing, then you need to check whether there is any error in the command. In this case, the error of one byte data may affect the realization of the whole data.

5. The content of my barcode is too long, and can't be displayed in one receipt. Which barcode shall I choose?

Due to the width limitation of the printing paper (generally 384 pixels width for handheld terminals and 576 pixels width for desktop terminals), the barcode with too many printing digits will not be displayed completely. At this time, it is best to adjust the width of the barcode to see whether the problem can be solved.
If the content of barcode is too much, exceeding 20 digits, then code128 code type is generally used in this case. Please call printBarCode interface to select code128 code type. At present, we have realized mixed coding in this interface, which means you can add {A, {B, {C to indicate dynamic switching of coding type (A: numbers, uppercase letters, and control characters; B: numbers, letters, and characters; C: double digits). If it is a number, change the type to C. If it is other characters, you can change the type to A or B, then the barcode can be printed. For example, if you want to print ABab1212, you can input "{BABab{C1212".
Mixed coding can be supported by printing service with versions above v4.0.0. If your printing service can't support it or you use Bluetooth printing, implement the following method to get a group of epson commands, so as to send the returned result through sendRawData interface or Bluetooth.
Among which the data is the barcode content to be printed. Since the default minimum width is 2 pixels, so when the width is smaller than it, the scan speed will decrease seriously. If you want to print the content of barcode exceeding 25 pixels or more, you can choose to set the pixels to 1.

```
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);
}
pos++;
pos++;
}
}
return temp.toByteArray();
```

}

6. Why can’t I receive the callback result of the printing interface?

Firstly, if you use AIDL to access the interface, you need to pay attention to use the matched AIDL resource.

AIDL resource for P series, V1s, and V2； AIDL resource for T series and S series; AIDL resource for Mini series; AIDL resource for V series.

After making sure the resource file is correct, please note that when creating the callback object, you need to use the callback of internal static class Stub generated by AIDL.
~~new ICallback(…)~~ —> new ICallback.Stub()

If you use the remote import library, you just need to implement the encapsulated InnerPrinterCallback to avoid result loss.

7. Select and set character set

Background: Since the inbuilt printer is compatible with binary byte stream transmission, the printed text content sent through bytecode involves character set selection and setting; The default setting in the device is multi-byte and GB18030 character encoding type.
The inbuilt printer supports the following single-byte encoding types:
[Parameter] [Code] [Country]
0 "CP437"; USA and Europe standard
2 "CP850"; multi-language
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 Europe
19 "CP858";
21 "CP874";
33 "Windows-775"; Balto-Slavic
34 "CP855"; Cyrillic
36 "CP862"; Hebrew
37 "CP864";
254 "CP855"; Cyrillic

The inbuilt printer supports the following multi-byte encoding types:
[Parameter] [Code]
0x00 || 0x48 "GB18030";
0x01 || 0x49 "BIG5";
0xFF “utf-8”;

For different countries, you can set the printer’s recognizable printing content data stream according to the country’s language or your own needs. For example, to print the content of CP437 page table, you need to send:
0x1C 0x2E - - Set to single-byte encoding type
0x1B 0x74 0x00 - - Set to CP437 code page table in the single-byte code page

To print the content of CP866 page table, you need to send:
0x1C 0x2E - - Set to single-byte encoding type

0x1B 0x74 0x11 - - Set to CP866 code page table in the single-byte code page

To print traditional content, you need to send:
0x1C 0x26 - - Set to multi-byte encoding type
0x1C 0x43 0x01- - Set to BIG5 code page table in the multi-byte code page

To print UTF-8 code content (if you use utf-8 code, which supports all Unicode character set, you can print all the contents), you need to send:
0x1C 0x26 - - Set to multi-byte encoding type
0x1C 0x43 0xFF- - Set to UTF8 code in the multi-byte code page

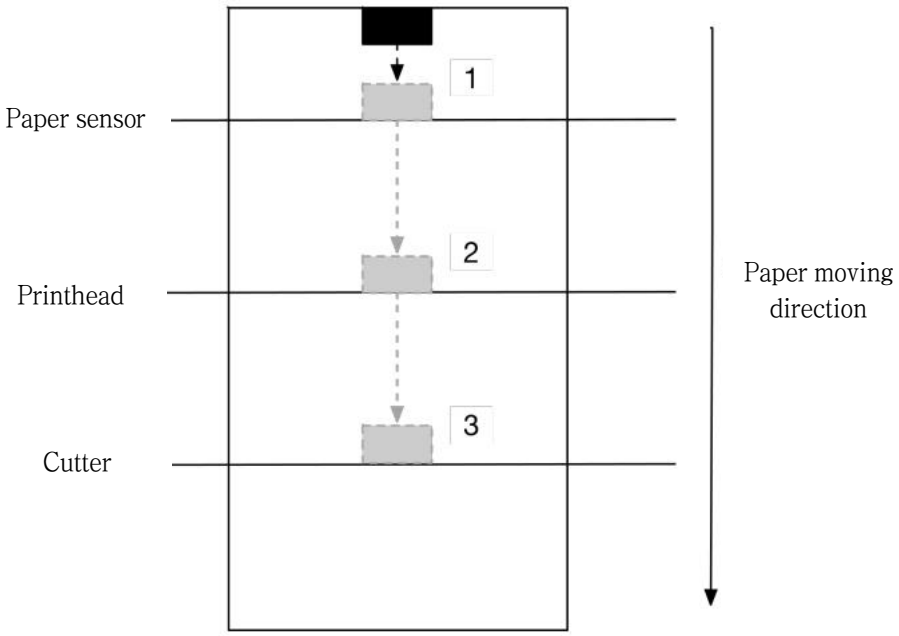
8. Black mark printing instructions

SUNMI T1 model supports printing with black mark printing paper in conformity with the specification.

Requirement on black mark printing paper: different from general black mark printer, SUNMI T1 hardware is not equipped with sensor detecting the black mark. However, it has the function similar to black mark printing, that is, it adopts the principle that the paper sensor regards the materials with reflectivity not more than 6% (infrared waves within 700 – 1000mm) as paperless. Due to different principles, you can't precisely locate the black mark position by using SUNMI T1 black mark printing mode as that of using black mark printer, which may lead to a certain error.

Requirement on the black mark position: the black mark needs to be in the center horizontally on the paper for the paper sensor to detect it.

The work principle of SUNMI T1 black mark:
The paper sensor is not on the same level as the position of the printhead and cutter, and the black mark on the paper will first pass through the paper sensor (as shown on Figure 1), then pass through the printhead (as shown on Figure 2), and finally reach the position of the cutter (as shown on Figure 3) to cut the paper.



Under the black mark mode, after the paper sensor detects that there's no paper, the printer will move the paper automatically for 7mm, and if the paper sensor still can't detect any paper after paper moving, the printer will implement it as no paper. If paper is detected after paper moving, the printer will implement it by black mark mode.

When the printing content covers the position of black mark, the printing will continue, that is, when the black mark comes to the position 2 in the figure, there is still a print job, and the print job will cover the black mark and continue until the printing is completed.

According to this mechanism, the last printing position should be kept a certain distance from the position of paper sensor (the distance between the last printing position and the edge of the black mark should be 6mm), to ensure that the printed content is within the range of two black marks.

Command sequence of black mark mode:

1. Send printing content;
2. Input command of paper moving to black mark {0x1c, 0x28, 0x4c, 0x02, 0x00, 0x42, 0x31};
3. Input cutter command {0x1d, 0x56, 0x00}.

After completing printing content, the printer will automatically detect the next black mark, and cut in the specified position.

You can modify the black mark mode and cutter position in Setting -> Printing -> Inbuilt printing management.

9. How to print special symbols?

There are several methods to print some special symbols such as the currency symbols ($ ¥ € £ ₣):

The simplest method is to directly call the printing interface printText provided by us to print.

printText("$ ¥ € £ ₣\n”, null)

If you print the content through Bluetooth or using esc commands, we recommend to use our specified utf-8 characters (GB18030 by default in the device), since different currency symbols correspond to different page numbers in different countries, which may lead to different character sets for printing special symbols. For character set setting, please see FAQ7.

Take printing currency symbols ($ ¥ € £ ₣) as an example:

You need to send command byte[] data = new byte[]{0x1C, 0x26, 0x1C, 0x43, 0xFF}, to achieve that after the printer receives utf-8 character code, it will send the corresponding character data “$¥€£₣”.getBytes(“utf-8”).

The commands are as follows:

sendData(new byte[]{0x1C, 0x26, 0x1C, 0x43, 0xFF}, null)
sendData(“$¥€ £F” .getBytes(“utf-8”))