SUNMI Printer Developer Documentation

Document Revisions

Version	Undate Date	Updates	Owner
1.0.0		Original version	徐赟庭
1.1.170301	2017/03/01	Addition: description on the spec of images to be printed Addition: description on whether there is a hardware query interface	徐赟庭
1.1.170315	2017/03/15	Addition: AIDL cutter interface Addition: AIDL interface to open the cash drawer Addition: AIDL interface to obtain the number of times the cutter has been used Addition: AIDL interface to obtain the number of times the cash drawer has been opened	徐赟庭
1.1.170322	2017/03/22	Addition: AIDL transaction print interface with feedback Addition: result feedback on callback transaction print Modification: description on barcode format	徐赟庭
1.1.170329	2017/03/29	Addition: description of character sets change	徐赟庭
1.1.170615	2017/06/15	Addition: description of transaction printing	徐赟庭
1.1.170726	2017/07/26	Modification: description of AIDL calling	徐赟庭
1.1.170802	2017/08/02	Addition: description of T1 black mark print	徐赟庭
1.1.170803	2017/08/03	Addition: description of T1black mark instructions	徐赟庭
1.1.180523	2018/05/23	Modification: new document formate; standardization	Darren
1.1.180601	2018/06/01	Addition: extension interface for image printing; instance description of ICallback	Darren
1.1.180612	2018/06/12	Modification: bold instruction and some descriptions.	Darren
1.1.180629	2018/06/29	Modification: ICallback interface method errors and descriptions	Darren

1.1.180912	2018/09/12	Modification: rearrange the document	田昆龙
1.1.190225	2019/02/25	Modification: FAQ	田昆龙
1.1.191008	2019/10/08	Modification: FAQ	田昆龙
1.1.191112	2019/11/12	Modification: update resource links and the introduction	田昆龙
1.1.191211	2019/12/11	Addition: introduction to label printing function	田坤龙

Introduction

Some SUNMI devices come with an inbuilt thermal printer (with a buffer), which allow Apps to directly print thermal receipts through SDK. The SUNMI devices which are equipped with a printer are:

Mobile products – V1, V1s, V2 Pro, etc.
Smart payment products – P1, P1 4g, etc.
Desktop POS – T1, T2, T1 MINI, T2 MINI, etc.
POS scale – S2, etc.

There are two specs of inbuilt printers for SUNMI devices:

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

An App developer can call the inbuilt printer with 3 methods:

- **Call the printer via the inbuilt print service interface** – this method works best for the developers who are not familiar with Epson command and develop Apps related to printing for the first time. Developers can realize the desired printing effects through multiple printing interfaces provided by SUNMI print service;
- **Call the printer via the inbuilt virtual Bluetooth device –** this method works best for developers who have developed Bluetooth or USB printers before, or the App being developed has already realized Bluetooth printing. Developers can change some codes and realize the desired printing effects.
- **Call the printer via JS bridge on H5 Web page** – this method works best for the app accessing to H5 webpage.

1.Call a Printer via the Inbuilt Print Service Interface

1.1 . Connect to the Inbuilt Print Service

1.1.1.Via Remote Dependency

For developers using gradle in configuration, we recommend to use remote dependency library in order to call the inbuilt print service conveniently. Meanwhile, our library has adapted itself to different device models, and interfaces for different models can be differentiated automatedly when using remote dependency. There is no need to configure different AIDL files to be in accordance with different device models, and reminders will be prompted if the developer uses a wrong interface.

● **Integration**

Add dependency and attribute configuration to the build.gradle under the directory of the App:

```
dependencies{
        implementation ‘com.sunmi:printerlibrary:1.0.13’
}
```

● **Initialization**

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

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

InnerPrinterCallback innerPrinterCallback = new InnerPrinterCallback(){

        @Override
        protected void onConnected(SunmiPrinterService service){
                //Here is the remote service interface handle after the binding service has been successfully connected
                //Supported print methods can be called through service
        }

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

result: to show the binding result (succeeded or failed)

● A Simple Case to Call Print

```
        try{
                sunmiPrinterService.printText(“content to be printed\n”,

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

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

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

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

● Disconnection

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

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

● Class Specifications

InnerPrinterManager

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

InnerPrinterCallback

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

SunmiPrinterService

SunmiPrinterService is the SUNMI print service interface class, which defines all printing methods. The instance of this type can be obtained through connecting to a service. It is the same as the AIDL IWoyouService interface.

InnerPrinterException

InnerPrinterException is the printing exception class. An error or exception may occur when calling a method because some devices are not equipped with some functions (for example, a handheld device is not equipped with a cash drawer, printing error will pop out when calling the interface of a cash drawer).

InnerResultCallbcak

InnerPrinterException is the callback interface of printing method, which is used to obtain the execution result of an interface method.

InnerLcdCallback

InnerLcdCallback is a callback interface of customer display method, which is used to obtain the implementation result of customer display method.

InnerTaxCallback

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

1.1.2.Via AIDL

Intro to AIDL

AIDL (Android Interface Definition language) is a description language of an Andriod inner progress communication interface, which can be used to define the commnucation interface in progress. SUNMI AIDL supplies capsulated common printing cammands to facilitate developers' quick access to SUNMI printers.

How to Use AIDL

Follow the following 5 steps to establish a connection:

1. Add the AIDL file in the AIDL resource document to the project, and please be aware that the package path and package name shall not be changed (java files might be included in some versions of devices);
2. Realize ServiceConnection in the code class for print control;
3. Call ApplicationContext.bindService(), and pass the interface object of _x0005_AIDL during the realization of ServiceConnection. Please note: bindservice is not a blocking call, which means that the binding has not been completed after calling, and the binding result can only be confirmed by the callback of onServiceConnected.
4. During the realization of ServiceConnection.onServiceConnected(), you will receive an Ibinder instance (the Service called). Transfer the parameter into IwoyouService type by calling **IWoyouService.Stub.asInterface(service).**
5. Now you can call all kinds of methods defined in the IwoyouService interface to print.

AIDL Resources Download

⚠Please note: Please use AIDL resources according to device models, for the interfaces of different devices may vary. There are four resource packages currently:

old (V1) for the first SUNMI V1 model
——Mainly includes:

IWoyouService.aidl
ICallback.aidl
TransBean.aidl
TransBean.java

handheld (handheld devices except V1) for V1s, V2, P2, etc.
—— Mainly includes:

IWoyouService.aidl
ICallback.aidl
TransBean.aidl
TransBean.java
ITax.aidl

desktop (desktop devices) for SUNMI T1, **T2,** etc.

——Mainly includes:

IWoyouService.aidl

ICallback.aidl

ITax.aidl

desktop + lcd (desktop + customer display) for SUNMI desktop devices with a customer display like **T1mini**, **T2mini.**

——Mainly includes:

IWoyouService.aidl

ICallback.aidl

ITax.aidl

ILcdCallback.aidl

General resource

WoyouConsts.java

An Example on Binding a Service

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);
}
```

Bind print 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 . Description of Interface Definition

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

1.2.1. Printer Initialization and Setting

No.	Method
1	void **printerInit( )** Printer initialization
2	void **printerSelfChecking(ICallback** callback**)** Printer self-inspection

1.Printer Initialization

Function: void printerInit( )

Note: reset the logic programs (type setting, bold, etc.) of a printer but not to clear the data in the buffer. Therefore, uncompleted print jobs will be continued after resetting.

2.Printer Self-Inspection

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 SN of a printer board
2	String **getPrinterModal( )** Get printer type interface
3	String **getPrinterVersion( )** Get printer firmware version
4	Build**.MODEL** (constant) Device name
5	int **updatePrinterState()** Get the latest status of a printer
6	String **getServiceVersion()** Get the version number of a print service

7	int **getPrintedLength(ICallback** callback**)** Get the print length of a printhead
8	int **getPrinterPaper()** Get the current paper spec of a printer

1. Get the Latest Status of a Printer

Function: String **updatePrinterState ()**

Return value:

1	The printer works normally
2	Preparing printer
3	Abnormal communication
4	Out of paper
5	Overheated
6	Open the lid
7	The paper cutter is abnormal
8	The paper cutter has been recovered
9	No black mark has been detected
505	No printer has been detected
507	Failed to upgrade the printer firmware

Note: this interface is not available to V1 for now. aside from getting the status actively, you can also obtain them asynchronously by registering broadcast. Please refer to Appendix A.

2.Get the Length the Printer Has Printed

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

Description: currently, the print length of a printer (since power on) can be obtained. For a desktop device and a handheld device, the ways to get the print length of a printer may vary due to the hardware difference, i.e. the print length can be obtained through the ICallback callback interface for a handheld device, and the print length can be obtained through the return value for a desktop device.

3.Get the Paper Spec of a Printer

Function: int **getPrinterPaper()**

Description: by default, a handheld device uses 58mm printing paper and a desktop device uses 80mm printing paper. However, adding a partition and adjusting some configurations can be used to enable a desktop device uses 58mm printing paper. This interface will return the printing paper spec that is currently being used by the printer.

Note: currently, desktop devices that support this interface are T1 (v2.4.0 and above), T2 (v1.0.5 and above) and S2 (v1.0.5 and above); other devices which with a version 4.1.2 and above also can use this interface to get the printing paper spec.

1.2.3. ESC/POS Instruction

编号 No.	方　法 Method
1	void **sendRAWData** **(**byte[] data, **ICallback** callback**)** 打印 ESC/POS 格式指令 Print ESC/POS format instruction

1.Print ESC/POS Format Instruction

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

data	ESC/POS instruction.
callback	result callback.

Note: for relevant commands, please refer to ***ESC/POS instruction set***.
Example:

```
    woyouService.sendRAWData(new
byte[]{0x1B,0x45,0x01}, callback);
   // 1B,45, 01 are instructions to realize bold font
```

1.2.4. Description of the Interface for Print Style Setting

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

1. Set Printer Style

Function: void **setPrinterStyle(**int key, int value**)**
Parameter: Refer to WoyouConsts.java printer style set constant class
key　　Set attributes which are usually divided into **ENABLE_XXX** and **SET_XXX** through the definition in constant type interface.
value　　corresponds to the attribute setting includes status or size
ENABLE or DISABLE should be chosen for the Enable attribute ENABLE_XXX.
The specific size should be set for the SET attribute SET_XXX.

Example:

```
  woyouService.setPrinterStyle( WoyouConsts.ENABLE
_ILALIC,WoyouConsts.ENABLE);
   //to print italic text for the content printed subsequently
  woyouService.setPrinterStyle(WoyouConsts.SET_LINE
_SPACING,123);
  // to realize the leading (123 pixels) for the content
printed subsequently
```

Note: this interface is only available to the print service (v4.2.22 and above)!

1.2.5. Black Mark Print

No.	Method/Instruction
1	int **getPrinterMode**() Get the printer mode
2	int **getPrinterBBMDistance**() Paper feed distance
3	For relevant mode settings, please complete in "Setting" "Print Setting".

1.Get Printer Mode

Function: int **getPrinterMode**()

Return values:

0	Normal mode;
1	Black mark mode;

Note: only available to T1 and T2.

2.Get the Paper Feed Distance of a Printer in Black Mark Mode

Function: int **getPrinterBBMDistance**()

Return value: paper feed distance (pixel line).

Note: only available to T1 and T2.

1.2.6.Character Printing

No.	Method/Instruction
1	void **setAlignment**(int alignment, **ICallback** callback**)** Set alignment
2	~~void **setFontName**(String typeface, **ICallback** callback)~~ ~~Set print typeface (unavailable for now)~~

3	void **setFontSize**(float fontsize, **ICallback** callback) Set font size
4	esc/pos instruction: bold font {0x1B, 0x45, 0x1}, cancel bold font {0x1B, 0x45, 0x0} Set and cancel the bold font effect
5	void **printText**(String text, **ICallback** callback) Print text
6	void **printTextWithFont**(String text, String typeface, float fontsize, **ICallback** callback) Print text in a specified typeface and size
7	void **printOriginalText** **(**String text, **ICallback** callback**)** Print vector font

1.Set Alignment

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

Parameter:

alignment alignment: 0 align left, 1 center, 2 align right.

callback Result callback.

Note: it is a global method which will affect the subsequent printing. If a printer initialization method has been called, this method will be restored to its default setting.

Example:

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

2.Set Typeface (Unavailable for Now)

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

Parameters:

typeface font name. Currently the font **"gh"** is available (a Chinese monospaced font).

callback Result callback.

Note: it is a global method which will affect the subsequent printing. If a printer initialization method has been called, this method will be restored to its default setting (setting fonts is unavailable to existing versions).

Example:

```
woyouService.setFontName("gh", callback);
```

3.Set Font Size

函数：void **setFontSize (**float fontsize, **ICallback** callback**)**

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

Parameters:

fontsize font size.

callback Result callback.

Note: it is a global method which will affect the subsequent printing. The settings will be canceled if an initialization has been called. Setting font size is not included in the print method in the standard international instruction, and it will affect character width and the number of characters in a line correspondingly. As a result of this, the type setting based on monospace font will be changed.

Example:

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

4.Set and Cancel Bold

Instruction: make fonts bold {0x1B, 0x45, 0x1}, cancel bold {0x1B, 0x45, 0x0}

Note: please refer to the **1.1.3.3. ESC/POS instruction** in this document.

Example:

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

5. Print Text

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

Parameters:

text the text to be printed. The part of the text **which exceeds a line will be warped onto the next line.** The **force line break"\n"** must be added to the end of a line of content which not fully occupy the line in order to print this line of content, otherwise it will be cached in the buffer.

callback Result callback.

Note: if a style (alignment, font size, bold, etc.) of the text to be printed needs to be changed, please set before calling **printText** method.

Example:

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

6. Print Text in a Specialized Font and Size

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

Parameters

text the text to be printed. The part of the text which **exceeds a line will be wrapped onto the next line.** The **force line breaker "\n"** must be added to the end of a line of content which not fully occupy a line in order to print this line of text, otherwise it will be cached in the buffer.

typeface font name (setting font is not available to the existing version. Default).

fontsize font size. It takes effect only to this method.

callback Result callback.

Note: font setting is only valid for this time.

Example:

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

7.Print Vector Font

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

Parameters:

text the text to be printed. The part of the text which exceeds a line will be wrapped onto the next line. The force line breaker "\n" must be added to the end of a line of content which not fully occupy a line in order to print this line of text, otherwise the line of content will be cached in the buffer.

callback Result callback.

Return value:

Note: output characters are **in the same width of vector fonts**, which means that **they are not monospace.**

Example:

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

1.2.7. Print Tables

No.	Method/Instruction
1	void **printColumnsText(**String[] colsTextArr, int[] colsWidthArr, int[] colsAlign, **ICallback** callback) Print a row of a table (unavailable to Arabic characters)
2	void **printColumnsString(**String[] colsTextArr, int[] colsWidthArr, int[] colsAlign, **ICallback** callback) Print a row of a table. Colunmn width and alignment can be specified.

1.Print a Row of a Table (Unavailable to Arabic Characters)

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

Parameters:

colsTextArr each column's array of text string.

colsWidthArr each column's width array, which is calculated based on English characters. Each Chinese character takes the space of two English characters, and the width exceeds 0.

colsAlign each column's alignment: 0 align left, 1 center, 2 align right.

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 wrapped onto the next line. This is unavailable to Arabic characters.

Example:

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

2.Print a Row of a Table With Specified Column Width and Alignment

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

Parameters:

colsTextArr each column's array of text string.

colsWidthArr the weight of each column's width, i.e. the width proportion of each column.

colsAlign each column's alignment: 0 align left, 1 center, 2 align right.

callback Result callback.

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

Example:

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

1.2.8. Print Images

编号 **No.**	方 法 **Method**
1	void **printBitmap(**Bitmap bitmap, **ICallback** callback**)** Print image
2	void **printBitmapCustom(**Bitmap bitmap, int type, **ICallback** callback**)** Print image (2)

1. Print Image

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

Parameters:

bitmap	image Bitmap object.
callback	Result callback.

Note: the resolution of an image should be within 2M, and the width should be set in accordance with the paper spec (58: 384 Pixel, 80: 576 Pixel). It cannot be shown if it exceeds the paper width.

Example:

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

2. Print Image (2)

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

Parameters:

bitmap picture bitmap object (the maximum width is 384 Pixel, and a picture exceeding 1M cannot be printed).

type currently, two print methods are available:

0	the same method of **printBitmap**();
1	black and white picture (with the threshold value of 200)
2	grayscale picture

callback Result callback.

Note: the resolution of an image should with within 2M, and the width should be set in accordance with the paper spec (58mm: 384 Pixel, 80mm: 576 Pixel). It cannot be shown if it exceeds the paper width.

Available to: P1 (v3.2.0 and above), P14g (v1.2.0 and above), V1s (v3.2.0 and above), V2 (v1.0.0 and above), T1 (v2.4.0 and above), T2 (v1.0.5 and above), S2 (v1.0.5 and above), T1mini (v2.4.1 and above), T2mini (v1.0.0 and above).

Example:

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

1.2.9. Print 1D/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 barcodes
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**)**

Parameters:

data the 1D barcode to be printed

symbology barcode type (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 barcode height (ranges from 1-255. 162 by default).

width barcode width (ranges from 2-6. 2 by default).

textPosition text position (0-3)

0	don’t print text
1	text above the barcode
2	text under the barcode
3	text above and under the barcode

callback Result callback

Note: the differences caused by different codes are listed below

Code	Description
code39	Numbers within 13 digits can be printed
code93	Numbers within 17 digits can be printed

ean8	8-digit numbers (the last digit is for parity check). The effective length is 8 digits (numbers).
ean13	The effective length is 13 digits, and the last digit is for parity check.
ITF	Number (even number of digits) within 14 digits is required.
Codabar	Numbers within 0-9 plus 6 special characters. Maximum 18 digits can be printed.
UPC-E	8-digit number (the last digit is for parity check)
UPC-A	12-digit number (the last digit is for parity check)
code128	Code128 can be devided into subset A, B and C: 128A: numbers, upper-case letters, and control characters, etc. 128B: numbers, upper- and lower-case letters and special character. 128C: numbers only. It must have an even number of digits (if it has an odd number of digits, the last digit will be omitted). By default, the interface uses subset B. “{A” or “{C” should be added before the content if you need to use subset A or C, for example: “{A2344A”, ”{C123123”, ”{A1A{B13B{C12”.

Example:

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

2. Print QR Barcodes

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

Parameters:

data QR code to be printed.

modulesize the size of the QR code block. Unit: dot, which should be within 4-16.

errorlevel QR code 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 callback.

Note: after calling this method, the content will be printed under normal print status, and every QR code block is 4 Pixel (when smaller than 4 Pixel, the code parsing might fail). version19 (93*93) is a maximum mode supported.

Example:

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

3. Print 2D barcodes

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

Parameters:

data the 2D barcode to be printed.

symbology the type of 2D barcode

1 Qr (same as the **printQRCode** interface)
2 PDF417
3 DataMatrix

modulesize the effective code block size, and it varies in accordance with different types of 2D barcode

Qr code	4 ~ 16 (same as the **printQRCode** interface)
PDF417	1 ~ 4
DataMatrix	4 ~ 16

errorlevel 2D barcode error correction level, and it varies in accordance with different types of 2D barcodes

Qr code	0 ~ 3 (same as the **printQRCode** interface)
PDF417	0 ~ 8

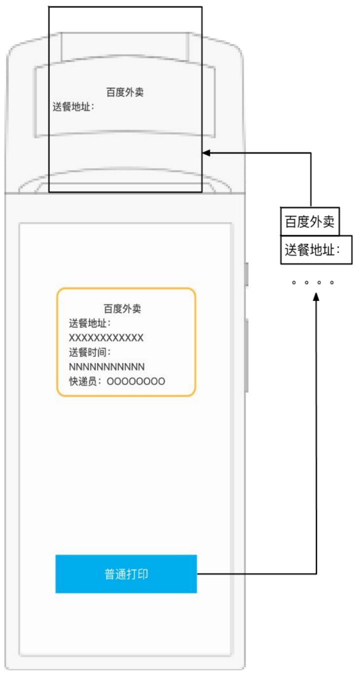
DataMatrix ECC200 error correction is used by default, and adjustment on setting is unavailable.

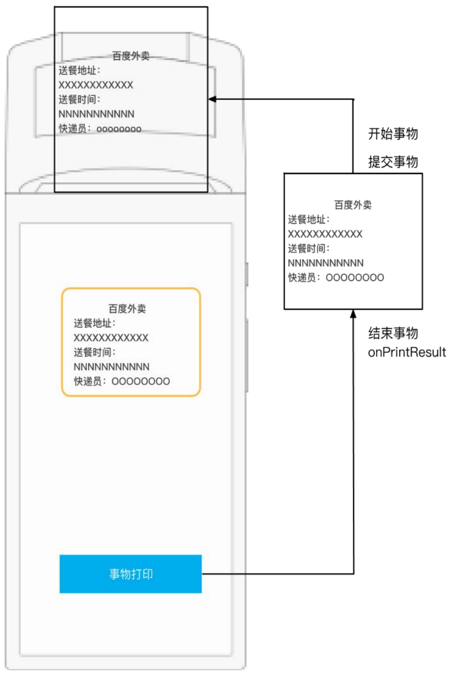
callback Result callback.

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

1.2.10. Transaction printing

The mode transaction printing caters to the needs to control content to be printed as well as get feedback on printing results. It is similar to establish a buffer for a transaction queue, when a developer uses this mode, a transaction list will be opened and print methods can be added. The printer will not print the content instantly, it will print by the order of transactions after the transactions being submitted. After printing, the feedback





on printing results will be obtained.

Notes on transaction printing:

1. when printing the queue of transactions, the printing results will be fed back after printing. However, **all print jobs submitted this time will loss if errors including out of paper, overheated occured, and an error feedback will be provided.** Which means, if the printer error occurred before or during a print job, the job won't be printed;

2. When alternately using (buffer) transaction printing and instruction printing, if a printer error occurred, the content of instruction printing won't be cleared!

3. When using transaction printing mode, other interfaces in **1.2.x.** can be used to output content, but the content will not be printed instantly but saved in the buffer, and it will be printed after methods **exitPrinterBuffer()** or **commitPrinterBuffer()** being called.

4. The result of a transaction printing will be returned in the **onPrintResult** (int code, String msg) method of **ICallback interface** (it may **take some time,** for you have to wait for **the paper to be printed out**. Transaction printing is **not recommended to repeatedly print a single line,** for it will slow down the print speed. Transaction printing is suitable to print a **whole receipt**). The corresponding return codes are:

a) 0 printed successfully, and the msg is "Transaction print successful!";

b) 1 failed to print, and the msg is "Transaction print failed!".

5. Below is a complete pseudocode example on transaction printing:

```
enterPrinterBuffer(true) ——enter into the transaction mode, the subsequent orders won't be outputted instantly
printText(/*something*/)
printBitmap(/*bitmap resource*/)
…… other print-relevant methods——print some content
commitPrinterBuffer()/commitPrinterBufferWithCallback(callback)——submit a
 transaction, and the printer will start printing. The result (succeeded or failed)
 will be returned in callback.
……Wait for the return of the last transaction
printText(/*something*/)
printBitmap(/*bitmap resource*/)
…… other print-relevant methods——you can choose to wait for the result of the last transaction printing or proceed printing
commitPrinterBuffer()/commitPrinterBufferWithCallback(callback)——continue to submit the next transaction, and the printer will continue to print at this time
exitPrinterBuffer(true)/exitPrinterBufferWithCallback(true, callback)——call it
 when you need to exit transaction mode, and printing will be continued if new
 data has been entered after the last submission, otherwise it won't continue
 printing.
```

2.

6. Detailed methods:

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

3.

1. The lib package transaction printing interface

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

Parameters:

tranBean	Print job list.
callback	Result callback.

Example:

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

2. Enetr into the transaction printing mode

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

Parameters:

clear whether to clear the cache in the buffer:

true clear the unsubmitted content of the last transaction printing;

false not to clear the unsubmitted content of the last transaction printing, and the content will be included in the submission next time.

Note: enable and enter into the transaction printing mode. In this mode, content will not be printed instantly until the submission of a transaction or exiting the submission of a transaction.

Version supported: except V1 device.

Example:

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

3. Exit the transation printing mode

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

Parameters:

commit whether to print the content in the buffer:

true print all content in transaction queue

false not to print the content in transaction queue, and the content will be saved and submitted next time.

Versions supported: except V1 device.

Example:

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

4. Exit the transaction printing mode and callback the result

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

Parameters:

commit whether to print the content in the buffer:

true print all content in the buffer

false not to print the content in the queue of transacion and save it to submit next time

callback Result callback

Versions supported: except V1 device.

Example

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

5. Submit transaction printing

Function: void **commitPrinterBuffer()**

Note: submit and print all content of a transaction queue, and the transaction mode will be remained after the submission and printing.

Versions supported: except V1 device.

Example:

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

6. Submit Transaction Printing and Callback the Result

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

Parameter:

callback Result callback

Versions supported: except V1 device.

Example:

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

1.2.11. Line feed

No.	Method
1	void **lineWrap(**int n, **ICallback** callback**)** Implement n LFs on the paper

1. Implement n LFs on the paper

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

Parameters:

n	The number of LFs on the paper
callback	Result callback.

Note: line feed by force. Implement **n** LFs on the paper after the content has been printed.

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 filled up by calling the interface to make sure the printed content won't be cut.

It is only available to 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 value: the times the cutter has been used.
Note: it is only available to the desktop devices with a cutter function.

1.2.13. Cash Drawer

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

1. Open a Cash Drawer

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

callback	Result callback.

Note: it is only available to the desktop devices with a cash drawer function.

Example:

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

2. Get the Number of Times a Cash Drawer Has Been Opened

Function: int **getCutPaperTimes ()**

Return value: the times a cash drawer has been opened.

Note: it is only available to the desktop devices with a cash drawer function.

3. Get the Current Status of a Cash Drawer

Function: int **getDrawerStatus()**

Description: this interface can be used to get the status of a cash drawer which is connected to a device (available to devices that can be connected to a cash drawer)

Note: currently, it is available to S2, T2 and T2 mini with v4.0.0 and above.

1.2.14. Get the Attributes of Global Settings

You can override the style set by a developer through some print style setting and configuration. The interfaces below 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 anti-white style enable of global fonts
3	boolean **isForcedBold()** Get the bold style enable of global fonts
4	boolean **isForcedUnderline()** Get the underline style enable 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 printer density

Note: the interfaces above (except the interface to get the printer density) are available to V1, V1s, P1 with a version v3.2.0 and above and P14g with a version v1.2.0 and above.

1.2.15. Description of the Customer Display Interface

This is available to the devices of mini series. Please see the table below for instruction:

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 content size of each line will be adjusted according to the 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 bitmap and show it on the customer screen

1. Send Customer Display Control Command

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

Parameter: flag 1 initialization 2 awake LCD 3 hibernate LCD 4 clear screen

Note: only available to the desktop devices of mini series which have a customer display.

2. Send a Single-Line Text

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

Parameters:

string	the text displayed
callback	the asynchronous callback result

Note: only available to the desktop devices of mini series which have a customer display. The text cannot be displayed if it is too long.

3. Send a Double-Line Text

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

Parameters

topText	display the top half text
bottomText	display the bottom half text
callback	the asynchronous callback result

Note: only available to the desktop devices of mini series which have a customer display. The text cannot be displayed if it is too long.

4. Send a Multi-Line Text

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

Parameters:

text display the array of each line, and determine the number of lines based on the array. No content will be displayed if the line is a blank.

align The weight ratio of each line takes in the area. The element size of this array must be consistent with the text array.

callback the asynchronous callback result

Note: only available to the desktop devices (v4.0.0 and above) of mini series which have a customer display. The 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)

Parameters

string	the text to be displayed
size	the font size of the text. The font is centered by default
fill	whether to stretch the text to fully fill the display area
callback	the asynchronous callback result

Note: only available to the desktop devices (v4.0.0 and above) of mini series which have a customer display. The text cannot be displayed if it is too long.

6. Send a bitmap and show it on the customer screen

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

Parameters:

bitmap	the picture to be displayed
callback	the asynchronous callback result

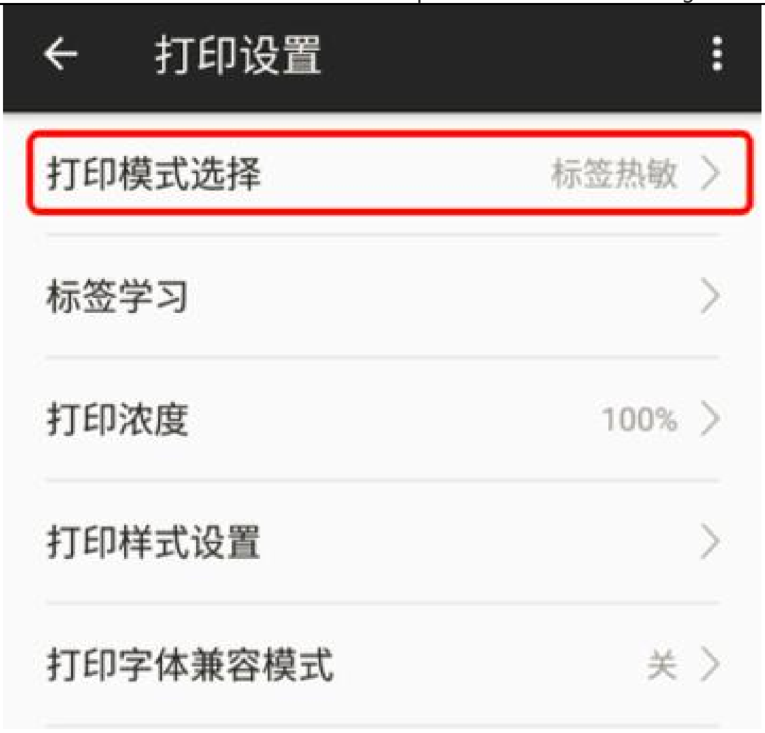
Note: only available to the desktop devices of mini series with a customer display. The picture with a maximum pixel128*40 can be shown on a customer display.

1.2.16. Introduction to label printing

Now, SUNMI V2 and V2 Pro support label printing functions. Label paper should be 50mm wide and 30mm high and a gap between labels should be at least 2mm (to reach the best positioning effect);

Precondition:

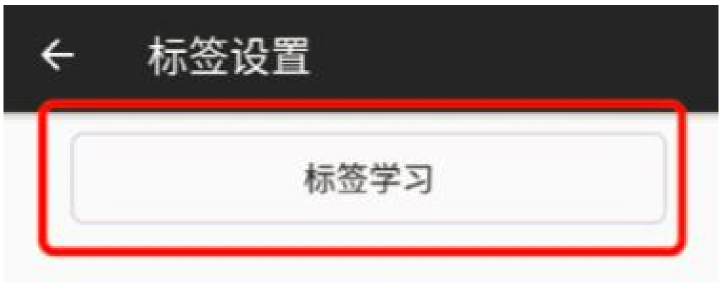
1. Set mode – to use label printing function, please open setting->inbuilt printing->printing mode selection and choose label thermal mode. Your selection will be recorded by the device and turned the device into a label printer when printing, please see the picture below:



A developer can get te printer mode interface getPrinterMode() to find out the mode of a printer.

2.Learn label paper – if it is the first time for you to use it or you have changed a different type of label paper, please open setting->inbuilt printing->label learning and click the button learn label paper which shows as the picture below.

This will initiate label paper learning process. Please ensure there are at least 3 pieces of labels. A popoup box will show to indicate the result of learning when the learning process ends.



Label-related interfaces:

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

Print a label:

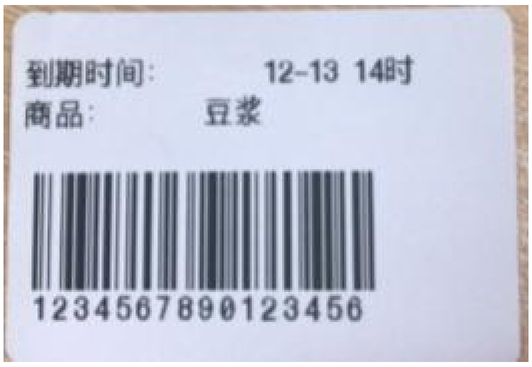
Label-related interfaces only provide how to position a label, so printing a label content shall be formed by developers to their needs, just like what should be done with thermal printing. What should be paid attention to is the height of a label content should be within the hight of a label, otherwise the content may exceed a label paper to the next label or lead to an inaccurate label positioning.

If you use a 30mm-high label paper, a content of 240 (30mmx8 dots) pixel-high can be printed. The default print line spacing is 32 pixel line, so an 8-line text content or a 384x240 high picture can be printed normally.

For example – how to form a simple label content:

```
labelLocate()                    //Positioning the label before printing is required
setPrinterStyle(WoyouConsts.ENABLE_BOLD, WoyouConsts.ENABLE)
                                 //Set to realize bold fonts
 linewrap(1, null)               //Leave a blank line at the head
 setAlignment(0, null)           //Set the content to align left
 printText("商品  Item:      豆豆浆  Soybean milk\n", null)
                                //Print content, \n line feed is needed
 printText("到期时间  Expiration time:     12-13 14 时  12-13 14:00\n", null)
                                //Print content, \n line feed is needed
 printBarCode("{C1234567890123456", 8, 90, 2, 2, null)
                                 //Print a barcode. Here the barcode printed is a code128c
type barcode with a height of 90, which occupies 154 pixel line (a blank line above and
below the barcode will be left)
 printText("\n", null)           //The barcode needs to be outputted with a line feed
 labelOutput()                   //The label will be outputted in accordance with needs
after printing
```

A content basically take all the area of a label will be outputted, please see the picture below:



A developer can add other APIs to design his/her desired label content.

Print more than one label:

If to print only one label, you can:

labelLocate -> Print label -> labelOutput

If to print more than one label, you do not have to send label content and conduct labelOutput to output labels every time, you only need to:

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

Note: Currently label can only be used through interface method instead of Bluetooth ESC command.

1.3 . Description of Interface Return Result

1.3.1.Description of ICallback Interface Methods

No.	Method
1	void **onRunResult(**boolean isSuccess**)** The result of instruction excution
2	void **onReturnString(**String result**)** The return result string of instruction execution
3	void **onRaiseException(**int code, String msg**)** The return exception
4	void **onPrinterResult(**int code, String msg**)** The feedback on transaction printing

1. Result on Instruction Execution

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

Parameters:

isSuccess	execution result:	
	true	succeeded
	false	failed

Note: this interface return result indicates the result of executing an instruction, instead of the result of printing a paper.

2. Return Result on Instruction Execution

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

Parameters

result	result on execution

3. Return Exception Message

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

Parameter:

code	exception code
msg	description on an exception

Note: see Exception Table.

4. Feedback on Transaction Print

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

Parameters:

code	status code:	
	0	succeeded
	1	failed

msg succeeded: null; failed: description on the exception.

Note: this interface return result indicates the results of instrctuction execution and printing (it may **take some time for printing**).

1.3.2. Examples on Callback Object

```
new ICallback.Stub(){
        @Override
        public void onRunResult(boolean b) throws RemoteException {
            // Instruction execution result
        }
        @Override
        public void onReturnString(String s) throws RemoteException {
            // String Return String on instruction execution
        }
        @Override
        public void onRaiseException(int i, String s) throws RemoteException {
            // Return Exception Message
        }
        @Override
        public void onPrintResult(int i, String s) throws RemoteException {
              // Feedback on Transaction Print
        }
     }
```

1.3.3. Exception Table

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 a Printer via the Inbuilt Virtual Bluetooth

2.1. Intro on Virtual Bluetooth

You can find a Bluetooth device "InnerPrinter" (it has already been paired and always stays in the list of Bluetooth list), it is a virtual Bluetooth printer invented by the OS. It supports SUNMI esc/pos instructions.

Among all the instructions, some are SUNMI customized instructions. For example:

Function	Instruction
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 (but save a little bit in the left side)	byte [4]: 0x1d 0x56 0x41 0x00

2.2. How to Use the Virtual Bluetooth

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

Note: **BluetoothUtil** is a Bluetooth tool class, which is 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 on Connecting Printing Service with 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 devices can only be used by adding Bluetooth Permission Decalaration to the project in the App:

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

3. Via JS in HTML

3.1. Description of JS

Essentially, calling a printer via JS is using JS bridge in HTML to operate Android native codes to print, and print via Bluetooth or via AIDL. (SUNMI printer itself is not a network printer, which means that the webpage application can not directly communicate with the printer and an app to receive data on Andriod is needed.)

3.2. How to Use HTML

The detailed calling processes are as follows:

a. Define the method to be used in Android in HTML.

```
document.getElementsByTagName('a')[0].addEventListener('click',
function(){
      var a = "wellcome to sunmi"; javascript:lee.funAndroid(a);
      return false; }, false);
```

b. Initialize WebView.

```
WebView mWebView = (WebView) findViewById(R.id.wv_view);
 // set the encoding
mWebView.getSettings().setDefaultTextEncodingName("utf-8");
 // support js
mWebView.getSettings().setJavaScriptEnabled(true);
mWebView.setWebChromeClient(new WebChromeClient());
```

c. Initialize Print Service

```
Intent intent = new Intent();
intent.setPackage("woyou.aidlservice.jiuiv5");
intent.setAction("woyou.aidlservice.jiuiv5.IWoyouService");
startService(intent);//Start printer service
bindService(intent, connService, Context.BIND_AUTO_CREATE);
```

d. Add Monitor to WebView and Call in the onPageFinished Callback Method

WebView.addJavascriptInterface(new JsObject(), 'lee');

Define the operation method in HTML through @JavascriptInterface in the JsObject class, and print a receipt by calling via AIDL print method in the method.

```
//Add a corresponding webpage monitor class
mWebView.setWebViewClient(new WebViewClientDemo());
WebViewClientDemo extends WebViewClient {
    @Override
    public boolean shouldOverrideUrlLoading(WebView view, String url) {
        //The current WebView will be used when opening a new link; other
    system browsers won't be used.
        view.loadUrl(url);
        return true;
    }
    @Override
    public void onPageFinished(WebView view, String url) {
        super.onPageFinished(view, url);
        /**
        * Register JavascriptInterface. You can name "lee" whatever you want.
        * If you use "lee", you only need to use lee. method name () in html
        * And you can call the method with the same name in
MyJavascriptInterface, and the parameters shall be the same.
        */
        mWebView.addJavascriptInterface(new JsObject(), "lee");
    }
}
class JsObject {
    @JavascriptInterface
    public void funAndroid(final String i) {
        Toast.makeText(getApplicationContext(), "calling the local method
    funAndroid by JS. " + i, Toast.LENGTH_SHORT).show();
        try {
            woyouService.printerSelfChecking(callback);//Using AIDL to print
        something.
        } catch (RemoteException e) {
            e.printStackTrace();
        }
    }
}
```

e. Load HTML File, and a Receipt Will be Printed When Clicking the Button in HTML

```
// Load the html which contains js
mWebView.loadData("", "text/html", null);
mWebView.loadUrl("file:///android_asset/test.html");//here is the
webpage containing your business html
```

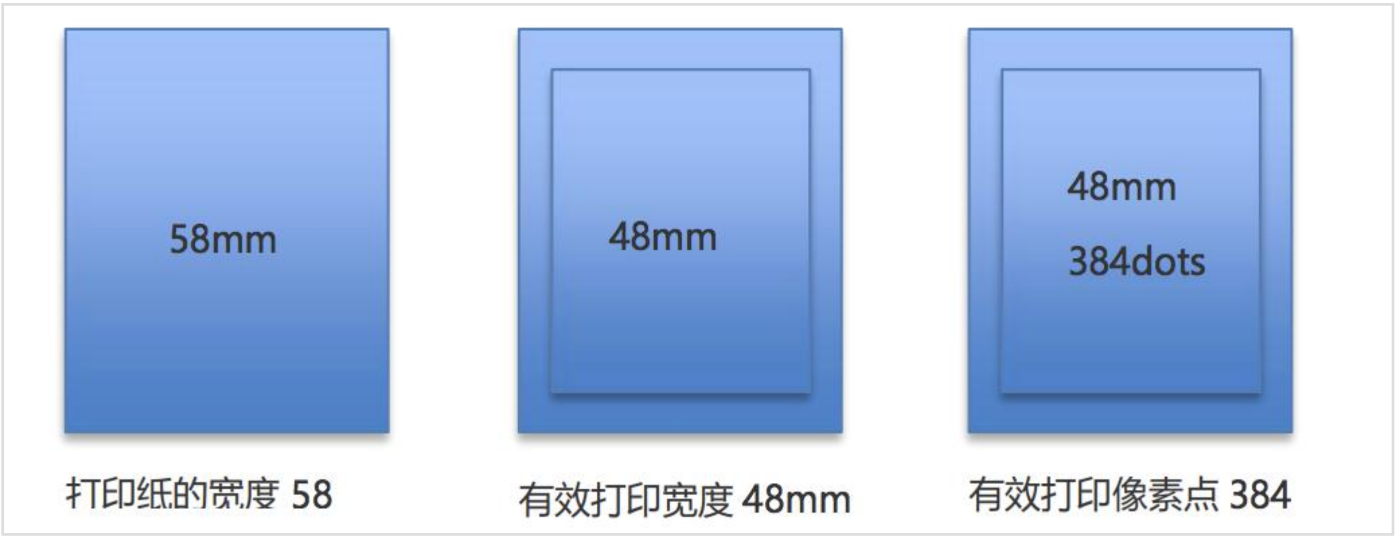
Appendix A Printing Service Broadcast

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 error	"woyou.aidlservice.jiuv5.ERROR_ACTION"
Out of paper	"woyou.aidlservice.jiuv5.OUT_OF_PAPER_ACTION"
Overheated printhead	"woyou.aidlservice.jiuv5.OVER_HEATING_ACITON"
The temperature of the printhead is back to normal	"woyou.aidlservice.jiuv5.NORMAL_HEATING_ACITON"
Open the lid	"woyou.aidlservice.jiuv5.COVER_OPEN_ACTION"
Something went wrong when closing the lid	"woyou.aidlservice.jiuv5.COVER_ERROR_ACTION"
Cutter error1 - block	"woyou.aidlservice.jiuv5.KNIFE_ERROR_ACTION_1"
Cutter error2 - fixed	"woyou.aidlservice.jiuv5.KNIFE_ERROR_ACTION_2"
The printer firmware starts upgrading	"woyou.aidlservice.jiuv5.FIRMWARE_UPDATING_ACITON"
Failed to upgrade the printer firmware	"woyou.aidlservice.jiuv5.FIRMWARE_FAILURE_ACITON"
No printer has been found	"woyou.aidlservice.jiuv5.PRINTER_NON_EXISTENT_ACITON"
No black mark has been found	"woyou.aidlservice.jiuv5.BLACKLABEL_NON_EXISTENT_ACITON"

Appendix B Printing Service FAQ

1. Description on the Specs of Printing Paper



Paper width: 58mm
Active print width: 48mm
Active print pixels: 384

Note: SUNMI printers support printing papers of 58mm, 80mm width. This document takes the printing paper of 58mm width as an example to illustrate; the spec of paper of 80mm width is similar to the example in this document.

The width of the 58 printing paper is 58mm, and the active print width is 48mm, the active print pixels per line are 384. The depth of V1 paper compartment is 40mm which can accommodate a 40mm-wide paper roll maximum.

2. The Resolution of a Printer

A 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 Printer?

Desktop devices can be divided into two types – with printer hardware or without printer hardware; users can find whether there is a printer through the query interface for printers.

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 can’t I print the image even though I execute the image printing?

First of all, the content to be printed (except QR code) which cannot fill a line will be saved in the buffer. If you find the image cannot be printed out after calling the printBitmap interface, the reason might be 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 exception. Commonly, 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 spec. As a result of this, developers need to adjust the size of the image to be printed.

Additionally, if a developer sends a raster bitmap through Bluetooth hexadecimal instruction and the printing failed, the developer needs to check whether there is a mistake in the instruction, for a byte data error might affect the realization of the whole instruction.

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

Due to the width limit of a printing paper (handheld device – 384 pixels wide; desktop device – 576 pixels wide), the barcode with pixels exceeds the limit cannot be displayed fully. 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 the interface **printBarCode** and select code128. This interface has realized mixed encoding, which means you can change encoding type by adding {A, {B, {C to dynamically switch encoding types(A: numbers, upper-case letters, and control characters, etc. B: numbers, upper- and lower-case letters and some additional 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 other characters, then you can print a long barcode for example to print ABab1212: pass "{BABab{C1212";

Mixed encoding is only available to a version 4.0.0 and above. If it is not available to the developer’s print service or the developer prints via Bluetooth, an array of Epson instructions needs to be obtained by introducing the following method, and send the return array through sendRawData interface or Bluetooth.

Among which, data is the direct incoming barcode content to be printed; the minimum width value is 2 pixels by default, and a width smaller than this value will drastically slow the code scanning speed. However, to print a barcode exceeding 25 or more, the value can be set as 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 cannot I receive callback results?

AIDL resource document – P series, V1S and V2; AIDL resource document - T series and S series; AIDL resource document - Mini series; AIDL resource document - series:

First of all, if a developer accesses an interface via AIDL method, attention should be paid on whether the suitable AIDL resource has been used.

On the premise of using the suitable resource document, please be sure that you have introduced the static nested class Stub! in callback class generated by AIDL introduced.

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

This problem won't occur if the developer using remote introduction library follows the document and uses the encapsulated InnerPrinterCallback class, etc.

7. How to select and set a character set.

Background: the inbuilt printers are compatible with the transmission in a form of binary byte stream, so a character set must be selected and set for the text to be printed which is sent in a form of byte code; multi-byte, GB18030 character encoding are the default device settings.

The single-byte encoding types for inbuilt printers are:

```
[参数] [编码] [国家]
[Parameter] [Encoding] [Country]
0 "CP437"; 美国欧洲标准 The USA, European standard
2 "CP850"; 多语言 Multiple languages
3 "CP860"; 葡萄牙语 Portuguese
4 “CP863"; 加拿大 -法语 Canada - French
5 “CP865"; 北欧 Northern Europe
13 "CP857"; 土耳其语 Turkish
14 "CP737"; 希腊语 Greek
15 "CP928";
16 "Windows-1252";
17 "CP866"; 西里尔文 Cyrillic
18 "CP852"; 拉丁-中欧语 Latin - Central European
19 "CP858";
21 "CP874";
33 "Windows-775"; 波罗的海语 Baltic
34 "CP855"; 西里尔文  Cyrillic
36 "CP862"; 希伯来语  Hebrew
37 "CP864";
254 "CP855"; 西里尔文  Cyrillic
```

The multiple-byte encoding types for inbuilt printers are:

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

The device settings can be adjusted in accordance to the needs of different countries or other requirements to enable the printers to identify the data stream of the content to be printed. 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 traditional characters, please send:
0x1C 0x26 ——set it as the multiple-byte encoding type
0x1C 0x43 0x01—— set it as the BIG5 encoding of the multiple-byte code page
To print UTF-8 encoding content (all Unicode character sets are supported if using UTF-8 encoding, and all content can be printed), please send:
0x1C 0x26 ——set it as the multiple-byte encoding type
0x1C 0x43 0xFF—— Set it as the UTF-8 encoding of the multiple-type code page

8. Description of black mark print

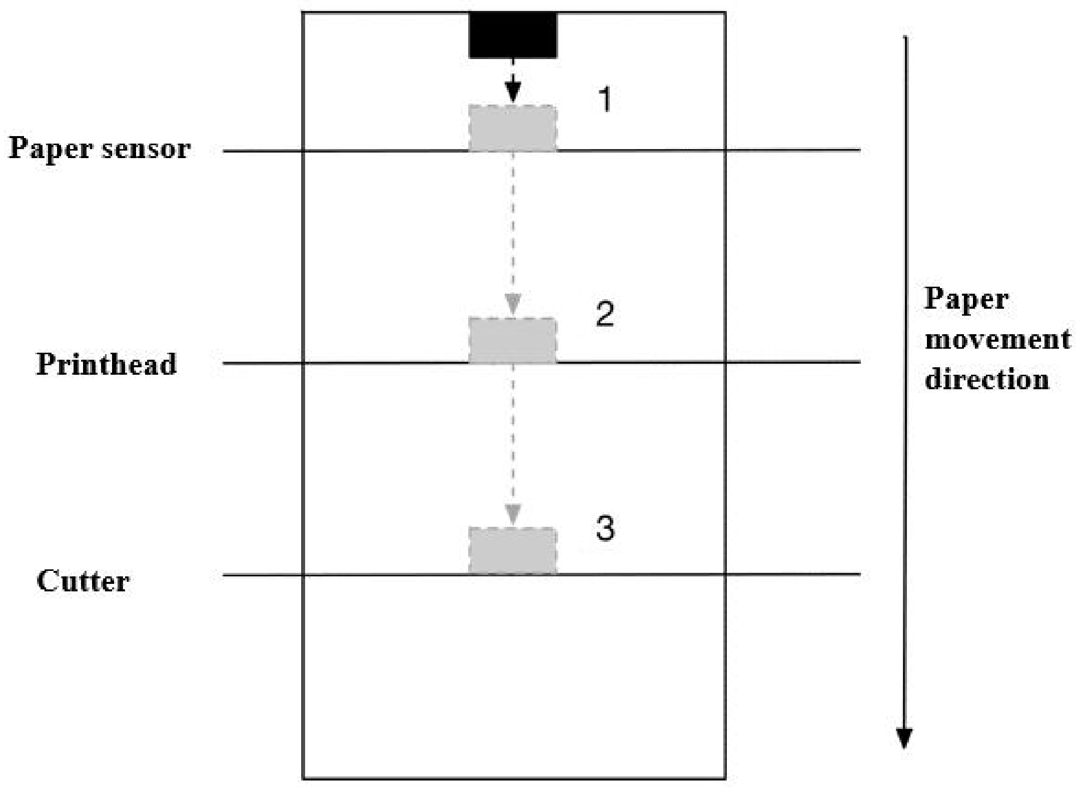
SUNMI T1 can print using the printing paper with black marks which comply to the standard.

Requirements on the printing paper with black marks: different from other black mark printers, SUNMI T1 printer has no sensor to detect black marks but provide a black mark printing function through reflectivity (if the reflectivity is not bigger than 6% (Infrared wave lengths within 700-1000nm), the paper sensor will detect it as no paper). Due to the different theory used, a SUNMI T1 printer in black mark print mode cannot accurately locate the black mark as other black mark printers and there are 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 it.

The theory SUNMI T1 used to realize black mark print:

The paper sensor and the cutter are not in the same horizontal line, and 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).d



In black mark print mode, if the printer detects that there is no paper, it will keeping moving 7mm. After moving 7mm, if no paper has been detected, the printer will process it as no paper, and if paper is detected after moving 7mm, the printer will process it as black mark print.

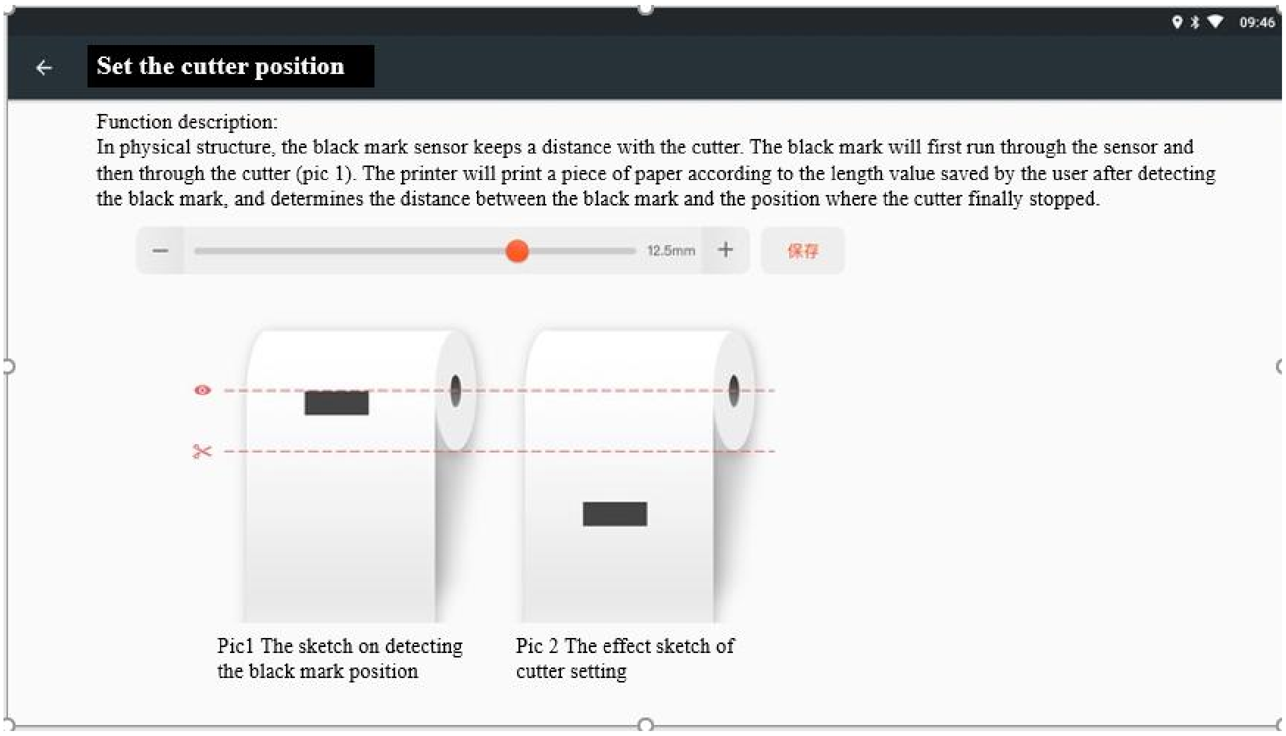
If the content to be printed covers the black mark, the printing will be proceeded, which means if the black mark arrives at the location of 2 and there is still a print job, the print job will continue till it is completed.

In this mechanism, the end of content to be printed should keep a certain distance with paper sensor (the end of content should keep a distance of 6mm with the black mark edge) to ensure that the content is between 2 black marks.

Instruction sequence in the black mark print mode:

1. send the content to be printed
2. enter the instruction on moving the paper till the black mark has been detected {0x1c, 0x28, 0x4c, 0x02, 0x00, 0x42, 0x31}
3. enter the cutter instruction {0x1d, 0x56, 0x00}

Users can change the black mark print mode and the location of cutter in Setting->Print->Inbuilt print management.



Function description:
In physical structure, the black mark sensor keeps a distance with the cutter. The black mark will first run through the sensor and then through the cutter (pic 1). The printer will print a piece of paper according to the length value saved by the user after detecting the black mark, and determines the distance between the black mark and the position where the cutter finally stopped.

Pic1 The sketch on detecting the black mark position

Pic 2 The effect sketch of cutter setting