

# CPCL SDK v1.0.02 Interface documentation

---

## Document modification records

### I. Instructions for Use

- 1.1 Engineering Configuration Description
- 1.2 Obfuscated Configuration
- 1.3 Initialization

### II. Sample Demonstration

- 2.1 Connect Printer Process
  - 2.1.1 Bluetooth Search Printer
  - 2.1.2 Connecting the printer

### III. Interface Description

PrinterHelper introduced

- 3.1.2 Bluetooth connection
- 3.1.3 WIFI connection
- 3.1.4 USB connection
- 3.1.5 Connection status
- 3.1.7 Disconnection
- 3.2.1 Instantiation Print Class
- 3.2.2 set label start parameters
- 3.2.3 Set label end
- 3.2.4 Head and tail inverted label printing
- 3.2.5 Set coding
- 3.2.6 Return paper
- 3.2.7 Label positioning
- 3.2.8 Text Printing
- 3.2.9 Font Scaling
- 3.2.10 Bold Font
- 3.2.11 Setting Alignment
- 3.2.12 Printing Barcode

- 3.2.13 Print QR Code
- 3.2.14 Print PDF417
- 3.2.15 Printing Rectangle
- 3.2.16 Print Line
- 3.2.17 Printing Pictures
- 3.2.18 Get Printer Status
- 3.2.19 Send Data
- 3.2.20 Read Data
- 3.2.21 Leak-proof function
- 4.1.1 Coding Table
- 4.1.2 Font Direction
- 4.1.3 Font Size
- 4.1.4 Barcode Type
- 4.1.5 Proportion of wide and narrow strips
- 4.1.6 Enable the SDK to send data logs

## Document modification records

serial Number	version Number	modify content	modifier	review	date modified
01	v1.0.0	document creation	qiux	ldc	2025-05-13
02	v1.0.01	Anti-leakage function	qiux	ldc	2025-06-19
03	v1.0.02	Added a function to obtain print results without ID	qiux	ldc	2025-09-09

# I. Instructions for Use

## 1.1 Engineering Configuration Description

1. This SDK interface needs to run on Android system
2. Put the lzo-sdk-v x.x.x.aar android-common-sdk-vx.x.x.aar into the project libs

directory, and add build.gradle to introduce aar

▼ build.gradle

Java

```
1  implementation(mapOf("name" to "lzo-sdk-v1.00.00", "ext" to "aar"))
2  implementation(mapOf("name" to "android-common-sdk-v1.00.00", "ext" to "aar"))
```

3. Bluetooth permission configuration

▼ AndroidManifest.xml

XML

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

3  <uses-permission android:name="android.permission.BLUETOOTH_CONNECT"/>
4  <uses-permission android:name="android.permission.BLUETOOTH_SCAN"/>
5  <uses-permission android:name="android.permission.BLUETOOTH_ADVERTISE"/>
>
```

▼ Dynamically configure location permissions in code

Kotlin

```
1  disposable = RxPermissions(this).request(
2      Manifest.permission.BLUETOOTH,
3      Manifest.permission.ACCESS_FINE_LOCATION,
4      Manifest.permission.ACCESS_COARSE_LOCATION
5  )
6  .subscribe {
7      if (it) {
8      }}
```

## 1.2 Obfuscated Configuration

1. Need to configure the code in the proguard-rules.pro to prevent SDK from being confused

▼ Do not confuse the following classes

Kotlin |

```
1 ▼ -keep class com.common.sdk.** { *; }
2 -keepnames class com.common.sdk.**
3 -keeppackageNames com.common.sdk.**
```

## 1.3 Initialization

1. Call the following code in the Application.

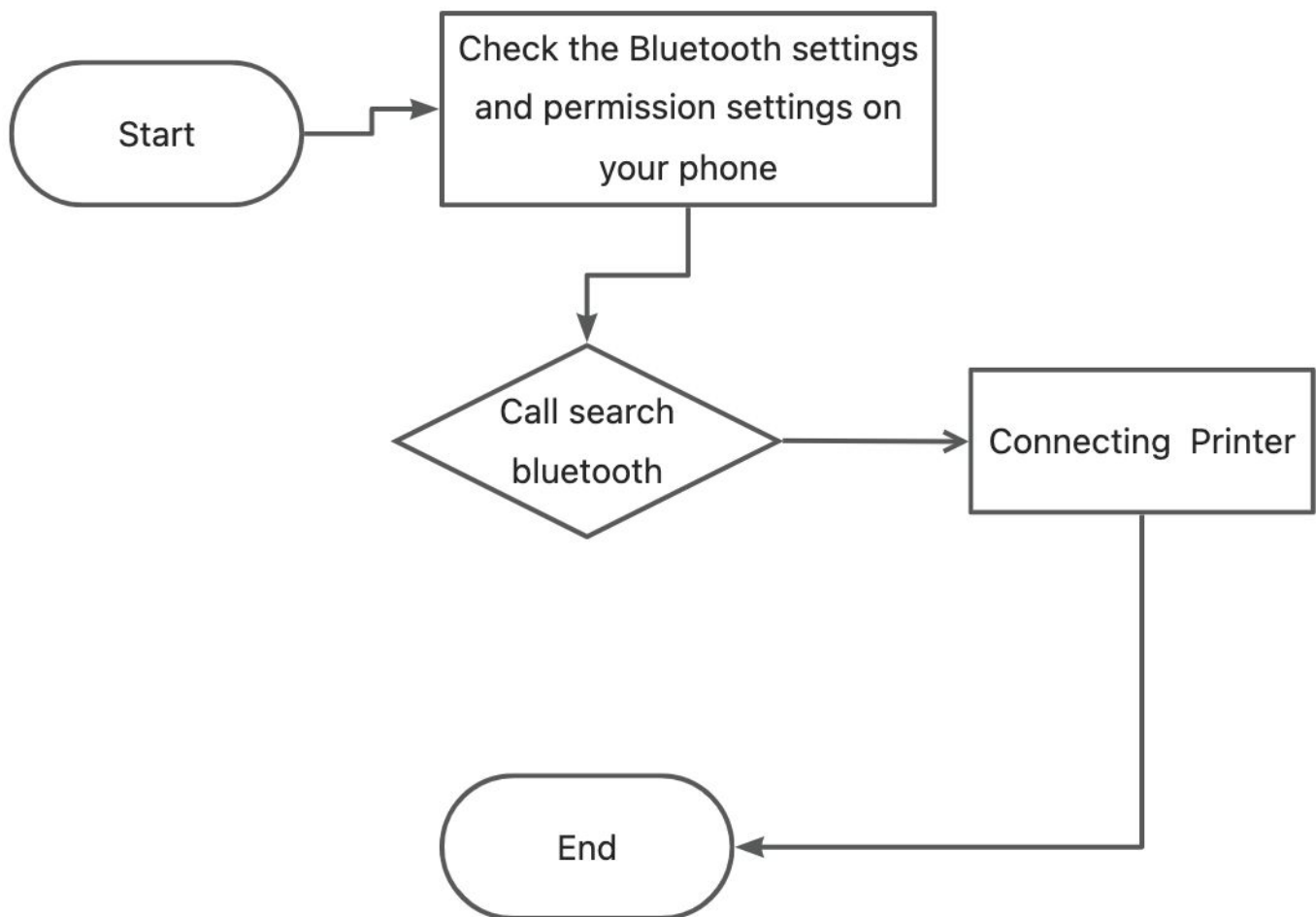
▼ initialization

Kotlin |

```
1 PrinterHelper.init(this);
```

## II. Sample Demonstration

### 2.1 Connect Printer Process



## 2.1.1 Bluetooth Search Printer

### ▼ Registering Bluetooth Broadcaster

Java |

```
1 private void registerBroadcast() {
2     IntentFilter intent = new IntentFilter();
3     intent.addAction(BluetoothDevice.ACTION_FOUND);
4     intent.addAction(BluetoothDevice.ACTION_BOND_STATE_CHANGED);
5     intent.addAction(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);
6     context.registerReceiver(mReceiver, intent);
7 }
8
```

### ▼ Search Bluetooth

Java |

```
1 RxPermissions rxPermissions = new RxPermissions((AppCompatActivity) context);
2 rxPermissions.request(Manifest.permission.BLUETOOTH_ADMIN,
3                       Manifest.permission.BLUETOOTH,
4                       Manifest.permission.ACCESS_FINE_LOCATION)
5 .subscribe(aBoolean -> {
6     if (aBoolean) {
7         if (null == mBluetoothAdapter) {
8             mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
9         }
10        if (!mBluetoothAdapter.isEnabled()) {
11            mBluetoothAdapter.enable();
12        }
13        if (mBluetoothAdapter.isDiscovering()) {
14            mBluetoothAdapter.cancelDiscovery();
15        }
16        mBluetoothAdapter.startDiscovery();//开启搜索
17    } else {
18        Utility.show(context, "no bluetooth permission");
19    }
20 });
```

```
1 private final BroadcastReceiver mReceiver = new BroadcastReceiver() {
2     @SuppressWarnings("MissingPermission")
3     @Override
4     public void onReceive(Context context, Intent intent) {
5         String action = intent.getAction();
6         if (BluetoothDevice.ACTION_FOUND.equals(action)) {
7             BluetoothDevice device =
8                 intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
9             if (device.getBluetoothClass().getMajorDeviceClass() == 1536)
10            {
11                //classic Bluetooth protocol
12            }
13            break;
14        }
15    };
```

## 2.1.2 Connecting the printer

```
1 val connection = BTConnection()
2 val resultCode = connection.connectBT(mac!!)
3 //resultCode:
4 //0: connection successful,
5 //-1: connection failed (timeout),
6 //-2: connection failed (parameter error),
7 //-3: connection failed (uninitialized))
```

## III. Interface Description

### PrinterHelper introduced

initialize PrinterHelper at APP startup

```
1 public class App extends Application {
2     @Override
3     public void onCreate() {
4         super.onCreate();
5         PrinterHelper.init(this);
6     }
7 }
```

### 3.1.2 Bluetooth connection

- pay attention to permission configuration

```
1 Bluetooth connection requires configuration permissions,
2 add in the AndroidManifest.xml file
3 <uses-permission android:name="android.permission.BLUETOOTH" />
4 <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
5 <uses-permission android:name="android.permission.BLUETOOTH_CONNECT"/>
6 <uses-permission android:name="android.permission.BLUETOOTH_SCAN"/>
7 <uses-permission android:name="android.permission.BLUETOOTH_ADVERTISE"/>
8
9 Then dynamically obtain permissions in the code
10 RxPermissions rxPermissions = new RxPermissions(this);
11 rxPermissions.request(Manifest.permission.BLUETOOTH_ADMIN,
12                     Manifest.permission.BLUETOOTH,
13                     Manifest.permission.BLUETOOTH_CONNECT,
14                     Manifest.permission.BLUETOOTH_SCAN,
15                     Manifest.permission.ACCESS_FINE_LOCATION)
16 .subscribe(aBoolean -> {
17     if (aBoolean) {
18         //Permissions obtained successfully
19     }
20 });
```

- Description

connect printer via Bluetooth mac address

▼ Bluetooth connect

Kotlin |

```
1 fun connectBT(mac: String): Int
```

- Parameters

parameters	description
mac	bluetooth address (uppercase)

- Examples

▼ Bluetooth connection example

Java |

```
1 val connection = BTConnection()  
2 val resultCode = connection.connectBT(mac!!)  
3 //resultCode:  
4 //0: connection successful,  
5 //-1: connection failed (timeout),  
6 //-2: connection failed (parameter error),  
7 //-3: connection failed (uninitialized))
```

### 3.1.3 WIFI connection

- note Configuration Permissions

▼ wifi permission configuration

Java |

```

1 //Wi-Fi connection requires configuration permissions,
2 //add in AndroidManifest.xml file
3 <uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
4 <uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
5 <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
6 <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
7 <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"
  />
8 <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
9
10 //Dynamic permissions are required in the code
11 RxPermissions rxPermissions = new RxPermissions(this);
12 rxPermissions.request(Manifest.permission.ACCESS_COARSE_LOCATION,
13                      Manifest.permission.ACCESS_FINE_LOCATION)
14 .subscribe(aBoolean -> {
15     if (aBoolean) {
16         //Permissions obtained successfully
17     }
18 });

```

- **Description**

connect through the IP address of the printer

▼ wifi connect

Kotlin |

```

1 fun connectWifi(ip: String): Int

```

- **Parameters**

parameters	description
ip	the IP address of the printer (example: 192.168.1.100)

- **Examples**

#### ▼ Wifi Connection Example

Java |

```
1 val connection = WifiConnect()
2 val resultCode = connection.connectWifi(ip)
3 //resultCode:
4 //0: connection successful,
5 //-1: connection failed (timeout),
6 //-2: connection failed (parameter error),
7 //-3: connection failed (uninitialized))
```

### 3.1.4 USB connection

- pay attention to permission configuration

#### ▼ USB permission configuration

Java |

```
1 //The USB connection requires configuration permissions,
2 //which are added in the AndroidManifest.xml file
3 <uses-feature android:name="android.hardware.usb.host" />
4 <uses-permission android:name="android.permission.USB_PERMISSION" />
5 <uses-permission android:name="android.permission.ACCESS_USB_DEVICE" />
6 <meta-data android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED" />
7 </meta-data>
```

- **Description**

connecting the printer via usbdevice

#### ▼ USB permission configuration

Java |

```
1 fun connectUSB(usbDevice: UsbDevice): Int
```

- **Parameters**

parameters	description
usbDevice	usb device class provided by android system

- **Examples**

```
1  val usbConnect = USBConnect()
2  val connectUSB = usbConnect.connectUSB(device!!)
3  //resultCode:
4  //0: connection successful,
5  //-1: connection failed (timeout),
6  //-2: connection failed (parameter error),
7  //-3: connection failed (uninitialized))
```

```
1  //device: You can refer to the Demo to obtain it,
2  //or you can refer to the following code
3  private void connectUSB() { // Traverse all USB devices in the system
4      mUsbManager = (UsbManager) thisCon.getSystemService(Context.USB_SERVICE);
5      HashMap<String, UsbDevice> deviceList = mUsbManager.getDeviceList();
6      Iterator<UsbDevice> deviceIterator = deviceList.values().iterator();
7
8      boolean HavePrinter = false;
9      while (deviceIterator.hasNext()) {
10         device = deviceIterator.next();
11         int count = device.getInterfaceCount();
12         for (int i = 0; i < count; i++) {
13             UsbInterface intf = device.getInterface(i);
14             if (intf.getInterfaceClass() == 7) { // Represents the printer type
15                 HavePrinter = true;
16                 if (mPermissionIntent != null) {
17                     // Apply for USB permission
18                     mUsbManager.requestPermission(device, mPermissionIntent);
19                 }
20             }
21         }
22     }
23 }
```

## ▼ Register USB system permission broadcast

Java

```

1  Intent intent = new Intent(ACTION_USB_PERMISSION);
2  intent.setPackage(thisCon.getPackageName());
3  if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
4      mPermissionIntent = PendingIntent.getBroadcast(thisCon, 0, intent, FLAG_MUTABLE);
5  } else {
6      mPermissionIntent = PendingIntent.getBroadcast(thisCon, 0, intent, 0);
7  }
8  IntentFilter filter = new IntentFilter(ACTION_USB_PERMISSION);
9  filter.addAction(UsbManager.ACTION_USB_DEVICE_DETACHED);
10 filter.addAction(BluetoothDevice.ACTION_ACL_DISCONNECTED);
11 filter.addAction(BluetoothAdapter.ACTION_STATE_CHANGED);
12 if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
13     thisCon.registerReceiver(mUsbReceiver, filter, RECEIVER_EXPORTED);
14 } else {
15     thisCon.registerReceiver(mUsbReceiver, filter);
16 }

```

## ▼ Receiving Broadcasts

Java

```

1  private val mUsbReceiver: BroadcastReceiver = object : BroadcastReceiver() {
2      override fun onReceive(context: Context, intent: Intent) {
3          try {
4              val action = intent.action
5              if (ACTION_USB_PERMISSION == action) {
6                  synchronized(this) {
7                      val device =
8                          intent.getParcelableExtra<Parcelable>(UsbManager.EXTRA_DEVICE)
9                      as UsbDevice?
10                     if (intent.getBooleanExtra(UsbManager.EXTRA_PERMISSION_GRANTED, false)) {
11                         val usbConnect = USBConnect()
12                         val connectUSB = usbConnect.connectUSB(device!!)
13                         if (connectUSB == 0) {
14                             //Connection successful
15                         } else {
16                             //Connection failed
17                         }
18                     } else {
19                         return
20                     }
21                 }
22             } catch (e: Exception) {
23             }
24         }
25     }

```

---

### 3.1.5 Connection status

- **Description**

determine whether the printer is connected

- ▼ Connection Status Interface

Java |

```
1 fun isConnect(): Boolean
2 //This interface cannot obtain real-time status.
3 //Bluetooth can be obtained through system Bluetooth broadcast.
4 //Refer to Demo
```

- **Examples**

- ▼ Connection Status Example

Java |

```
1 connection.isConnect()
2 //true: connected,
3 //false: not connected
4 //connection is obtained by different connection classes,
5 //Bluetooth is BTConnection,
6 //wifi is WifiConnect,
7 //USB is USBConnect
```

---

### 3.1.7 Disconnection

- **Description**

disconnect a connected device



Java |

```
1 disconnect(): Boolean
```

- **Examples**

```

1 connection?.disconnect()
2 //connection is obtained by different connection classes,
3 //Bluetooth is BTConnection,
4 //wifi is WifiConnect,
5 //USB is USBConnect

```

### 3.2.1 Instantiation Print Class

- **Description**

all print-related interfaces are in the PrinterCPCLHelper class

```

1 class PrinterCPCLHelper(baseConnection: BaseConnection)

```

- **Parameters**

parameters	description
baseConnection	get by creating a connection

- **Examples**

```

1 val cpclHelper = PrinterCPCLHelper(baseConnection)

```

### 3.2.2 set label start parameters

- **Description**

this interface can set some parameters of printing labels, such as left margin, resolution, label height, number of copies printed

```

1  /**
2   * Set the parameters at the beginning of the label
3   * @param offset String Left Margin (Dot)
4   * @param horizontal String Landscape resolution (depending on the printer)
5   * @param vertical String Portrait resolution (depending on the printer)
6   * @param height String Label Height (dots)
7   * @param qty String Number of copies printed
8   * @return Boolean Whether the message is successful
9   */
10 fun printAreaSize(offset: String, horizontal: String, vertical: String,
11                   height: String, qty: String): Boolean

```

- Examples

```

1  cpclHelper?.printAreaSize("0","200","200", bitmapPrint.height.toString(),
2    "1")
3  cpclHelper?.printBitmapCPCL(bitmapPrint,0,0,
4    if (type == 0) ImageTypeEnum.THRESHOLD
5    else if(type == 1) ImageTypeEnum.RASTERMONO
6    else ImageTypeEnum.POLYMERIZATION, compressType, 0)
7  cpclHelper?.print()

```

### 3.2.3 Set label end

- Description

will let the printer print

```

1  fun print(): Boolean

```

- Examples

```

1  cpclHelper?.printAreaSize("0","200","200", bitmapPrint.height.toString(),
    "1")
2  cpclHelper?.printBitmapCPCL(bitmapPrint,0,0,
3  if (type == 0) ImageTypeEnum.THRESHOLD
4  else if(type == 1) ImageTypeEnum.RASTERMONO
5  else ImageTypeEnum.POLYMERIZATION, compressType, 0)
6  cpclHelper?.print()
7  //false: failed to send, true: sent

```

### 3.2.4 Head and tail inverted label printing

- **Description**

let the printer perform printing and rotate the content 180 degrees. Cannot be used with print() interface

```

1  poPrint(): Boolean

```

- **Examples**

```

1  cpclHelper?.printAreaSize("0","200","200", bitmapPrint.height.toString(),
    "1")
2  cpclHelper?.printBitmapCPCL(bitmapPrint,0,0,
3  if (type == 0) ImageTypeEnum.THRESHOLD
4  else if(type == 1) ImageTypeEnum.RASTERMONO
5  else ImageTypeEnum.POLYMERIZATION, compressType, 0)
6  cpclHelper?.poPrint()

```

### 3.2.5 Set coding

- **Description**

set printer and SDK encoding

```

1  /**
2   * Set coding
3   * @param code String Encoding (see Table 4.1.1)
4   * @return Boolean Whether the message is successful
5   */
6  fun encoding(code: String): Boolean

```

- Examples

```

1  cpclHelper?.encoding("gb2312")

```

### 3.2.6 Return paper

- Description

printer back paper

```

1  /**
2   * Return paper
3   * @param backFeed String Number of lines in the paper
4   * @return Boolean Whether to send or not
5   */
6  fun backFeed(backFeed: String): Boolean

```

- Examples

```

1  cpclHelper?.backFeed("100")

```

### 3.2.7 Label positioning

- Description

this interface is used to position the seam mark in the label mode.

```

1  /**
2   * Label positioning
3   * @return Boolean Whether to send or not
4   */
5   fun form(): Boolean

```

- Examples

```

1  cpclHelper?.printAreaSize("0","200","200", bitmapPrint.height.toString(),
2   "1")
3  cpclHelper?.printBitmapCPCL(bitmapPrint,0,0,
4   if (type == 0) ImageTypeEnum.THRESHOLD
5   else if(type == 1) ImageTypeEnum.RASTERMONO
6   else ImageTypeEnum.POLYMERIZATION, compressType, 0)
7  cpclHelper?.form()
8  cpclHelper?.poPrint()

```

### 3.2.8 Text Printing

- Description

printTextCPCL is used for Chinese firmware, printcodetagetextcpcl is used for English firmware, text can be used for both firmware, printTextPro Select Font Print Text

```

1  /**
2   * Text Printing
3   * @param textEnum CPCLTextEnum Direction (see Table 4.1.2)
4   * @param font String Font size (see Table 4.1.3)
5   * @param size String Suspension of use (fixed transmission 0)
6   * @param x String Transverse start coordinates
7   * @param y String Longitudinal start coordinates
8   * @param data String Text data
9   * @return Boolean Whether to send or not
10  */
11  fun text(textEnum: CPCLTextEnum, font: String, size: String,
12           x: String, y: String, data: String): Boolean

```

```

1  /**
2   * Text Printing
3   * @param command CPCLTextEnum Direction (see Table 4.1.2)
4   * @param font Int Font size (see Table 4.1.3)
5   * @param x String Transverse start coordinates
6   * @param y String Longitudinal start coordinates
7   * @param data String Text data
8   * @param n Int Font effect (
9       n&1==1: bold,
10      n&2==2: inverted,
11      n&4==4: double-wide,
12      n&8==8: double-height)
13   * @return Boolean Whether to send or not
14   */
15 fun printCodepageTextCPCL(command: CPCLTextEnum, font: Int ,
16                           x: String ,y: String ,data: String ,n: Int): Bo
    olean

```

```

1  /**
2   * Text Printing
3   * @param command CPCLTextEnum Direction (see Table 4.1.2)
4   * @param font Int Font size (see Table 4.1.3)
5   * @param x String Transverse start coordinates
6   * @param y String Longitudinal start coordinates
7   * @param data String Text data
8   * @param n Int Font effect (
9       n&1==1: bold,
10      n&2==2: inverted,
11      n&4==4: double-wide,
12      n&8==8: double-height)
13   * @param isCenter Boolean Whether it is centered or not
14   * @param width Int The width of the center
15       (isCenter = true is in effect) unit: point
16   * @return Boolean Whether to send or not
17   */
18 fun printTextCPCL(command: CPCLTextEnum ,font: Int ,x: String, y: String,
19                  data: String, n: Int, isCenter: Boolean, width: Int): Bo
    olean

```

```

1  /**
2   * Text Printing
3   * @param command CPCLTextEnum Direction (see Table 4.1.2)
4   * @param fontName String The name of the font (SIMSUN.TTF, TT0003M_.TTF)
5   * @param xScale Int Font magnification in the x-axis direction
6   * @param yScale Int Y-axis direction font magnification
7   * @param x Int Abscissa (in dot)
8   * @param y Int Ordinate (in dot)
9   * @param data String Text data
10  * @return Boolean Whether to send or not
11  */
12  fun printTextPro(command: CPCLTextEnum, fontName: String, xScale: Int,
13                  yScale: Int , x: Int, y: Int, data: String): Boolean

```

- Examples

```

1  cpclHelper?.printAreaSize("0","200","200","500","1")
2  //Make the font below bold (if you don't need to be bold, don't add it)
3  cpclHelper?.setBold("1")
4  //Enlarge the font below (don't add it if you don't need it)
5  cpclHelper?.setMag("2","2")
6  cpclHelper?.text(CPCLTextEnum.TEXT,"7","0","10","10","TEXT")
7  //Turn off zoom-in
8  cpclHelper?.setMag("1","1")
9  //Turn off bolding
10 cpclHelper?.setBold("0")
11 cpclHelper?.form()
12 cpclHelper?.print()

```

### 3.2.9 Font Scaling

- Description

this interface is used to scale the font size of printed text.

```

1  /**
2   * Font Scaling
3   * @param width String Magnification of font width (1-8)1: Not enlarged
4   * @param height String Magnification of font height (1-8)1: Not magnified
5   * @return Boolean Whether to send or not
6   */
7   fun setMag(width: String, height: String): Boolean

```

- Examples

```

1  cpclHelper?.printAreaSize("0","200","200","500","1")
2  //Make the font below bold (if you don't need to be bold, don't add it)
3  cpclHelper?.setBold("1")
4  //Enlarge the font below (don't add it if you don't need it)
5  cpclHelper?.setMag("2","2")
6  cpclHelper?.text(CPCLTextEnum.TEXT,"7","0","10","10","TEXT")
7  //Turn off zoom-in
8  cpclHelper?.setMag("1","1")
9  //Turn off bolding
10 cpclHelper?.setBold("0")
11 cpclHelper?.form()
12 cpclHelper?.print()

```

### 3.2.10 Bold Font

- Description

this interface is used to bold the font of printed text

```

1  /**
2   * Bold Font
3   * @param bold String Bold factor (range: 0-5) 0: off
4   * @return Boolean Whether to send or not
5   */
6   fun setBold(bold: String): Boolean

```

- Examples

```

1  cpclHelper?.printAreaSize("0","200","200","500","1")
2  //Make the font below bold (if you don't need to be bold, don't add it)
3  cpclHelper?.setBold("1")
4  //Enlarge the font below (don't add it if you don't need it)
5  cpclHelper?.setMag("2","2")
6  cpclHelper?.text(CPCLTextEnum.TEXT,"7","0","10","10","TEXT")
7  //Turn off zoom-in
8  cpclHelper?.setMag("1","1")
9  //Turn off bolding
10 cpclHelper?.setBold("0")
11 cpclHelper?.form()
12 cpclHelper?.print()

```

### 3.2.11 Setting Alignment

- **Description**

the interface can be used to align text, bar code, two-dimensional code, pictures, etc.

#### align

```

1  /**
2   * Setting Alignment
3   * @param align CPCLAlignEnum Alignment (
4   *     CENTER: Center,
5   *     LEFT: Left Alignment,
6   *     RIGHT: Right Alignment)
7   * @return Boolean
8   */
9  fun align(align: CPCLAlignEnum): Boolean

```

- **Examples**

```

1  cpclHelper?.printAreaSize("0","200","200","100","1")
2  cpclHelper?.align(CPCLAlignEnum.CENTER)
3  cpclHelper?.text(CPCLTextEnum.TEXT,"4","0","0","0","TEXT")
4  cpclHelper?.form()
5  cpclHelper?.print()

```

## 3.2.12 Printing Barcode

- Description

this interface is used to print bar codes, which can print UPC-A,UPC-E,JAN13/EAN13,JAN8/EAN8,CODE39,ITF,CODABAR(NW-7),CODE93,CODE128 type bar codes

```
1  ▾ /**
2    * Printing Barcode
3    * @param command CPCLBarcodeEnum direction (
4    *    BARCODE: horizontally,
5    *    VBARCODE: vertically)
6    * @param type String Barcode Type (Lookup Table 4.1.4)
7    * @param width String The unit width of the narrow strip
8    * @param ratio String Ratio of wide and narrow strips (see Table 4.1.5)
9    * @param height String Barcode height (unit: dot)
10   * @param x String The start abscissa of the barcode (unit: dot)
11   * @param y String The start ordinate of the barcode (unit: dot)
12   * @param data String Barcode data
13   * @param barcodeUnderText BarcodeUnderText? Whether the text is visible
14   * @return Boolean Whether to send or not
15   */
16   fun barcode(command: CPCLBarcodeEnum, type: String, width: String,
17               ratio: String, height: String, x: String, y: String,
18               data: String, barcodeUnderText: BarcodeUnderText?):Boolean
19
20  ▾ /**
21    * Text style
22    * @param number The type of font
23    * @param size The size of the font
24    * @param offset The distance between the barcode and the text
25    */
26   data class BarcodeUnderText(val number: String, val size: String,
27                               val offset: String)
```

- Examples

```

1  cpclHelper?.printAreaSize("0","200","200","100","1")
2  cpclHelper?.barcode(CPCLBarcodeEnum.BARCODE, "128", "2",
3                      "1", "50", "0", "100", "123456789",
4                      BarcodeUnderText("7", "0", "5"))
5  cpclHelper?.form()
6  cpclHelper?.print()

```

### 3.2.13 Print QR Code

- **Description**

the interface is used to print two-dimensional code

```

1  /**
2   * Print QR Code
3   * @param command CPCLBarcodeEnum direction (
4   *   BARCODE: horizontally,
5   *   VBARCODE: vertically)
6   * @param x String The start abscissa of the QR Code (unit: dot)
7   * @param y String The start ordinate of the QR Code (unit: dot)
8   * @param m String Types of QR
9   *   (1: Normal type,
10   *   2: Individual symbols added to Type 1)
11   * @param u String Unit width/unit height of the module,
12   *   ranging from 1 to 32 defaults to 6
13   * @param data String The data of the QR code
14   * @return Boolean Whether to send or not
15   */
16  fun printQR(command: CPCLBarcodeEnum, x: String , y: String,
17             m: String, u: String, data: String): Boolean

```

- **Examples**

```

1  cpclHelper?.printAreaSize("0","200","200","100","1")
2  cpclHelper?.printQR(CPCLBarcodeEnum.BARCODE, "0", "730", "4", "6", "ABC123"
3  )
4  cpclHelper?.form()
5  cpclHelper?.print()

```

### 3.2.14 Print PDF417

- **Description**

this interface is used to print PDF417 code

```
1  /**
2   * Print PDF417
3   * @param command CPCLBarcodeEnum direction (
4   *     BARCODE: horizontally,
5   *     VBARCODE: vertically)
6   * @param x String The start abscissa (unit: dot)
7   * @param y String The start ordinate (unit: dot)
8   * @param xd String The unit width of the narrowest element.
9   *     The range is 1 to 32, and the default is 2
10  * @param yd String The unit height of the widest element.
11  *     The range is 1 to 32, and the default value is 6
12  * @param c String Number of columns used, data columns excluding
13  *     start/stop characters and left/right indicators,
14  *     ranging from 1 to 30; The default value is 3
15  * @param s String The security level indicates the maximum amount and/or
16  *     correction of errors to be detected and ranges from 0 to
17  *     8;
18  *     The default value is 1
19  * @param data String PDF417 code
20  * @return Boolean Whether to send or not
21  */
22 fun printPDF417(command: CPCLBarcodeEnum, x: String, y: String,
23                xd: String, yd: String, c: String,
24                s: String, data: String): Boolean
```

- **Examples**

```
1  cpclHelper?.printAreaSize("0","200","200","100","1")
2  cpclHelper?.printPDF417(CPCLBarcodeEnum.BARCODE,"0","0","2","6","3","1","12
3  3ABC")
4  cpclHelper?.form()
5  cpclHelper?.print()
```

## 3.2.15 Printing Rectangle

- **Description**

the interface is to draw a rectangle to print

▼ Java

```
1  /**
2   * Printing Rectangle
3   * @param x0 String X coordinates in the upper left corner. (Unit: DOT)
4   * @param y0 String Y coordinates in the upper left corner. (Unit: DOT)
5   * @param x1 String X coordinates in the lower right corner. (Unit: DOT)
6   * @param y1 String Y coordinates in the lower right corner. (Unit: DOT)
7   * @param width String The unit width of the line. (Default: 1)
8   * @return Boolean Whether to send or not
9   */
10 fun box(x0: String, y0: String, x1: String, y1: String, width: String): Boolean
```

- **Examples**

▼ Java

```
1  cpclHelper?.printAreaSize("0","200","200","200","1")
2  cpclHelper?.box("0","0","150","150","1")
3  cpclHelper?.form()
4  cpclHelper?.print()
```

## 3.2.16 Print Line

- **Description**

this interface is used to print straight lines

```
1  /**
2   * Print Line
3   * @param x0 String The starting X coordinate. (Unit: DOT)
4   * @param y0 String The starting Y coordinate. (Unit: DOT)
5   * @param x1 String X coordinates at the end. (Unit: DOT)
6   * @param y1 String Y coordinates at the end. (Unit: DOT)
7   * @param width String The unit width of the line. (Default: 1)
8   * @return Boolean Whether to send or not
9   */
10 fun line(x0: String, y0: String, x1: String, y1: String, width: String): Boolean
```

- Examples

```
1  cpclHelper?.printAreaSize("0","200","200","200","1")
2  cpclHelper?.line("10","10","150","10","1")
3  cpclHelper?.form()
4  cpclHelper?.print()
```

## 3.2.17 Printing Pictures

- Description

this interface is used to print pictures, and the print size is based on the original picture (203DPI, 8dot = 1mm)

```

1  /**
2   * Printing Pictures
3   * @param bitmap Bitmap Bitmap objects
4   * @param x Int Starting abscissa (unit: dot)
5   * @param y Int Start ordinate (unit: dot)
6   * @param type ImageTypeEnum
7   * (THRESHOLD: Binary values,
8   * RASTERMONO: dithering,
9   * POLYMERIZATION: gather)
10  * @param compressType Int Compression Type (
11  * 0: Uncompressed,
12  * 1: Monolithic Compression,
13  * 2: Subpackaged Compression)
14  * @param light Int brightness
15  * @return Boolean Whether to send or not
16  */
17  fun printBitmapCPCL(bitmap: Bitmap, x: Int, y: Int,
18                      type: ImageTypeEnum, compressType: Int, light: Int): Boolean

```

- Examples

#### 打印图片示例

```

1  cpclHelper?.printAreaSize("0","200","200", bitmapPrint.height.toString(),
2  "1")
3  cpclHelper?.printBitmapCPCL(bitmapPrint,0,0,
4  if (type == 0) ImageTypeEnum.THRESHOLD
5  else if(type == 1) ImageTypeEnum.RASTERMONO
6  else ImageTypeEnum.POLYMERIZATION, compressType
7  , 0)
8  cpclHelper?.print()

```

## 3.2.18 Get Printer Status

- Description

this interface is used to obtain the printer status

▼ getPrinterStatus

Java |

```

1  /**
2   * Get Printer Status
3   * @return Int
4   */
5  fun getPrinterStatus(): Int
6
7  val statusStr = if (printerStatus == 0) {
8      //normal
9  } else if (printerStatus?.and(2) == 2) {
10     //Lack of paper
11 } else if (printerStatus?.and( 4) == 4) {
12     //Open the lid
13 } else {
14     //Other abnormalities
15 }
```

- Examples



Kotlin |

```

1  val printerStatus = cpcHelper?.getPrinterStatus()
```

## 3.2.19 Send Data

- Description

the interface can send data directly to the printer



Java |

```

1  /**
2   * send data
3   * @param data ByteArray Data that needs to be sent
4   * @return Boolean Whether to send or not
5   */
6  fun writeData(data: ByteArray): Boolean
```

- Examples



Java |

```

1  cpcHelper?.writeData(byteArrayOf(0x0d,0x0a))
```

### 3.2.20 Read Data

- **Description**

this interface can directly read the data returned by the printer

```
1  /**
2   * Read Data
3   * @param timeout Long Timeout Period (ms)
4   * @return ByteArray? The data returned by the printer
5   */
6  fun readData(timeout: Long): ByteArray?
```

- **Examples**

```
1  var readData = cpclHelper?.readData(2000)
```

### 3.2.21 Leak-proof function

- **description**

this function can be used to prevent the printer from missing printing. First, you need to call the interface that enables this function, then set the task ID, send the print data, and finally obtain the print result.

```
1  /** Set the missed hit function switch
2   * @param isOpen
3   * @return Boolean Whether the message is successful
4   */
5  public boolean setRePrintSwitch(boolean isOpen)
6  //Note that the interface is in the PrinterCommonHelper class
```

```

1  /**
2   * Set the print job ID
3   * @param id Int (0-65535) The ID of the task
4   * @return Boolean
5   */
6  fun setTaskID(id: Int): Boolean

```

- **Note**

There are two interfaces to get printer results (one with ID and one without ID, depending on the protocol supported by the printer)

- ▼ ID

```

1  /**
2   * Get the print result
3   * @param timeout Long Timeout Period (ms)
4   * @return TaskResult
5   */
6  fun getPrintResult(timeout: Long): TaskResult
7  /**
8   * id: The ID of the task
9   * status: Printing Status (-1: Get Timeout, 0: Print Successful,
10  *      1: Out of Paper, 2: Open Cap, 3: Other Exceptions)
11  */
12  data class TaskResult(val id: Int, val status: Int)

```

- ▼ without ID

```

1  /**
2   * Get the print result
3   * @param timeout Long Timeout Period (ms)
4   * @return -1: Get Timeout, 0: Print Successful,
5   *      1: Out of Paper, 2: Open Cap, 3: Other Exceptions
6   */
7  fun getPrintResultNullID(timeout: Long): Int

```

- **Examples**

ID	Kotlin
<pre> 1  commandHelper?.setRePrintSwitch(true) 2  cpclHelper?.setTaskID(1) 3  cpclHelper?.cleanReadBuffer()//Clear the printer read buffer data 4  samplePrint(cpclHelper, false) 5  val printResult = cpclHelper?.getPrintResult(5000) </pre>	
without ID	Kotlin
<pre> 1  commandHelper?.setRePrintSwitch(true) 2  cpclHelper?.cleanReadBuffer()//Clear the printer read buffer data 3  samplePrint(cpclHelper, false) 4  val printResult = cpclHelper?.getPrintResultNullID(5000) </pre>	

## 4.1.1 Coding Table

Name	SDK encoding
chinese	gb2312
western European	ISO8859-1
latin (2)	ISO8859-2
latin (3)	ISO8859-3
the language of polo	ISO8859-4
cyrillic	ISO8859-5
arabic	ISO8859-6
hebrew	ISO8859-8
turkish	ISO8859-9
latin (9)	ISO8859-15
greek (windows)	ISO8859-11
greek (ISO)	ISO8859-7
thai	windows-874

## 4.1.2 Font Direction

Value	description
CPCLTextEnum.TEXT	level
CPCLTextEnum.TEXT90	rotate 90 degrees counterclockwise
CPCLTextEnum.TEXT180	rotate 180 degrees counterclockwise
CPCLTextEnum.TEXT270	rotate 270 degrees counterclockwise

## 4.1.3 Font Size

Value	description
0	12x24.
1	12x 24 (print traditional Chinese in Chinese mode), font size becomes (9x 17) in English mode
2	8x 16.
3	20x20.
4	32x 32 or 16x 32, enlarged twice by ID3 font width and height.
7	24x 24 or 12x 24, depending on Chinese and English.
8	24x 24 or 12x 24, depending on Chinese and English.
20	16x 16 or 8x 16, depending on Chinese and English.
24	24x 24 or 12x 24, depending on Chinese and English.
55	16x 16 or 8x 16, depending on Chinese and English.

Other	the default is 24x 24 or 12x 24, depending on Chinese and English.
-------	--

#### 4.1.4 Barcode Type

bcType	name	length	data Range
UPCA	UPC-A	11-12	numbers
EAN8	JAN8/EAN8	7-8	numbers
39	CODE39	1-255	numbers, letters, some symbols
93	CODE93	1-255	numbers, letters, symbols
128	CODE128	2-255	numbers, letters, symbols (select the character set first, {A: ASCII characters 00h to 5FH {B: ASCII characters 20h to 7FH {C: 100 digits of 00-99)

#### 4.1.5 Proportion of wide and narrow strips

Value	proportion
0	1.5: 1
1	2.0:1
2	2.5:1
3	3.0:1
4	3.5:1
20	2.0:1

21	2.1:1
22	2.2:1
23	2.3:1
24	2.4:1
25	2.5:1
26	2.6:1
27	2.7:1
28	2.8:1
29	2.9:1
30	3.0:1

## 4.1.6 Enable the SDK to send data logs

- **description**

this function is used to record the data sent by the SDK to the printer and is used to determine whether the data is correct. The path to log is in

/sdcard/Android/data/app package name/files/Documents/SDK\_log.txt the app package name is the package name of your project. Opening method:

▼

Java |

```

1 Constants.isWriteLog = true
2 Constants.isHex = true

```