

PrinterSDK 使用说明书 (Windows / C++ 版)

本说明书对应 **HS-POS PrinterSDK Windows / C++ 版**，与 Android 1.6.3 API 对齐。

版本：**1.6.3-windows-cpp**
适用平台：**Windows 10 / 11, C++17 (CMake / MSVC 或 MinGW)**
命名空间：`hspos`
最后更新：**2026-07-18**
对应 Android 版本：`com.hspos.printerSDK 1.6.3` (`PrinterSDK_使用说明书.pdf`)
配套文档：`PrinterSDK_API_说明文档.md`、`PrinterSDK_Windows_CPP_API对照表.md`

重要 (C++)：`FeedPaper(connection, n)` 的 `n` 为 **点 (ESC J)**；走纸 **行** 请使用 `PrintAndFeedLines`。票据收尾推荐 `FeedAndCutPaper`。

目录

- 1. 产品简介
- 2. 系统要求与集成方式
- 3. SDK 架构说明
- 4. 快速开始
- 5. Windows 运行环境与权限说明
- 6. 打印机连接
- 7. 标准 POS 打印
- 8. 页模式打印
- 9. 标签模式打印
- 10. 状态查询与设备控制
- 11. 常量参考
- 12. 完整示例
- 13. 常见问题与排查
- 14. API 速查表
- 15. 与 Android / C# 版的平台差异说明

1. 产品简介

PrinterSDK (Windows 版) 是 HS-POS PrinterSDK 在 Windows / 原生 C++ 平台上的实现，与 Android 版 `com.hspos.printerSDK 1.6.3` API 对齐，支持 ESC/POS 标准小票打印和 JPL 标签打印。通过统一的 `PrinterSdk` 入口类，开发者可以用同一套 API 完成设备发现、连接、打印与状态查询。

主要特性

特性	说明
多通道连接	USB、蓝牙、网络 (TCP/IP)、串口，以及 Windows 独有的并口 (LPT)
标准小票打印	文本、条码、二维码、PDF417、图片、切纸、钱箱、蜂鸣
页模式	坐标定位绘制文本、条码、图片、矩形
标签模式	标签纸专用布局与 JPL 指令
多编码支持	UTF-8、GBK 便捷 API；其它编码可通过 <code>SetDoubleByteEncoding</code> + 自编码数据 / <code>SendCommand</code>
连接管理	无需 Android 风格的运行时权限申请；USB/串口/并口依赖 Windows 驱动与设备管理器
网络连接 (同步 TCP；异步 API 待补齐)	当前提供同步 <code>ConnectNetworkPrinter</code> ；UI 场景请自行放到工作线程

对外核心类型

集成后，业务代码通常只需关注以下类型：

```
#include "hspos/HsposPrinterSdk.hpp"
using namespace hspos;
```

- `PrinterSdk`：统一入口类（单例，`PrinterSdk::GetInstance()`）
- `PrinterConstants`：常量定义（对齐 Android `PrinterConstants`，命名为 `PascalCase`（与 C# 一致））
- `IPrinterConnection`：打印机连接句柄接口（对齐 Android `PrinterConnection`）

与 **Android** 的命名对照：`PrinterSDK` → `PrinterSdk`；`PrinterConnection` → `IPrinterConnection`；`com.hspos.printerSDK` → `hspos`；`camelCase` 方法 → `PascalCase` 方法（如 `printText` → `PrintText`）。完整对照见 `PrinterSDK_Windows_CPP_API对照表.md`。

2. 系统要求与集成方式

2.1 环境要求

项目	要求
操作系统	Windows 10 / Windows 11 (x64)
语言标准	C++17
构建工具	CMake 3.20+
编译器	MSVC (Visual Studio 2022) 或 MinGW-w64 (MSYS2 UCRT64)
依赖	Windows API (串口/USB/蓝牙/并口)；图像解码使用 WIC (windowscodecs)

2.2 集成方式一：CMake 链接静态库

```
add_subdirectory(path/to/PrinterAPI_CPP) # 或 find_package 后
target_link_libraries(your_app PRIVATE hspos_printer_sdk)
target_include_directories(your_app PRIVATE path/to/PrinterAPI_CPP/include)
```

MinGW 本地构建示例：

```
$env:PATH = "C:\msys64\ucrt64\bin;" + $env:PATH
cd "C:\Windows Project\PrinterAPI_CPP"
cmake -B build -G "MinGW Makefiles" -DCMAKE_BUILD_TYPE=Release -DHSPS_BUILD_SHARED=OFF
cmake --build build -j
```

产物：`build\libHspos.PrinterSdk.a`（或 `SHARED` 时的 `.dll` / `.lib`）、`win_demo.exe`、`console_demo.exe`。

2.3 集成方式二：源码一起编译

将 `include/hspos` 加入头文件搜索路径，并把 `src/*.cpp` 与 `src/communication/*.cpp` 编入工程，链接 `windowscodecs`、`ole32`、`ws2_32`、`setupapi`、`bthprops` 等 Windows 库（以 `CMakeLists.txt` 为准）。

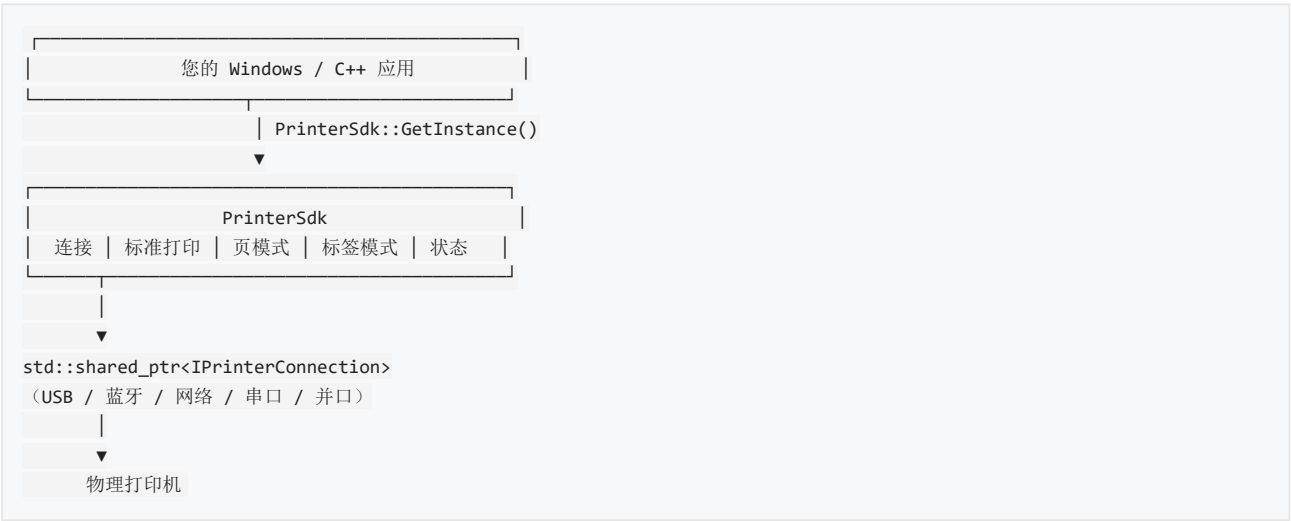
2.4 源码模块结构

PrinterAPI_CPP/

- ├─ CMakeLists.txt
- ├─ include/hspos/ ← 对外头文件
 - ├─ HsposPrinterSdk.hpp ← 一键包含入口
 - ├─ PrinterSdk.hpp ← 统一门面
 - ├─ PrinterConstants.hpp
 - ├─ PrinterCommands.hpp
 - ├─ JplCommands.hpp
 - ├─ ImageUtils.hpp
 - └─ communication/ ← 连接抽象与实现
- ├─ src/ ← 实现
- ├─ samples/
 - ├─ console_demo/ ← 控制台样例
 - └─ win_demo/ ← 窗体测试工具
- └─ docs/ ← 本说明书与 API 文档

建议：业务层统一调用 `PrinterSdk`，不要直接拼装底层指令类，以便升级兼容。

3. SDK 架构说明



连接对象使用 `std::shared_ptr<IPrinterConnection>`。打印前请确保 `connection && connection->IsConnected()`。

4. 快速开始

4.1 初始化

Windows C++ 版无需 `Android Context`，也无需权限回调转发：

```
#include "hspos/HsposPrinterSdk.hpp"
using namespace hspos;

int main() {
    auto& sdk = PrinterSdk::GetInstance();
    // ... 连接与打印 ...
    sdk.Release();
    return 0;
}
```

4.2 最小打印流程

```
#include "hspos/HsposPrinterSdk.hpp"
using namespace hspos;

auto& sdk = PrinterSdk::GetInstance();
auto conn = sdk.ConnectNetworkPrinter("192.168.1.100", 9100);
if (!conn || !conn->IsConnected()) {
    // 连接失败
    return;
}

sdk.InitPrinter(conn);
sdk.SetAlignment(conn, PrinterConstants::AlignmentCenter);
sdk.PrintText(conn, "Hello PrinterSdk\n");
sdk.FeedAndCutPaper(conn);
sdk.DisconnectPrinter(conn);
```

中文小票推荐：

```
sdk.InitPrinter(conn);
sdk.SetChineseMode(conn);
sdk.SetDoubleByteEncoding(conn, PrinterConstants::DoubleByteEncodingGbk);
sdk.PrintTextInGbk(conn, "你好，打印机");
sdk.FeedAndCutPaper(conn);
```

5. Windows 运行环境与权限说明

Windows 版没有 Android 风格的运行时权限（Runtime Permission）模型，因此不需要：

- 在应用清单中声明 BLUETOOTH / USB_PERMISSION / ACCESS_FINE_LOCATION 等权限；
- 在窗体/Activity 中转发 onRequestPermissionsResult ；
- 处理 USB / 蓝牙权限弹窗回调。

取而代之，Windows 上的设备访问权限由操作系统驱动与设备管理器管理。以下是各连接方式在 Windows 上的运行环境要求：

5.1 串口（Serial / COM）

项目	要求
驱动	打印机厂商提供的串口驱动，或 USB-转串口芯片（CH340/CP2102 等）驱动
端口确认	「设备管理器」→「端口（COM 和 LPT）」中确认端口号，如 COM3
参数匹配	波特率、校验位、数据位、停止位、流控须与打印机设置一致（常见 9600 / 115200，NOPARITY，No FlowControl）
权限	一般无需管理员权限；若端口被其他程序占用会导致打开失败

5.2 并口（Parallel / LPT，Windows 额外能力）

Android 平台不支持并口；Windows 版新增了对传统并口打印机/工控设备的支持：

项目	要求
硬件	需要物理 LPT 接口或虚拟并口驱动；多数现代消费级 PC 已不带物理并口
端口	通常为 LPT1 – LPT3
权限	一般无需管理员权限；部分驱动下写入需要独占端口
限制	并口通常只支持写出，状态查询/读回可能不可用，属预期行为

5.3 USB

项目	要求
驱动	Windows 自带的「USB 打印支持」驱动，或厂商驱动
确认方式	「设备管理器」中能看到打印机，或在 SDK 中通过 <code>GetUsbDevices()</code> 枚举到对应 VID/PID
权限	Windows 无 USB 运行时权限模型，不存在 Android 的 <code>hasUsbPermission</code> / <code>requestUsbPermission</code> 概念；只要驱动安装正确即可直接连接
常见问题	若连接失败，检查是否被其他打印软件（如系统打印队列、厂商工具）占用该 USB 打印口

5.4 蓝牙

项目	要求
硬件	本机需具备蓝牙适配器并已启用
配对	首次连接建议先在「Windows 设置 → 蓝牙和其他设备」中配对，或调用 <code>PairBluetoothDevice</code> 由 SDK 触发系统配对
权限	无需声明位置权限（Windows 蓝牙扫描不要求位置权限）；系统可能弹出配对确认框，需要用户交互确认
扫描耗时	<code>DiscoverBluetoothDevices</code> / <code>RefreshBluetoothDevices</code> 为阻塞式扫描，建议在后台线程调用，避免卡住 UI

5.5 网络 (TCP/IP)

项目	要求
网络	PC 与打印机需在同一网段可达
端口	默认 9100（RAW 打印端口），需确认打印机及网络防火墙未拦截
权限	无需任何特殊权限；如系统防火墙拦截出站连接，需要在防火墙规则中放行

5.6 与 Android 权限模型对照小结

Android 概念	Windows 对应
<code>AndroidManifest.xml</code> 权限声明	不需要；改为驱动安装 + 设备管理器确认
<code>BLUETOOTH_CONNECT</code> / <code>BLUETOOTH_SCAN</code> + 位置权限	不需要；系统级蓝牙配对确认框
<code>USB_PERMISSION</code> 运行时申请	不需要；USB 驱动装好即可访问
<code>onRequestPermissionsResult</code> 回调转发	不存在，无需实现
USB 设备过滤器 <code>device_filter.xml</code>	不适用；改为在代码中按 <code>VendorId</code> / <code>ProductId</code> 过滤 <code>GetUsbDevices()</code> 返回列表

6. 打印机连接

所有连接方法返回 `std::shared_ptr<IPrinterConnection>` 对象。连接失败时返回 `nullptr`。

6.1 USB 连接

```
auto devices = sdk.GetUsbDevices();
if (devices.size() > 0)
{
    auto conn = sdk.ConnectUsbPrinter(devices[0]);
}

// 也可按端口名/设备路径直连
shared_ptr<IPrinterConnection> conn2 = sdk.ConnectUsbPrinter("USB001");
```

- `GetUsbDevices()` 返回 `UsbPrinterInfo` 列表, 包含 `PortName`、`DevicePath`、`VendorId`、`ProductId`、`VidPidText` (形如 `VID:0x0FE6,PID:0x811E`) 等信息, 便于按厂商设备过滤。
- Windows 无需申请 USB 权限, 不存在 `HasUsbPermission` / `RequestUsbPermission` (详见 第 15 节)。

6.2 蓝牙连接

```
// 获取已配对设备
auto paired = sdk.GetBluetoothPairedDevices();

// 获取蓝牙设备 (已配对 + 已扫描缓存)
auto devices = sdk.GetBluetoothDevices();

// 获取蓝牙打印机列表 (内部别名, 等价于 GetBluetoothDevices)
var printers = sdk.GetBluetoothPrinters();

// 方式一: 注册监听器 + 后台扫描 (贴近 Android 用法)
sdk.AddBluetoothListener(new MyBluetoothListener());
sdk.StartBluetoothDiscovery();
// ... 等待若干秒后
auto all = sdk.GetBluetoothPairedDevices(); // 或 DiscoverBluetoothDevices
sdk.StopBluetoothDiscovery();

// 方式二: Windows 额外提供的同步扫描 (简单场景更方便, 注意耗时, 建议放在后台线程)
auto discovered = sdk.DiscoverBluetoothDevices(true);

// 连接 (未配对设备可让 SDK 自动触发系统配对)
auto conn = sdk.ConnectBluetoothPrinter(paired[0]);
```

C++ 版当前不提供 Android 风格的蓝牙监听器回调; 请使用 `DiscoverBluetoothDevices` / `GetBluetoothPairedDevices` 主动获取列表。蓝牙设备需先在系统设置中配对, 或通过 `PairBluetoothDevice(address, pin)` / `ConnectBluetoothPrinter(device, true)` 触发系统配对流程。

6.3 网络连接

当前提供同步连接。UI 场景请放到工作线程调用。

```
auto conn = sdk.ConnectNetworkPrinter("192.168.1.100", 9100);
if (!conn || !conn->IsConnected()) {
    // 连接失败
    return;
}
// 在工作线程中执行打印, 避免阻塞 UI
```

参数	说明
IP 地址	打印机局域网 IP, 如 192.168.1.100
端口	默认 9100 (RAW 打印端口)

6.4 串口连接

```
auto ports = PrinterSdk::ListSerialPorts();
// 常见端口: COM1、COM3、COM5 (以设备管理器实际显示为准)

// 简化方式: 端口名 + 波特率
auto conn = sdk.ConnectSerialPrinter("COM3", 9600);

// 完整方式: 自定义校验位 / 流控
SerialPortSettings settings;
settings.portName = "COM3";
// ...
auto conn2 = sdk.ConnectSerialPrinter(settings);
/* 原 C# 对象初始化等价字段:
    PortName = "COM3",
    BaudRate = 9600,
    Parity = System.IO.Ports.Parity.None,
    DataBits = 8,
    StopBits = System.IO.Ports.StopBits.One,
    Handshake = System.IO.Ports.Handshake.None
  });
```

常见波特率: 9600、19200、38400、57600、115200。

6.5 并口连接 (Windows 额外能力)

Android 平台不支持并口; 如果客户设备为传统工控一体机、老式针式/热敏打印机, Windows 版提供并口支持:

```
var lptPorts = sdk.GetParallelDevices();
// 常见端口: LPT1、LPT2、LPT3

auto conn = sdk.ConnectParallelPrinter("LPT1");
```

并口通常只支持写出, 多数机型不支持状态回读, `QueryRtStatus / GetPrinterStatus` 在并口连接下可能返回 `nullptr / StatusOffline`, 属预期行为。

6.6 断开连接与通信辅助

```
// 断开连接
sdk.DisconnectPrinter(connection);

// 释放 SDK 资源
sdk.Release();
```

方法	说明
<code>DisconnectPrinter(connection)</code>	断开打印机连接
<code>Release()</code>	释放 SDK 资源

Windows 版不提供 `GetCommunicationManager()`。
`ClearBuffer / CancelPrint` 当前 C++ 版暂未封装, 可用 `SendCommand` 发送 `CAN` (0x18) 等指令。

7. 标准 POS 打印

以下所有方法第一个参数均为 `auto connection`, 返回 `bool` 表示指令是否成功写入端口 (不代表打印机机械动作已完成)。

7.1 初始化与重置

方法	说明
InitPrinter(connection)	初始化打印机 (ESC @)
ResetPrinter(connection)	重置打印机
SendSelfTest(connection)	发送自检命令
SendCommand(connection, byte[])	发送原始指令

7.2 文本打印

方法	编码
PrintText(connection, text)	默认编码
PrintTextInUtf8(connection, text)	UTF-8
PrintTextInGbk(connection, text)	GBK (简体中文)
PrintTextInBig5(connection, text) (别名 PrintTextInBIG5 , 已标记过时)	BIG5 (繁体中文)
PrintTextInShiftJis(connection, text)	Shift-JIS (日文)
PrintTextInEucKr(connection, text)	EUC-KR (韩文)

```
sdk.InitPrinter(connection);
sdk.SetAlignment(connection, PrinterConstants::AlignmentCenter);
sdk.SetChineseDoubleSize(connection, true); // 汉字倍宽倍高 FS W
sdk.SetBold(connection, true);
sdk.PrintTextInGbk(connection, "欢迎光临\n");
sdk.SetBold(connection, false);
sdk.SetChineseDoubleSize(connection, false); // 恢复汉字正常尺寸
```

左右分列一行 (80mm 小票常用)

方法	说明
PrintTextLineLeftRight(connection, left, right)	80mm 默认宽度 576 点、字符宽 12 点
PrintTextLineLeftRight(connection, left, right, areaWidthDots, charWidthDots)	自定义区域宽与字符宽

纸宽常量 (PrinterConstants) :

常量	值	说明
PaperWidth58Mm	384	58mm 纸有效宽度 (点)
PaperWidth80Mm	576	80mm 纸有效宽度 (点)

```
// 80mm 商业小票：左商品名、右价格
sdk.InitPrinter(connection);
sdk.SetPrintAreaWidth(connection, PrinterConstants::PaperWidth80Mm);
sdk.SetPrintAreaLeftMargin(connection, 0);
sdk.SetAlignment(connection, PrinterConstants::AlignmentLeft);

sdk.PrintTextLineLeftRight(connection, "Lorem ipsum", "$1.25");
sdk.PrintTextLineLeftRight(connection, "Sub Total", "$150.70");

// 合计行放大时，字符宽改为 24（GS！倍宽倍高后）
sdk.SetBold(connection, true);
sdk.SetTextScale(connection, 1, 1);
sdk.PrintTextLineLeftRight(connection, "TOTAL", "$155.99",
    PrinterConstants::PaperWidth80Mm, 24);
sdk.SetTextScale(connection, 0, 0);
sdk.SetBold(connection, false);
```

实现原理：ESC \$ 定位到行首打印左列，再 ESC \$ 定位到右侧打印右列并换行。倍宽倍高后请同步增大 charWidthDots（通常 12→24）。

7.2.1 字体倍宽倍高（三种指令，勿混淆）

打印机有 **三套不同的倍宽倍高指令**，适用场景不同：

方法	指令	码	适用
SetChineseDoubleSize()	FS W n	1C 57 n	汉字倍宽倍高（开/关）
SetTextScale()	GS ! n	1D 21 n	中外文字符尺寸（0-7 档）
SetPrintMode()	ESC ! n	1B 21 n	打印模式倍宽/倍高

打印机会按 **最后收到的指令** 生效；混用时请注意顺序。

① 汉字倍宽倍高 — SetChineseDoubleSize()（推荐打中文）

```
// 开启汉字倍宽倍高 → 1C 57 01
sdk.SetChineseDoubleSize(connection, true);
// 或
sdk.SetChineseDoubleSize(connection, PrinterConstants::ChineseDoubleSizeOn);

sdk.PrintTextInGbk(connection, "爱尚自身\n");

// 取消，恢复汉字正常尺寸 → 1C 57 00
sdk.SetChineseDoubleSize(connection, false);
// 或传入 int（仅最低位有效：0/1）
sdk.SetChineseDoubleSize(connection, PrinterConstants::ChineseDoubleSizeNormal);
```

参数 n 仅 **最低位** 有效：0 =取消，1 =汉字倍宽倍高同时开启。

② 字符尺寸 — SetTextScale()（GS！n）

```
sdk.SetTextScale(connection, 0, 0); // 正常
sdk.SetTextScale(connection, 1, 1); // 2倍宽 + 2倍高
sdk.SetTextScale(connection, 1, 0); // 仅2倍宽
sdk.SetTextScale(connection, 0, 1); // 仅2倍高
```

③ 打印模式 — SetPrintMode()（ESC！n）

```
sdk.SetPrintMode(connection, PrinterConstants::PrintModeDoubleWidth);
sdk.SetPrintMode(connection, PrinterConstants::PrintModeDoubleHeight);
sdk.SetPrintMode(connection, PrinterConstants::PrintModeDouble);
sdk.SetPrintMode(connection, PrinterConstants::PrintModeNormal);
```

7.3 格式设置

方法	说明
SetAlignment(connection, alignment)	对齐: 左/中/右
SetChineseDoubleSize(connection, enable)	汉字倍宽倍高 (FS W n, bool)
SetChineseDoubleSize(connection, n)	汉字倍宽倍高 (FS W n, int , 0/1)
SetTextScale(connection, widthScale, heightScale)	字符尺寸 GS ! n (0=正常, 1=2倍 ...)
SetBold(connection, bold)	加粗
SetUnderline(connection, underline)	下划线
SetReverse(connection, reverse)	反白
SetLineSpacing(connection, spacing)	行间距
SetCharSpacing(connection, spacing)	字符间距
SetPrintMode(connection, mode)	打印模式 ESC ! n (含倍宽/倍高)
SetTextUpsideDown(connection, upsideDown)	上下倒置
SetTextRotate(connection, rotate)	旋转 90°
SetTextLineHeight(connection, lineHeight)	行高
SetPrintPosition(connection, position)	打印起始位置
SetPrintAreaLeftMargin(connection, margin)	左边距
SetPrintAreaWidth(connection, width)	打印区域宽度
SetHorizontalAbsolutePrintPosition(connection, pos)	水平绝对位置
SetHorizontalRelativePrintPosition(connection, offset)	水平相对偏移
SetVerticalAbsolutePrintPosition(connection, pos)	垂直绝对位置
SetVerticalRelativePrintPosition(connection, offset)	垂直相对偏移
SetSingleByteMode(connection)	单字节模式
SetCharacterSet(connection, charset)	字符集
SetCharacterCodepage(connection, codepage)	代码页
SetChineseMode(connection)	中文模式
SetDoubleByteEncoding(connection, n)	双字节编码
SetAsciiTextFontType(connection, fontType)	ASCII 字体 A/B
SetAsciiTextCharRightSpacing(connection, spacing)	ASCII 右间距
SetKanjiTextCharSpacing(connection, left, right)	汉字间距

7.4 条码与二维码

如何选择条码 TYPE (重要)

PrintBarcode() 的第二个参数 type 必须传入 `PrinterConstants::BarcodeType*` 常量, 不要手写 0、1、2 等数字, 也不要与标签模式的条码类型编号 (0-29) 混用。

测试时不知道选哪个? 优先使用 **Code 128**——兼容性最好, 支持字母和数字, 适合订单号、物流单号等大多数场景:

```
// 推荐：首次测试条码功能时使用此组合
sdk.PrintBarcode(connection,
    PrinterConstants::BarcodeTypeCode128, // ← TYPE 选这个
    "1234567890", // 任意字母数字均可
    PrinterConstants::BarcodeHriBelow,
    80, 2);
```

按业务场景选择：

业务场景	选用常量	示例数据	数据要求
通用测试 / 订单号 / 物流单号	BarcodeTypeCode128	"1234567890"	字母、数字、符号均可
北美零售商品 (UPC)	BarcodeTypeUpcA	"012345678905"	11-12 位数字
国际零售商品 (EAN)	BarcodeTypeEan13	"6901234567892"	12-13 位数字
小包装商品	BarcodeTypeEan8	"96385074"	7-8 位数字
工业 / 仓储标签	BarcodeTypeCode39	"ABC-1234"	数字、大写字母及 -. \$/ +%
箱码 (ITF)	BarcodeTypeItf	"123456789012"	必须为偶数位数字
图书馆 / 血库等	BarcodeTypeCodabar	"A40156B"	以 A-D 开头和结尾
北美零售压缩码	BarcodeTypeUpcE	"01234565"	需符合 UPC-E 编码规则
高密度编码	BarcodeTypeCode93	"TEST93"	字母数字

注意：标准小票 (ESC/POS) 与标签模式 (JPL) 的条码 type 不是同一套编号。标准打印用 BarcodeType* (值为 0x41 - 0x49)；标签打印用 LabelBarcodeType* (值为 0-29)，详见第 9 节。

```
// 一维条码
sdk.PrintBarcode(connection,
    PrinterConstants::BarcodeTypeCode128, // 条码类型
    "1234567890", // 数据
    PrinterConstants::BarcodeHriBelow, // 文字位置
    80, // 高度 (点)
    2); // 宽度

// 二维码 (简易)
sdk.PrintQrCode(connection, "https://example.com", 3);

// 二维码 (标准, 可指定对齐和纠错级别)
sdk.PrintQrCodeStandard(connection,
    "https://example.com", 3,
    PrinterConstants::AlignmentCenter,
    PrinterConstants::QrEccLevelM);

// 双二维码
sdk.PrintDoubleQrCode(connection, n, qr1Pos, qr1Data, qr1Ecc, qr1Ver,
    qr2Pos, qr2Data, qr2Ecc, qr2Ver);

// PDF417
sdk.PrintPdf417(connection, columns, rows, moduleWidth, rowHeight,
    errorCorrectionLevel, option, data);
```

条码格式预设置 (可选, 在 PrintBarcode 之前调用)：

方法	说明
<code>SetBarcodeHriPosition(connection, pos)</code>	HRI 文字位置（条码下方/上方等）
<code>SetBarcodeHeight(connection, height)</code>	条码高度（点）
<code>SetBarcodeWidth(connection, width)</code>	条码窄条宽度
<code>SetBarcodeReadableTextFontType(connection, fontType)</code>	HRI 文字字体
<code>SetBarcodeReadableTextPosition(connection, position)</code>	HRI 文字相对位置

```
sdk.SetBarcodeHriPosition(connection, PrinterConstants::BarcodeHriBelow);
sdk.SetBarcodeHeight(connection, 80);
sdk.SetBarcodeWidth(connection, 2);
sdk.PrintBarcode(connection, PrinterConstants::BarcodeTypeCode128, "1234567890",
    PrinterConstants::BarcodeHriBelow, 80, 2);
```

条码类型常量：

常量	类型	ESC/POS m 值	适用场景
<code>BarcodeTypeUpcA</code>	UPC-A	<code>0x41</code> (65)	北美 12 位商品码
<code>BarcodeTypeUpcE</code>	UPC-E	<code>0x42</code> (66)	北美 8 位压缩商品码
<code>BarcodeTypeEan13</code>	EAN-13	<code>0x43</code> (67)	国际 13 位商品码
<code>BarcodeTypeEan8</code>	EAN-8	<code>0x44</code> (68)	国际 8 位商品码
<code>BarcodeTypeCode39</code>	Code 39	<code>0x45</code> (69)	工业标识、物流
<code>BarcodeTypeItf</code>	ITF	<code>0x46</code> (70)	箱码（偶数位数字）
<code>BarcodeTypeCodabar</code>	Codabar	<code>0x47</code> (71)	图书馆、血库
<code>BarcodeTypeCode93</code>	Code 93	<code>0x48</code> (72)	高密度编码
<code>BarcodeTypeCode128</code>	Code 128	<code>0x49</code> (73)	通用首选 ，测试推荐

HRI 文字位置：

常量	说明
<code>BarcodeHriNone</code>	不打印条码下方/上方文字
<code>BarcodeHriBelow</code>	在条码下方打印可读文字（常用）

7.5 图片打印

`PrintRasterImageFrom*` 会将图片转为黑白点阵后下发。默认使用 **AUTO 压缩**（在 `GS v 0` 与 `JBIG` 间自动择优，传输更少、打印更快）。

与 Android 的差异：Android 版 `printRasterImageFromFile/FromData/FromPixels` 各提供 **3 个重载**（无尺寸、指定尺寸、指定尺寸+压缩模式）。Windows 版目前各提供 **2 个重载**：无尺寸（内部使用 58mm 纸宽 + AUTO 压缩的默认值）与「指定尺寸 + 压缩模式」（`mode` 为必填参数，非可选重载）。业务层如只需默认压缩，直接传 `PrinterConstants::RasterCompressionAuto` 即可，效果与 Android 一致。

```
// 从文件（默认参数：58mm 纸宽、AUTO 压缩）
sdk.PrintRasterImageFromFile(connection, @"C:\images\logo.png");

// 从文件（自定义尺寸 + 压缩模式，推荐显式传入）
sdk.PrintRasterImageFromFile(connection, @"C:\images\logo.png", 384, 200,
    PrinterConstants::RasterCompressionAuto);

// 从字节数组
sdk.PrintRasterImageFromData(connection, imageBytes);
sdk.PrintRasterImageFromData(connection, imageBytes, 384, 200,
    PrinterConstants::RasterCompressionJbig); // 强制 JBIG

// 从像素数组（ARGB int[]）
int[] pixels = /* ... */;
sdk.PrintRasterImageFromPixels(connection, pixels, width, height);
sdk.PrintRasterImageFromPixels(connection, pixels, width, height, 384, 200,
    PrinterConstants::RasterCompressionGsV0); // 强制标准 GS v 0
```

压缩模式常量（PrinterConstants::RasterCompression*）：

常量	值	指令	说明
RasterCompressionGsV0	0	GS v 0 (1D 76 30)	标准光栅，兼容性最好
RasterCompressionLineRle	1	US (P (1F 28 50)	逐行 RLE；需固件支持，多数机型不兼容
RasterCompressionJbig	2	US (J (1F 28 4A)	JBIG 压缩，体积更小、速度更快
RasterCompressionAuto	3	自动	默认；在 GS v 0 与 JBIG 间选传输量更小者

- 图片会自动转为黑白点阵。宽度建议不超过打印机有效打印宽度（58mm 纸约 384 点，80mm 纸约 576 点）。
- targetHeight 传 0 表示按宽度等比缩放。
- 日常业务使用默认 AUTO 即可；若 JBIG 打印异常，可改为 RasterCompressionGsV0。

7.6 走纸与切纸

方法	说明
FeedLine(connection)	换一行 (LF)
FeedPaper(connection, dots)	走纸 N 点 (ESC J; 注意不是行)
PrintAndFeedLines(connection, lines)	走纸 N 行 (ESC d)
FeedAndCutPaper(connection)	走纸越过切刀后半切（推荐收尾）
CutPaper(connection, cutType)	切纸 (CutFull = ESC i / CutPartial = ESC m)

切刀通常在打印头上方约 10–20mm。内容打完后请用 FeedAndCutPaper 。
制表 / 反向走纸 / 水平线等 API 当前 C++ 版尚未提供，可用 SendCommand 扩展。

```
sdk.PrintAndFeedLines(connection, 4); // 走纸 4 行
sdk.FeedPaper(connection, 100); // 再走纸 100 点
sdk.CutPaper(connection, PrinterConstants::CutPartial); // 半切
// 或一步完成：
sdk.FeedAndCutPaper(connection);
```

7.7 外设控制

```
// 打开钱箱
sdk.OpenCashDrawer(connection, PrinterConstants::CashDrawerPin2);
sdk.OpenCashDrawer(connection, PrinterConstants::CashDrawerPin5);

// 蜂鸣
sdk.Beep(connection);           // 默认 1 次
sdk.Beep(connection, 3, 2);     // 3 次，每次约 100ms
```

7.8 缓冲区控制

当前 C++ 版暂未提供 `ClearBuffer` / `CancelPrint` 封装；如需可使用 `SendCommand` 发送 `CAN` (0x18) 等指令。

8. 页模式打印

页模式 (Page Mode) 允许在指定区域内按坐标绘制内容，适合制作复杂票据版式。

8.1 基本流程

```
sdk.InitPrinter(connection);

// 1. 进入页模式（自动完成 GS P + ESC W + ESC T 初始化）
sdk.SelectPageMode(connection);
// 或自定义打印区域（80mm 纸宽约 576 点）
sdk.SelectPageModeEx(connection, 0, 0, 576, 800);

// 2. 可选：调整区域或方向
sdk.SetPrintAreaInPageMode(connection, 0, 0, 576, 800);
sdk.SetPrintAreaDirection(connection, PrinterConstants::PageDirectionLeftToRight);

// 3. 绘制内容（坐标单位为点，原点在打印区域左上角）
sdk.DrawTextGbkInPageMode(connection, 100, 100, "商品名称");
sdk.DrawBarcodeInPageMode(connection, 100, 200, PrinterConstants::BarcodeTypeCode128, "123456");
sdk.DrawQrCodeInPageMode(connection, 100, 300, 3, PrinterConstants::QrEccLevelM, "https://example.com");
sdk.DrawImageFromFileInPageMode(connection, 100, 400, @"C:\images\logo.png", 100, 50);
sdk.DrawRectangleBorderInPageMode(connection, 50, 50, 500, 100); // 边框
sdk.DrawFilledRectangleInPageMode(connection, 60, 60, 200, 80);   // 实心

// 4. 打印并退出
sdk.PagePrintPage(connection); // ESC FF，将页缓冲内容打印出来
sdk.ExitPageMode(connection);  // ESC S，返回标准模式
```

页模式排版辅助 (80mm 小票常用)

方法	说明
<code>EstimateTextWidthDots(text, charWidthDots)</code>	估算文本宽度 (Font A 正常约 12 点/字符)
<code>DrawTextCenteredInPageMode(connection, pageWidth, y, text)</code>	水平居中
<code>DrawTextRightInPageMode(connection, pageWidth, y, text)</code>	右对齐
<code>DrawTextLineLeftRightInPageMode(connection, pageWidth, y, left, right)</code>	一行左右分列

```
int w = PrinterConstants::PaperWidth80Mm; // 576
int y = 40;
sdk.DrawTextCenteredInPageMode(connection, w, y, "2847 Madison Avenue");
y += 36;
sdk.DrawTextLineLeftRightInPageMode(connection, w, y, "Bank Card", "**** * 1234");
```

NV 位图 / Logo (打印机内存, FS p)

打印机出厂或工具预存的 Logo 保存在 NV 存储器中，无需传图即可打印：

方法	说明
<code>PrintNvBitmap(connection, nvNumber, mode)</code>	标准模式打印 NV 位图
<code>DrawNvBitmapInPageMode(connection, x, y, nvNumber, mode)</code>	页模式指定坐标
<code>DrawNvBitmapCenteredInPageMode(connection, pageWidth, y, nvNumber, mode, imageWidthDots)</code>	页模式水平居中

mode 使用 `PrinterConstants::NvBitmapMode*` (0 正常 / 1 倍宽 / 2 倍高 / 3 倍宽倍高)。

```
// 页模式顶部居中 Logo (nvNumber、imageWidthDots 按打印机手册调整)
sdk.DrawNvBitmapCenteredInPageMode(connection, 576, 40, 1,
    PrinterConstants::NvBitmapModeNormal, 256);
y += 100; // 预留 Logo 高度
```

注意：页模式与标签模式 (JPL) 是不同指令体系，请勿混用。 `SelectPageMode()` / `SelectPageModeEx()` 会自动设置移动单位、打印区域和默认方向；若只调用 `SelectPageMode()` 而不设置区域，打印机无法正确定位内容。

8.2 页模式 API 列表

分类	方法
模式控制	<code>SelectPageMode</code> 、 <code>ExitPageMode</code> 、 <code>SelectPageModeEx</code> (3 个重载)
区域设置	<code>SetMovementUnits</code> 、 <code>PageSetPageArea</code> 、 <code>PageSetPageDrawDirection</code> 、 <code>SetPrintAreaInPageMode</code> 、 <code>SetPrintAreaDirection</code>
打印控制	<code>PagePrintPage</code> 、 <code>PageClearPage</code> (别名 <code>CancelPrintDataInPageMode</code>)
文本绘制	<code>DrawTextInPageMode</code> 、 <code>DrawTextUtf8InPageMode</code> 、 <code>DrawTextGbkInPageMode</code> 、 <code>DrawTextBIG5InPageMode</code> 、 <code>DrawTextShiftJISInPageMode</code> 、 <code>DrawTextEUCKRInPageMode</code>
页模式排版	<code>DrawTextCenteredInPageMode</code> 、 <code>DrawTextRightInPageMode</code> 、 <code>DrawTextLineLeftRightInPageMode</code> 、 <code>EstimateTextWidthDots</code> 、 <code>EstimateCode128WidthDots</code>
NV 位图	<code>PrintNvBitmap</code> 、 <code>DrawNvBitmapInPageMode</code> 、 <code>DrawNvBitmapCenteredInPageMode</code>
图形绘制	<code>DrawLineInPageMode</code> 、 <code>DrawRectangleInPageMode</code> 、 <code>DrawRectangleBorderInPageMode</code> 、 <code>DrawFilledRectangleInPageMode</code>
条码/码	<code>DrawBarcodeInPageMode</code> (2 个重载)、 <code>DrawBarcodeInPageModeWithCenteredHri</code> 、 <code>DrawQrCodeInPageMode</code>
图片	<code>DrawImageFromFileInPageMode</code> 、 <code>DrawImageFromDataInPageMode</code> 、 <code>DrawImageFromPixelsInPageMode</code>

9. 标签模式打印

标签模式适用于标签纸打印机，使用 JPL 指令集进行精确布局。

9.1 基本流程

```
// 1. 启用标签模式
sdk.EnableLabelMode(connection);

// 2. 开始标签页 (x, y, width, height, rotation)
sdk.LabelPageBegin(connection, 0, 0, 576, 400, 0);

// 3. 绘制内容
sdk.PrintLabelTextGbk(connection, "商品标签", 50, 50, 24, 0);
sdk.PrintLabelBarcode(connection,
    PrinterConstants::LabelBarcodeTypeCode128, "1234567890", 50, 100, 80, 2);
sdk.PrintLabelQrCode(connection, "https://example.com", 50, 200, 3);
sdk.DrawLabelImageFromFile(connection, 300, 50, @"C:\images\logo.png");
sdk.DrawLabelLine(connection, 50, 300, 500, 300);
sdk.DrawLabelRect(connection, 50, 320, 500, 380);

// 4. 打印
sdk.BackPaperToPrintPosition(connection);
sdk.LabelPagePrint(connection);           // 打印 1 份
sdk.LabelPagePrint(connection, 3);        // 打印 3 份

// 5. 退出标签模式
sdk.DisableLabelMode(connection);
```

说明：PrintLabelQrCode 简易重载的实际签名为 PrintLabelQrCode(connection, string data, int x, int y) (无 size 参数, 模块大小由内部默认值决定)；如需指定版本、纠错等级、模块宽度, 请使用完整参数重载 PrintLabelQrCode(connection, x, y, version, eccLevel, unitWidth, rotate, data)。

9.2 标签模式 API 列表

分类	方法
模式控制	EnableLabelMode、DisableLabelMode、LabelPageBegin、BackPaperToPrintPosition、LabelPagePrint (含份数重载)
文本	PrintLabelText、PrintLabelTextUtf8、PrintLabelTextGbk (均含完整参数重载)
条码	PrintLabelBarcode (2 个重载)、SetLabelBarcodeTextPosition
二维 码/PDF417	PrintLabelQrCode (2 个重载)、PrintLabelPdf417Code (2 个重载)
图片	DrawLabelImageFromFile、DrawLabelImageFromData、DrawLabelImageFromPixels、DrawLabelImageAdvanced
图形	DrawLabelLine (2 个重载)、DrawLabelRect (2 个重载)、DrawLabelBox

10. 状态查询与设备控制

SDK 提供 两种 状态获取方式, 适用场景不同:

方式	API	方向	SDK 是否解析	适用场景
便捷判断	GetPrinterStatus / IsPrinterOnline 等	应用主动查	是 (合成简化常量)	快速判断在线/缺纸/开盖
主动查询	QueryRtStatus / QueryTransmitStatus	应用主动查	否 (返回原始字节)	按 ESC/POS 标准自行解析

说明：除「便捷判断」外, SDK 不对状态字节做业务解析, 位含义请按下文对照表在业务层自行判断。不同机型可能有差异, 以实机为准。网口、串口连接下状态查询更可靠; 并口通常不支持状态回读。

10.1 SDK 便捷状态 API

```
int status      = sdk.GetPrinterStatus(connection);
bool online     = sdk.IsPrinterOnline(connection);
bool hasPaper   = sdk.HasPaper(connection);
bool coverOpen  = sdk.IsCoverOpen(connection);
```

GetPrinterStatus 内部依次查询 StatusTypePrinter（打印机状态）、StatusTypeOffline（脱机状态）、StatusTypePaperSensor（纸传感器状态），合成为下列简化常量的按位组合：

常量	值	含义
StatusOffline	0x00	查询失败或无响应（默认返回值）
StatusOnline	0x01	在线
StatusPaperOut	0x02	缺纸
StatusCoverOpen	0x04	开盖

上述常量为 SDK 封装用简化值，与 ESC/POS 标准状态字节的位定义不完全相同。精确判断请用 10.2 的原始字节按位解析。

10.2 主动查询（DLE EOT / GS r）

API

```
// DLE EOT: 10 04 n
byte[]? rt = sdk.QueryRtStatus(connection, type, timeoutMs);

// GS r: 1D 72 n（部分机型更兼容，返回格式与 DLE EOT 同类）
byte[]? tx = sdk.QueryTransmitStatus(connection, type, timeoutMs);
```

查询类型 type（PrinterConstants::StatusType*）

常量	n 值	说明	典型返回
StatusTypePrinter	1	打印机状态	1 字节
StatusTypeOffline	2	脱机/机械状态	1 字节
StatusTypeError	3	错误状态	1 字节
StatusTypePaperSensor	4	纸传感器状态	1 字节

timeoutMs 为等待响应毫秒数；无响应返回 nullptr。

n=1 打印机状态字节（DLE EOT 1 / GS r 1）

Bit	0	1
0	（固定 0）	—
1	纸量正常	纸将尽（Near-end，视机型）
2	（固定 0）	—
3	在线	脱机
4	（固定 0）	—
5	正常	恢复请求（Recoverable error）
6	（固定 0）	—
7	（固定 0）	—

n=2 脱机状态字节

Bit	0	1
0	(固定 0)	—
1	关盖	开盖
2	未走纸	走纸键走纸中
3	正常	纸尽停止
4	正常	错误 (Recoverable)
5	正常	不可恢复错误
6	(固定 0)	—
7	(固定 0)	—

n=3 错误状态字节

Bit	0	1
0	(固定 0)	—
1	正常	切刀错误
2	正常	不可恢复错误
3	正常	可恢复错误
4	正常	打印头温度异常
5	(固定 0)	—
6	(固定 0)	—
7	(固定 0)	—

n=4 纸传感器状态字节

Bit	含义 (0 / 1)
0, 1	纸将尽传感器: 00=纸充足, 11=纸将尽 (视机型)
2, 3	纸尽传感器: 00=有纸, 11=纸尽 (视机型)
4-7	(固定 0)

10.3 状态相关 API 汇总

方法	说明
GetPrinterStatus(connection)	便捷取状态 int
IsPrinterOnline(connection)	是否在线
HasPaper(connection)	是否有纸
IsCoverOpen(connection)	是否开盖
QueryRtStatus(connection, type, timeout)	DLE EOT 主动查询
QueryTransmitStatus(connection, type, timeout)	GS r 主动查询

11. 常量参考

说明: `PrinterConstants` 中的 `int / byte[]` 常量供集成方在业务代码中使用 (如 `PrinterConstants::AlignmentCenter`)。命名规则为 C# PascalCase; 括号内注明对应的 Android 常量名, 便于迁移比对。

11.1 对齐与切纸

```
PrinterConstants::AlignmentLeft    // 0 - 左对齐 (Android: ALIGNMENT_LEFT)
PrinterConstants::AlignmentCenter  // 1 - 居中 (Android: ALIGNMENT_CENTER)
PrinterConstants::AlignmentRight   // 2 - 右对齐 (Android: ALIGNMENT_RIGHT)

PrinterConstants::CutFull          // 0 - 全切 (Android: CUT_FULL)
PrinterConstants::CutPartial       // 1 - 半切 (Android: CUT_PARTIAL)
```

11.2 二维码

```
PrinterConstants::QrcodeSizeSmall // 1
PrinterConstants::QrcodeSizeNormal // 2
PrinterConstants::QrcodeSizeLarge  // 3

PrinterConstants::QrEccLevelL      // 1 - 约 7% 纠错
PrinterConstants::QrEccLevelM      // 2 - 约 15% 纠错
PrinterConstants::QrEccLevelQ      // 3 - 优化纠错
PrinterConstants::QrEccLevelH      // 4 - 约 30% 纠错
```

11.3 页模式方向

```
PrinterConstants::PageDirectionLeftToRight // 0
PrinterConstants::PageDirectionBottomToTop // 1
PrinterConstants::PageDirectionRightToLeft // 2
PrinterConstants::PageDirectionTopToBottom // 3
```

11.4 打印模式组合

```
PrinterConstants::PrintModeNormal    // 0 - 正常
PrinterConstants::PrintModeDoubleWidth // 1 - 倍宽
PrinterConstants::PrintModeDoubleHeight // 2 - 倍高
PrinterConstants::PrintModeDouble     // 3 - 倍宽+倍高
PrinterConstants::PrintModeBold       // 8 - 加粗
PrinterConstants::PrintModeReverse    // 16 - 反白
PrinterConstants::PrintModeUnderline  // 128 - 下划线
```

11.4.1 光栅图压缩模式

用于 `PrintRasterImageFrom*` 的 `compressionMode` 参数:

```
PrinterConstants::RasterCompressionGsV0 // 0 - GS v 0 标准光栅
PrinterConstants::RasterCompressionLineRle // 1 - US ( P 逐行 RLE (需固件支持) )
PrinterConstants::RasterCompressionJbig // 2 - US ( J JBIG 压缩 )
PrinterConstants::RasterCompressionAuto // 3 - 自动 (默认, GS v 0 ↔ JBIG)
```

11.5 钱箱与行间距

```
PrinterConstants::CashDrawerPin2    // 0 - 钱箱引脚 2
PrinterConstants::CashDrawerPin5    // 1 - 钱箱引脚 5

PrinterConstants::LineSpacingDefault // 0 - 默认行间距
PrinterConstants::LineSpacing18     // 1 - 1/8 英寸
PrinterConstants::LineSpacing16     // 2 - 1/6 英寸
```

11.6 状态相关常量

完整位定义见 第 10 节。

```
// SDK 便捷判断（简化值，非 ESC/POS 原位）
PrinterConstants::StatusOffline    // 0x00
PrinterConstants::StatusOnline     // 0x01
PrinterConstants::StatusPaperOut   // 0x02
PrinterConstants::StatusCoverOpen  // 0x04

// 主动查询 type (DLE EOT / GS r)
PrinterConstants::StatusTypePrinter // 1
PrinterConstants::StatusTypeOffline // 2
PrinterConstants::StatusTypeError   // 3
PrinterConstants::StatusTypePaperSensor // 4
```

11.7 条码类型（标准小票 ESC/POS）

测试条码功能时，不确定选哪个 TYPE 请直接用 `BarcodeTypeCode128`。完整选型说明见 7.4 节。

```
// 通用首选（测试推荐）
PrinterConstants::BarcodeTypeCode128 // 0x49 - 字母数字均可

// 商品零售
PrinterConstants::BarcodeTypeEan13 // 0x43 - 12~13 位数字
PrinterConstants::BarcodeTypeEan8  // 0x44 - 7~8 位数字
PrinterConstants::BarcodeTypeUpca  // 0x41 - 11~12 位数字
PrinterConstants::BarcodeTypeUpce  // 0x42 - UPC-E 压缩码

// 其他
PrinterConstants::BarcodeTypeCode39 // 0x45 - 工业标识
PrinterConstants::BarcodeTypeItf    // 0x46 - 偶数位数字
PrinterConstants::BarcodeTypeCodabar // 0x47 - 以 A~D 起止
PrinterConstants::BarcodeTypeCode93 // 0x48 - 高密度

// HRI 可读文字位置
PrinterConstants::BarcodeHriNone    // 0 - 不显示
PrinterConstants::BarcodeHriBelow   // 2 - 条码下方（常用）

// 旧版别名（等价于 BarcodeType*）
PrinterConstants::BarcodeCode128    // = BarcodeTypeCode128
PrinterConstants::BarcodeCode39     // = BarcodeTypeCode39
PrinterConstants::BarcodeEan13      // = BarcodeTypeEan13
```

11.8 标签模式条码类型（JPL，与 ESC/POS 不同）

标签模式 `PrintLabelBarcode()` 的 type 使用 **0~29** 编号，与上方 `BarcodeType*` 无关：

```
PrinterConstants::LabelBarcodeTypeCode128 // 8 - 标签模式 Code 128（常用）
PrinterConstants::LabelBarcodeHeightDefault // 80 - 标签条码默认高度
PrinterConstants::LabelBarcodeUnitWidth2 // 2 - 标签条码默认模块宽度
PrinterConstants::LabelFontHeight24 // 24 - 标签常用字号
```


12. 完整示例

12.1 小票打印 (蓝牙)

```
#include "hspos/HsposPrinterSdk.hpp"
using namespace hspos;

public void PrintReceipt()
{
    auto& sdk = PrinterSdk::GetInstance();
    auto devices = sdk.GetBluetoothPairedDevices();
    if (devices.size() == 0)
    {
        std::cerr <<("未找到已配对打印机");
        return;
    }

    auto conn = sdk.ConnectBluetoothPrinter(devices[0]);
    if (conn == nullptr || !conn.IsConnected())
    {
        std::cerr <<("连接失败");
        return;
    }

    try
    {
        sdk.InitPrinter(conn);

        // 标题
        sdk.SetAlignment(conn, PrinterConstants::AlignmentCenter);
        sdk.SetChineseDoubleSize(conn, true);
        sdk.SetBold(conn, true);
        sdk.PrintTextInGbk(conn, "=== 收银小票 ===\n\n");

        // 正文
        sdk.SetBold(conn, false);
        sdk.SetChineseDoubleSize(conn, false);
        sdk.SetAlignment(conn, PrinterConstants::AlignmentLeft);
        sdk.PrintTextInGbk(conn, "商品: 苹果 x2\n");
        sdk.PrintTextInGbk(conn, "单价: 5.00 元\n");
        sdk.PrintTextInGbk(conn, "合计: 10.00 元\n\n");

        // 分隔线
        sdk.FixedHorizontalLine(conn);

        // 条码
        sdk.PrintBarcode(conn,
            PrinterConstants::BarcodeTypeCode128, "20260707001",
            PrinterConstants::BarcodeHriBelow, 80, 2);

        // 二维码
        sdk.PrintQrCode(conn, "https://example.com/order/001", 3);

        // 切纸
        sdk.FeedLine(conn);
        sdk.CutPaper(conn, PrinterConstants::CutPartial);

        std::cerr <<("打印完成");
    }
    catch (Exception ex)
    {
        std::cerr <<($"打印失败: {ex.Message}");
    }
    finally
```

```
    {  
        sdk.DisconnectPrinter(conn);  
    }  
}
```

12.2 标签打印 (网络)

```
public void PrintLabel()  
{  
    auto& sdk = PrinterSdk::GetInstance();  
    auto conn = sdk.ConnectNetworkPrinter("192.168.1.100", 9100);  
    if (!conn || !conn->IsConnected()) {  
        // 连接失败  
        return;  
    }  
    // 在工作线程中执行打印, 避免阻塞 UI  
    sdk.LabelPageBegin(conn, 0, 0, 576, 400, 0);  
  
    sdk.PrintLabelTextGbk(conn, "产品名称", 30, 30, 24, 0);  
    sdk.PrintLabelBarcode(conn,  
        PrinterConstants::LabelBarcodeTypeCode128,  
        "6901234567890", 30, 80, 80, 2);  
    sdk.PrintLabelQrCode(conn, "https://example.com/p/001", 30, 200, 3);  
    sdk.DrawLabelRect(conn, 10, 10, 566, 390);  
  
    sdk.LabelPagePrint(conn);  
    sdk.DisableLabelMode(conn);  
    sdk.DisconnectPrinter(conn);  
},  
onFailed: errorMessage =>  
{  
    std::cerr <<(errorMessage);  
});  
}
```

12.3 并口打印 (Windows 独有, 工控设备常用)

```
public void PrintOverParallelPort()  
{  
    auto& sdk = PrinterSdk::GetInstance();  
    var ports = sdk.GetParallelDevices();  
    if (ports.Count == 0)  
    {  
        std::cerr <<("未检测到并口设备");  
        return;  
    }  
  
    auto conn = sdk.ConnectParallelPrinter(ports[0]);  
    if (conn == nullptr)  
    {  
        std::cerr <<("并口连接失败");  
        return;  
    }  
  
    sdk.InitPrinter(conn);  
    sdk.PrintText(conn, "并口打印测试\n");  
    sdk.FeedAndCutPaper(conn);  
    sdk.DisconnectPrinter(conn);  
}
```

13. 常见问题与排查

13.1 连接问题

现象	可能原因	解决方案
蓝牙连接失败	未配对，或本机蓝牙未开启	先在 Windows 设置中配对，或调用 <code>PairBluetoothDevice</code>
蓝牙扫描无结果	蓝牙适配器未启用/打印机不在可发现模式	检查系统蓝牙开关；确认打印机处于配对/可发现状态
USB 连接失败	驱动未安装或端口被占用	在设备管理器确认驱动安装正常；关闭占用该端口的其他打印软件
网络连接失败	IP 不可达或端口未开放	ping 打印机 IP，确认端口 9100 开放，检查防火墙规则
串口连接失败	端口号错误或参数不匹配	在设备管理器确认端口号；核对波特率/校验位/流控设置
并口连接失败	无物理 LPT 或端口被占用	确认工控机具备 LPT 接口；关闭占用该端口的其他程序

13.2 打印问题

现象	可能原因	解决方案
中文乱码	编码不匹配	使用 <code>PrintTextInGbk</code> 等对应编码方法
图片不打印	文件路径无效或格式不支持	检查文件路径与图片格式（建议 PNG/JPG）
切纸无效	打印机不支持切刀	改用 <code>FeedLine</code> 走纸代替
条码无法扫描	数据格式与类型不匹配	确认 <code>type</code> 使用 <code>BarcodeType*</code> 常量；测试时优先用 <code>BarcodeTypeCode128</code> + <code>"1234567890"</code> ；EAN/UPC 需符合位数要求
并口/串口打印后无状态回读	硬件本身不支持双向通信	属预期行为，改用网口/USB 验证状态查询功能

13.3 开发建议

- 网络连接**：当前为同步 `ConnectNetworkPrinter`；在 UI 程序中请放到工作线程调用。
- 检查返回值**：所有打印方法返回 `bool`，失败时应记录日志。
- 及时释放资源**：应用退出前调用 `sdk.Release()`，并对每个连接调用 `DisconnectPrinter`。
- 使用常量**：不要硬编码对齐、条码类型等数值。
- 连接后先 init**：打印前调用 `InitPrinter(connection)` 确保打印机就绪。
- 一次连接多次打印**：同一 `IPrinterConnection` 可复用，无需每次重连。

13.4 日志与调试

- 可在应用中使用 `Microsoft.Extensions.Logging` 或 `Console` 输出记录关键调用与异常，SDK 内部对多数底层异常做了吞掉处理（返回 `false / nullptr`），排查连接问题建议先用 第 10 节 的状态查询 API 或抓包确认底层通信是否成功。
- 建议先用官方 窗体测试工具 `win_demo`（`samples\win_demo`）验证目标打印机连接与出纸正常，再接入业务工程。

14. API 完整速查表

以下列出 `PrinterSdk.cs` 中全部 **公开方法**（含重载）。

14.1 单例与生命周期

方法	返回值	说明
<code>GetInstance()</code>	<code>PrinterSdk</code>	获取单例（Windows 不需要 Context）
<code>Release()</code>	<code>void</code>	释放 SDK 资源

14.2 连接与通信

方法	返回值	说明
GetUsbDevices()	std::vector<UsbPrinterInfo>	USB 设备列表
ConnectUsbPrinter(string) / ConnectUsbPrinter(UsbPrinterInfo)	std::shared_ptr<IPrinterConnection>	连接 USB
GetBluetoothPairedDevices()	std::vector<BluetoothPrinterInfo>	已配对蓝牙
GetBluetoothDevices() / GetBluetoothPrinters()	std::vector<BluetoothPrinterInfo>	可用蓝牙设备
ConnectBluetoothPrinter(string) / ConnectBluetoothPrinter(BluetoothPrinterInfo, bool)	std::shared_ptr<IPrinterConnection>	连接蓝牙
StartBluetoothDiscovery()	bool	后台开始蓝牙扫描
StopBluetoothDiscovery()	void	停止蓝牙扫描
RefreshBluetoothDevices()	bool	同步刷新（已配对 +扫描）
DiscoverBluetoothDevices(bool)	std::vector<BluetoothPrinterInfo>	同步扫描
DiscoverBluetoothDevices(bool)	std::vector<BluetoothPrinterInfo>	同步扫描 (Windows 额外)
PairBluetoothDevice(string, string?)	bool	触发系统配对 (Windows 额外)
(蓝牙监听器)	—	C++ 暂未提供，请 用主动枚举/扫描
ConnectNetworkPrinter(string, int)	std::shared_ptr<IPrinterConnection>	同步网络连接
ConnectNetworkPrinter(const std::string&, int)	shared_ptr<IPrinterConnection>	网络连接（同步 TCP）
GetSerialDevices()	std::vector<string>	串口设备列表
ConnectSerialPrinter(string, int) / ConnectSerialPrinter(SerialPortSettings)	std::shared_ptr<IPrinterConnection>	连接串口
GetParallelDevices()	std::vector<string>	并口设备列表 (Windows 额外)
ConnectParallelPrinter(string)	std::shared_ptr<IPrinterConnection>	连接并口 (Windows 额外)
DisconnectPrinter(shared_ptr<IPrinterConnection>)	void	断开连接
(缓冲清除/取消打印)	—	C++ 暂未封装，可 用 SendCommand
QueryRtStatus(connection, type, timeout)	byte[]?	查询实时状态（DLE EOT）
QueryTransmitStatus(connection, type, timeout)	byte[]?	传送状态（GS r）

14.3 标准打印 — 初始化与文本

方法	说明
<code>InitPrinter(connection)</code>	初始化打印机
<code>ResetPrinter(connection)</code>	重置打印机
<code>SendSelfTest(connection)</code>	自检
<code>SendCommand(connection, byte[])</code>	发送原始指令
<code>PrintText(connection, text)</code>	默认编码文本
<code>PrintTextInUtf8(connection, text)</code>	UTF-8 文本
<code>PrintTextInGbk(connection, text)</code>	GBK 文本
<code>PrintTextInBig5(connection, text)</code> (别名 <code>PrintTextInBIG5</code>)	BIG5 文本
<code>PrintTextInShiftJis(connection, text)</code>	Shift-JIS 文本
<code>PrintTextInEucKr(connection, text)</code>	EUC-KR 文本

14.4 标准打印 — 格式设置

方法	说明
<code>SetAlignment(connection, alignment)</code>	对齐
<code>SetChineseDoubleSize(connection, enable)</code>	汉字倍宽倍高 (bool)
<code>SetChineseDoubleSize(connection, n)</code>	汉字倍宽倍高 (int, 0/1)
<code>SetTextScale(connection, w, h)</code>	字符尺寸 GS ! n
<code>SetPrintMode(connection, mode)</code>	打印模式 ESC ! n
<code>SetBold(connection, bold)</code>	加粗
<code>SetUnderline(connection, underline)</code>	下划线
<code>SetReverse(connection, reverse)</code>	反白
<code>SetLineSpacing(connection, spacing)</code>	行间距
<code>SetCharSpacing(connection, spacing)</code>	字符间距
<code>SetSingleByteMode(connection)</code>	单字节模式
<code>SetCharacterSet(connection, charset)</code>	字符集
<code>SetCharacterCodepage(connection, codepage)</code>	代码页
<code>SetChineseMode(connection)</code>	汉字模式 FS &
<code>SetDoubleByteEncoding(connection, n)</code>	双字节编码
<code>SetAsciiTextFontType(connection, fontType)</code>	ASCII 字体 A/B
<code>SetTextUpsideDown(connection, upsideDown)</code>	上下倒置
<code>SetTextRotate(connection, rotate)</code>	旋转 90°
<code>SetTextLineHeight(connection, lineHeight)</code>	行高
<code>SetAsciiTextCharRightSpacing(connection, spacing)</code>	ASCII 右间距
<code>SetKanjiTextCharSpacing(connection, left, right)</code>	汉字左右间距
<code>SetPrintPosition(connection, position)</code>	打印起始位置
<code>SetPrintAreaLeftMargin(connection, margin)</code>	左边距
<code>SetPrintAreaWidth(connection, width)</code>	打印区域宽度
<code>SetHorizontalAbsolutePrintPosition(connection, pos)</code>	水平绝对位置
<code>SetHorizontalRelativePrintPosition(connection, offset)</code>	水平相对偏移
<code>SetVerticalAbsolutePrintPosition(connection, pos)</code>	垂直绝对位置
<code>SetVerticalRelativePrintPosition(connection, offset)</code>	垂直相对偏移

14.5 标准打印 — 条码 / 二维码 / 图片

方法	说明
<code>PrintBarcode(connection, type, data, hri, height, width)</code>	一维条码
<code>SetBarcodeHriPosition(connection, pos)</code>	条码 HRI 位置
<code>SetBarcodeHeight(connection, height)</code>	条码高度
<code>SetBarcodeWidth(connection, width)</code>	条码窄条宽度
<code>SetBarcodeReadableTextFontType(connection, fontType)</code>	条码 HRI 字体
<code>SetBarcodeReadableTextPosition(connection, position)</code>	条码 HRI 文字位置
<code>PrintQrCode(connection, content, size)</code> (别名 <code>PrintQRCode</code>)	二维码 (简易)
<code>PrintQrCodeStandard(connection, content, size, align, ecc)</code> (别名 <code>PrintQRCodeStandard</code>)	二维码 (标准)
<code>PrintDoubleQrCode(connection, ...)</code>	双二维码
<code>PrintPdf417(connection, ...)</code>	PDF417
<code>PrintRasterImageFromFile(connection, path)</code>	图片 (文件, 默认参数)
<code>PrintRasterImageFromFile(connection, path, w, h, mode)</code>	图片 (文件+尺寸+压缩模式)
<code>PrintRasterImageFromData(connection, bytes)</code>	图片 (字节, 默认参数)
<code>PrintRasterImageFromData(connection, bytes, w, h, mode)</code>	图片 (字节+尺寸+压缩模式)
<code>PrintRasterImageFromPixels(connection, pixels, w, h)</code>	图片 (像素, 默认参数)
<code>PrintRasterImageFromPixels(connection, pixels, w, h, tw, th, mode)</code>	图片 (像素+尺寸+压缩模式)

14.6 标准打印 — 走纸 / 切纸 / 外设 / 缓冲

方法	说明
<code>FeedLine(connection)</code>	换行
<code>FeedPaper(connection, dots)</code>	走纸 N 点 (ESC J)
<code>PrintAndFeedLines(connection, lines)</code>	走纸 N 行 (ESC d)
(制表相关)	C++ 暂未提供
<code>FeedAndCutPaper(connection)</code>	走纸并切纸
<code>CutPaper(connection, cutType)</code>	切纸
<code>PrintHorizontalLine(connection, start, end)</code>	水平线
<code>FixedHorizontalLine(connection)</code>	固定宽度水平线
<code>OpenCashDrawer(connection, drawerIndex)</code>	开钱箱
<code>Beep(connection)</code>	蜂鸣 (默认)
<code>Beep(connection, count, duration)</code>	蜂鸣 (自定义)
(缓冲/取消)	C++ 暂未封装

14.7 页模式

方法	说明
SelectPageMode(connection)	进入页模式
ExitPageMode(connection)	退出页模式
SelectPageModeEx(connection, ...)	进入页模式 (3 个重载)
PagePrintPage(connection)	打印页内容
PageClearPage(connection) (别名 CancelPrintDataInPageMode)	清除页缓冲
SetMovementUnits(connection, x, y)	移动单位
PageSetPageArea(connection, x, y, w, h) (别名 SetPrintAreaInPageMode)	设置页区域
PageSetPageDrawDirection(connection, dir) (别名 SetPrintAreaDirection)	页绘制方向
DrawLineInPageMode(connection, x1, y1, x2, y2[, lineType])	绘制线段 (2 重载)
DrawRectangleInPageMode(connection, mode, x, y, w, h)	绘制矩形
DrawRectangleBorderInPageMode(connection, ...)	矩形边框
DrawFilledRectangleInPageMode(connection, ...)	实心矩形
DrawTextInPageMode(connection, x, y, text)	绘制文本
DrawTextUtf8InPageMode(connection, ...)	UTF-8 文本
DrawTextGbkInPageMode(connection, ...) (别名 DrawTextGBKInPageMode)	GBK 文本
DrawTextBIG5InPageMode(connection, ...)	BIG5 文本
DrawTextShiftJISInPageMode(connection, ...)	Shift-JIS 文本
DrawTextEUCKRInPageMode(connection, ...)	EUC-KR 文本
EstimateTextWidthDots(text, charWidthDots)	估算文本宽度
EstimateCode128WidthDots(data, moduleWidth)	估算 Code128 宽度
DrawTextCenteredInPageMode(connection, ...)	居中绘制
DrawTextRightInPageMode(connection, ...)	右对齐绘制
DrawTextLineLeftRightInPageMode(connection, ...)	左右分列绘制
DrawBarcodeInPageMode(connection, ...)	绘制条码 (2 重载)
DrawBarcodeInPageModeWithCenteredHri(connection, ...)	条码+居中 HRI
DrawQrCodeInPageMode(connection, ...)	绘制二维码
DrawImageFromFileInPageMode(connection, ...)	绘制图片 (文件)
DrawImageFromDataInPageMode(connection, ...)	绘制图片 (数据)
DrawImageFromPixelsInPageMode(connection, ...)	绘制图片 (像素)
PrintNvBitmap(connection, nvNumber, mode)	NV 位图 (标准模式)
DrawNvBitmapInPageMode(connection, ...)	NV 位图 (页模式坐标)
DrawNvBitmapCenteredInPageMode(connection, ...)	NV 位图 (页模式居中)

14.8 标签模式

方法	说明
EnableLabelMode(connection)	启用标签模式
DisableLabelMode(connection)	禁用标签模式
LabelPageBegin(connection, x, y, w, h, rotation)	开始标签页
BackPaperToPrintPosition(connection)	回纸到打印位
LabelPagePrint(connection)	打印标签
LabelPagePrint(connection, printNum)	打印 N 份标签
PrintLabelText(connection, ...)	标签文本 (2 个重载)
PrintLabelTextUtf8(connection, ...)	标签 UTF-8 (2 个重载)
PrintLabelTextGbk(connection, ...)	标签 GBK (2 个重载)
PrintLabelBarcode(connection, ...)	标签条码 (2 个重载)
SetLabelBarcodeTextPosition(connection, position)	标签条码文字位置
PrintLabelQrCode(connection, ...)	标签二维码 (2 个重载)
PrintLabelPdf417Code(connection, ...)	标签 PDF417 (2 个重载)
DrawLabelImageFromFile(connection, ...)	标签图片 (文件)
DrawLabelImageFromData(connection, ...)	标签图片 (数据)
DrawLabelImageFromPixels(connection, ...)	标签图片 (像素)
DrawLabelImageAdvanced(connection, ...)	标签图片 (高级)
DrawLabelLine(connection, ...)	标签线段 (2 个重载)
DrawLabelRect(connection, ...)	标签矩形 (2 个重载)
DrawLabelBox(connection, ...)	标签实心框

14.9 状态查询

方法	返回值	说明
GetPrinterStatus(connection)	int	获取状态字节
IsPrinterOnline(connection)	bool	是否在线
HasPaper(connection)	bool	是否有纸
IsCoverOpen(connection)	bool	是否开盖
QueryRtStatus(connection, type, timeout)	byte[]?	DLE EOT 实时状态
QueryTransmitStatus(connection, type, timeout)	byte[]?	GS r 传送状态

14.10 接口 / 类型

名称	说明
<code>IPrinterConnection</code>	打印机连接句柄接口（对齐 Android <code>PrinterConnection</code> ）
（蓝牙监听器）	C++ 暂未提供
<code>UsbPrinterInfo</code>	USB 设备信息（对齐 Android <code>UsbDevice</code> ）
<code>BluetoothPrinterInfo</code>	蓝牙设备信息（对齐 Android <code>BluetoothDevice</code> ）
<code>SerialPortSettings</code>	串口参数（波特率/校验位/数据位/停止位/流控）

方法签名与参数详见 `PrinterSdk.hpp` 或 IDE 自动补全，也可参阅 `PrinterSDK_API_说明文档.md`。

15. 与 Android / C# 版的平台差异说明

分类	Android	Windows	说明
生命周期	<code>getInstance(Context)</code> + <code>onRequestPermissionsResult</code>	<code>GetInstance()</code> (无参)	Windows 无运行时权限模型, 不需要转发权限回调
权限体系	AndroidManifest 声明 + 运行时申请	无需声明; 依赖驱动/设备管理器	见 第 5 节
USB 权限 API	<code>hasUsbPermission</code> / <code>requestUsbPermission</code>	不提供	Windows 无 USB 运行时权限模型
通信管理器	<code>getCommunicationManager()</code> 公开	不提供	内部实现细节, 不对外暴露
集成方式	AAR + Gradle 依赖	NuGet / 项目引用	CMake 链接静态库 / 源码编译
命名风格	camelCase (Java/Kotlin)	PascalCase (C#)	如 <code>printText</code> → <code>PrintText</code>
连接方式	USB / 蓝牙 / 网络 / 串口	USB / 蓝牙 / 网络 / 串口 / 并口 (LPT)	并口为 Windows 独有能力, 见 6.5 节
蓝牙扫描	仅监听器回调模式	监听器回调 + 同步阻塞扫描 (<code>DiscoverBluetoothDevices</code>)	Windows 额外提供同步 API, 简化控制台/服务场景
光栅图重载	3 个重载 (含 tw/th 但无 mode 的中间重载)	2 个重载 (无尺寸 / 尺寸+模式必填)	业务层显式传 <code>RasterCompressionAuto</code> 即可对齐 Android 默认行为
USB 设备类型	<code>UsbDevice</code> (Android API)	<code>UsbPrinterInfo</code> (<code>VendorId</code> / <code>ProductId</code> / <code>PortName</code> / <code>DevicePath</code>)	Windows 侧走打印口/SetupAPI, 非 WinUSB
蓝牙设备类型	<code>BluetoothDevice</code> (Android API)	<code>BluetoothPrinterInfo</code> (<code>Address</code> / <code>Name</code> / <code>IsPaired</code>)	

分类	Android	Windows	说明
异步网络回调	<code>CommunicationManager.NetworkConnectionCallback</code> 接口	<code>Action<IPrinterConnection></code> + <code>Action<string></code> 委托	C++ 当前仅同步 <code>ConnectNetworkPrinter</code> ; 异步占位见 API 文档

API 完备性：已实现方法见 PrinterSDK_API_说明文档.md 正文；Android 有而 C++ 暂未实现 / 不移植的方法，已在同一文档增加 **「暂未实现 API」** 与 **「Android 平台专属 API」** 占位章节（含规划签名与临时替代说明）。对照表见 Android_API覆盖对照.md。

附录：版本更新记录

版本	日期	主要变更
1.0.0 – 1.6.2	2025-08-28 ~ 2026-07-09	Android 版本迭代（USB/蓝牙/网络/串口、权限自动化、页模式接入 ESC/POS、状态查询等），详见 Android 端 PrinterSDK_使用说明书.md 附录
1.6.3-windows-cpp	2026-07-18	Windows / C++17 对齐版：CMake；多通道连接；POS/页模式/标签/光栅；FeedPaper 为点数
1.6.3-windows	2026-07-16	Windows / .NET 8 首个对齐版本：对齐 Android 1.6.3 API 面；新增并口（LPT）连接；新增同步蓝牙扫描 <code>DiscoverBluetoothDevices</code> 与主动配对 <code>PairBluetoothDevice</code> ；移除 Android 专属权限回调与 USB 权限 API

技术支持：如有问题，请参考本文 第 13 节，或使用 `samples\win_demo` 窗体测试工具 `win_demo`进行连接与打印诊断。方法级详细说明见 PrinterSDK_API_说明文档.md。