

- 一、SDK导入说明
- 二、SDK开放类说明
 - 2.0 PTPrinter
 - 2.1 PTDispatcher
 - 2.2 PTCommandCPCL
 - 2.3 PTCommandESC
 - 2.4 PTCommandTSPL
 - 2.5 PTCommandZPL
 - 2.6 PTEncode
 - 2.7 PTBitmap
 - 2.9 PTLLabel
 - 2.10 PTOldCommandCPCL
 - 2.11 PTOldCommandTSPL
 - 2.12 PTCommandCommon
- 三、如何连接外设说明
- 四、指令使用案例
 - 4.0 SDK提供的功能
 - 4.1 通过模板打印
 - 4.2 SDK打印图片案例(参考Demo)

一、SDK导入说明

- 1.把压缩包里面的PrinterSDK.framework拖进你的项目中
- 2.需要另外添加SystemConfiguration.framework这个系统库
- 3.app打包时需要将Enable Bitcode的选项设置成No，防止打包时间过长
- 4.在Build Settings -> Linking -> Other Linker Flags 选项中添加-ObjC
- 5.使用蓝牙的地方添加#import <CoreBluetooth/CoreBluetooth.h>，视情况而定
- 6.注释警告解除方法：Build Settings -> Documentations Comments -> 将YES改为NO
- 7.由于CPCL、TSPL接口优化后，老用户如果使用该版本的SDK，则修改的内容较多，因此SDK预留了PTOldCommandCPCL、PTOldCommandTSPL两个类，他们分别保留了之前旧的接口
- 8.每次发送数据，SDK都会把PTDispatcher中的receiveDataBlock、sendSuccessBlock、sendFailureBlock、sendProgressBlock置为空
- 9.压缩包中有两个SDK，一个只有真机架构的，用于打包上架；一个含有真机和模拟器架构的，用于调试

二、SDK开放类说明

2.0 PTPrinter

外设的属性类，定义了外设的名称name、mac地址、蓝牙的广播包advertisement、蓝牙的uuid、信号强度、WiFi相关的router和ip等，当扫描到外设或者连接外设、断开连接时，所传的参数就是属性类

- 属性

```
/// 打印机名称
@property(strong, nonatomic, readwrite) NSString *name;
/// 打印机mac地址
@property(strong, nonatomic, readwrite) NSString *mac;
/// 打印机蓝牙模块
@property(assign, nonatomic, readwrite) PTPrinterModule module;
/// 发现蓝牙时获取到的广播信息
@property(strong, nonatomic, readwrite) NSDictionary *advertisement;
/// 蓝牙外设UUID
@property(strong, nonatomic, readwrite) NSString *uuid;
/// 发现外设时获取到的信号强度值，单位分贝
@property(strong, nonatomic, readwrite) NSNumber *rssi;
/// 信号强度等级分0-5级
@property(strong, nonatomic, readwrite) NSNumber *strength;
/// 由信号强度计算的距离
@property(strong, nonatomic, readwrite) NSNumber *distance;
/// 蓝牙外设
@property(strong, nonatomic, readwrite) CBPeripheral *peripheral;
/// 外设的ip地址
@property(strong, nonatomic, readwrite) NSString *ip;
/// 端口
@property(strong, nonatomic, readwrite) NSString *port;
```

2.1 PTDispatcher

打印机通讯类

- 1.定义连接方式、手机蓝牙状态、连接失败类型、固件升级状态等枚举
- 2.定义Block协议和属性，这些Block属性APP可不用处理
- 3.定义和打印机交互的接口

- 枚举

```
/// 连接模式
typedef NS_ENUM(NSInteger, PTDispatchMode) {

    /// 未知类型
    PTDispatchModeUnconnect = 0,
```

```

    /// 蓝牙
    PTDispatchModeBLE          = 1,

    /// 无线
    PTDispatchModeWiFi         = 2
};

/// 手机蓝牙状态
typedef NS_ENUM(NSInteger, PTBluetoothState) {
    /// 未授权, 请前往系统设置授权
    PTBluetoothStateUnauthorized = 0,
    /// 蓝牙未开
    PTBluetoothStatePoweredOff   = 1,
    /// 正常
    PTBluetoothStatePoweredOn    = 2,
};

/// 打印完成后打印机返回的状态
typedef NS_ENUM(NSInteger, PTPrintState) {

    /// 打印成功
    PTPrintStateSuccess          = 0xcc00,

    /// 打印失败 (缺纸)
    PTPrintStateFailurePaperEmpty = 0xcc01,

    /// 打印失败 (开盖)
    PTPrintStateFailureLidOpen    = 0xcc02
};

/// 返回连接的错误类型
typedef NS_ENUM(NSInteger, PTConnectError) {

    /// 连接超时
    PTConnectErrorBleTimeout      = 0,

    /// 获取服务超时
    PTConnectErrorBleDisvocerServiceTimeout = 1,

    /// 验证超时
    PTConnectErrorBleValidateTimeout = 2,

    /// 未知设备
    PTConnectErrorBleUnknownDevice = 3,

    /// 系统错误, 由coreBluetooth框架返回
    PTConnectErrorBleSystem       = 4,

    /// 验证失败

```

```

PTConnectErrorBleValidateFail = 5,

/// 无线连接超时
PTConnectErrorWifiTimeout = 6,

/// socket错误
PTConnectErrorWifiSocketError = 7
};

/// 返回固件升级错误
typedef NS_ENUM(NSInteger, PTUpgradeFirmwareState) {

    /// 升级成功
    PTUpgradeFirmwareStateSuccess = 0,

    /// 升级失败, 数据长度错误
    PTUpgradeFirmwareStateFailureDataLengthError,

    /// 升级失败, 验证或者校验失败
    PTUpgradeFirmwareStateFailureValidateFail,

    /// 升级失败, 写入超时
    PTUpgradeFirmwareStateFailurewriteTimeout,

    /// 升级失败, 包序号错误
    PTUpgradeFirmwareStateFailurePackageNumberError,

    /// 升级失败, 包长度错误
    PTUpgradeFirmwareStateFailurePackageLengthError,

    /// 升级失败, 写入失败
    PTUpgradeFirmwareStateFailurewriteFail,

    /// 升级失败
    PTUpgradeFirmwareStateFail
};

```

- 属性

```

@property (assign, nonatomic) PTDispatchMode
mode;
///该属性表示连接后的打印机对象
@property (strong, nonatomic, readwrite) PTPrinter
*printerConnected;
@property (copy, nonatomic, readwrite) PTSendSuccessParameterBlock
sendSuccessBlock;

```

```

@property (copy, nonatomic, readwrite) PTEmptyParameterBlock
    sendFailureBlock;
@property (copy, nonatomic, readwrite) PTNumberParameterBlock
    sendProgressBlock;
@property (copy, nonatomic, readwrite) PTDataParameterBlock
    receiveDataBlock;
@property (copy, nonatomic, readwrite) PTPrintStateBlock
    printStateBlock;
@property (copy, nonatomic, readwrite) PTPrinterParameterBlock
    findBluetoothBlock;
@property (copy, nonatomic, readwrite) PTPrinterMutableArrayBlock
    findAllPeripheralBlock;
@property (copy, nonatomic, readwrite) PTEmptyParameterBlock
    connectSuccessBlock;
@property (copy, nonatomic, readwrite) PTBluetoothConnectFailBlock
    connectFailBlock;
@property (copy, nonatomic, readwrite) PTUnconnectBlock
    unconnectBlock;
@property (copy, nonatomic, readwrite) PTNumberParameterBlock
    readRSSIBlock;
@property (copy, nonatomic, readwrite) PTPeripheralFilterBlock
    peripheralFilterBlock;
@property (copy, nonatomic, readwrite) PTUpgradeFirmwareStateBlock
    upgradeFirmwareStateBlock;

```

- 方法

```

/// 创建单例对象
+ (instancetype)share;

/// 发送数据
- (void)sendData:(NSData *)data;

/// 暂停发送
- (void)pauseWriteData;

/// 继续发送
- (void)resumeWriteData;

/// 开始扫描蓝牙
- (void)scanBluetooth;

/// 停止扫描蓝牙, 连接成功后SDK会自动停止扫描
- (void)stopScanBluetooth;

/// 扫描Wi-Fi
- (void)scanWiFi:(PTPrinterMutableArrayBlock)wifiAllBlock;

```

```
/// 获取已发现的所有打印机，每新发现新的打印机或隔三秒调用一次，返回的是打印机数组
- (void)whenFindAllBluetooth:(PTPrinterMutableArrayBlock)bluetoothBlock;

/// 发现蓝牙回调，coreBlueTooth框架每发现一台打印机就会调用，返回的是打印机数组
- (void)whenFindBluetooth:(PTPrinterParameterBlock)bluetoothBlock;

/// 获取蓝牙的rssi信号强度
- (void)whenReadRSSI:(PTNumberParameterBlock)readRSSIBlock;

/// 连接打印机，传入的是打印机对象
- (void)connectPrinter:(PTPrinter *)printer;

/// 断开打印机连接
- (void)disconnectPrinter:(PTPrinter *)printer;

/// 连接成功回调，连接成功后，会停止扫码设备
- (void)whenConnectSuccess:(PTEmptyParameterBlock)connectSuccessBlock;

/// 连接失败的回调
- (void)whenConnectFailureWithErrorBlock:
(PTBluetoothConnectFailBlock)connectFailBlock;

/// 断开连接的回调，调用disconnectPrinter断开打印机后，会调用该方法
- (void)whenUnconnect:(PTUnconnectBlock)unconnectBlock;

/// 数据发送成功的回调，数据发送完成后，会调用该方法
- (void)whenSendSuccess:(PTSendSuccessParameterBlock)sendSuccessBlock;

/// 数据发送失败的回调
- (void)whenSendFailure:(PTEmptyParameterBlock)sendFailureBlock;

/// 数据发送进度的回调
- (void)whenSendProgressUpdate:(PTNumberParameterBlock)sendProgressBlock;

/// 接收到打印机返回的数据，比如获取打印机名称就是通过该方法返回
- (void)whenReceiveData:(PTDataParameterBlock)receiveDataBlock;

/// 接收到打印机打印状态回调，使用该方法前，需要保证打印机打开了状态回调的开关，比如CPCL
指令集中的cpclTurnOnPrintStatusCallBack方法，ESC中的turnOnPrintStatusCallBack
方法
- (void)whenUpdatePrintState:(PTPrintStateBlock)printStateBlock;

/// 设置蓝牙连接超时时间
- (void)setupBleConnectTimeout:(double)timeout;

/// 设置外设过滤，在发现打印机设备的回调中进行一些过滤，返回自己想要的机型
- (void)setupPeripheralFilter:(PTPeripheralFilterBlock)block;
```

```
///  
// 注册蓝牙中心, 该接口用于兼容你自己实现的CoreBluetooth框架  
- (void)registerCentralManager:(CBCentralManager *)manager delegate:  
  (id<CBCentralManagerDelegate>)delegate;  
  
///  
// 注销代理  
- (void)unregisterDelegate;  
  
///  
// 升级固件状态回调  
- (void)whenUpgradeFirmwareStateBlock:  
  (PTUpgradeFirmwareStateBlock)upgradeFirmwareStateBlock;  
  
///  
// 手机的蓝牙状态, 为了准确获取手机蓝牙状态, 需要在app启动的时候初始化PTDispatcher  
- (PTBluetoothState)getBluetoothStatus;  
  
///  
// SDK打包时间  
- (NSString *)SDKBuildTime;
```

2.2 PTCommandCPCL

CPCL指令接口类, 详情在cpc1指令接口文档

2.3 PTCommandESC

ESC指令接口类, 详情在ESC指令接口文档

2.4 PTCommandTSPL

TSPL指令接口类, 详情在TSPL指令接口文档

2.5 PTCommandZPL

ZPL指令接口类, 详情在ZPL指令接口文档

2.6 PTEncode

编码类, 默认是kCFStringEncodingGB_18030_2000的编码

- 接口

```
/// 编码, 默认是GBK
+ (NSData *)encodeDataWithString:(NSString *)string;
+ (NSData *)encodeDataWithString:(NSString *)string encodingType:
(CFStringEncoding)encodingType;

/// 解码, default:GBK
+ (NSString *)decodeStringWithData:(NSData *)data;
+ (NSString *)decodeDataWithString:(NSData *)data encodingType:
(CFStringEncoding)encodingType;
```

2.7 PTBitmap

图片处理类,一般在SDK里已经处理

- 枚举

```
/// 压缩模式
typedef NS_ENUM(NSInteger, PTBitmapCompressMode) {

    PTBitmapCompressModeNone = 0,    /*! *~chinese 不压缩 *~english None
    */
    PTBitmapCompressModeZPL2 = 16,    /*! *~chinese ZPL2压缩算法 *~english
    ZPL2 compress */
    PTBitmapCompressModeTIFF = 32,    /*! *~chinese TIFF压缩算法 *~english
    TIFF compress */
    PTBitmapCompressModeLZO = 48,    /*! *~chinese LZO压缩算法 *~english
    LZO compress */
};

/// 图片效果
typedef NS_ENUM(NSInteger, PTBitmapMode) {

    PTBitmapModeBinary = 0,          /*! *~chinese 黑白二值图像 *~english
    Binary */
    PTBitmapModeDithering = 1,        /*! *~chinese 灰阶抖动图像 *~english
    Dithering */
    PTBitmapModeColumn = 2,          /*! *~chinese 无效 *~english not
    supported */
};
```

- 接口

```
/// 生成打印机打印图片数据
```



```

/// @param image 图片
/// @param mode 图片效果
/// @param compress 压缩模式
/// @param package 数据是否分包
/// @param inversion 数据是否需要取反
+ (NSData *)getImageData:(CGImageRef)image mode:(PTBitmapMode)mode
compress:(PTBitmapCompressMode)compress package:(BOOL)package inversion:
(BOOL)inversion;

/// 用column算法生成的图片数据
/// @param sourceBitmap 输入数据
+ (NSData *)generateColumnData:(CGImageRef)sourceBitmap;

/// 将bitmap数据转成图片
/// @param image 图片
/// @param mode 生成的位图数据类型 简单的黑白二值化或者抖动处理
+ (UIImage *)generateRenderingwithImage:(CGImageRef)image mode:
(PTBitmapMode)mode;

```

- 其他接口

```

/// 获取图像的灰度数据
/// @param image 图片
+ (NSData *)graylevel256Datas:(UIImage *)image;

/// 获取图像的灰度数据
/// @param image 图片
/// @param needReverse 是否反转灰度值 0->255 254->1
+ (NSData *)graylevel256Datas:(CGImageRef)image reverse:(BOOL)needReverse;

/// 从灰度图片数据获取图像
/// @param data 灰度数据, 每个字节代表每个像素的256阶灰度值
/// @param width 图片宽度
+ (UIImage *)imagewithGraylevel256Data:(NSData *)data width:
(NSUInteger)width;

/// 灰度数据锐化
/// @param data 灰度数据, 每个字节代表每个像素的256阶灰度值
/// @param width 图片宽度
+ (NSData *)sharpenGraylevel256Data:(NSData *)data width:
(NSUInteger)width;

/// 获取经过热补偿处理的数据
/// @param data 灰度数据, 每个字节代表每个像素的256阶灰度值
/// @param width 图片宽度

```

```

+ (NSData *)historyCompensateGraylevel256Data:(NSData *)data width:
(NSUInteger)width;

/// 返回黑白图像数据
/// @param data 灰度数据，每个字节代表每个像素的256阶灰度值
/// @param width 图片宽度
+ (NSData *)binaryDataOneBytePerPixelGraylevel256Data:(NSData *)data
width:(NSUInteger)width;

/// 将图片导入设备灰度空间获取的灰度数据
/// @param image 图片
+ (NSData *)systemGraylevel256Data:(UIImage *)image;

/// 将bitmap数据转成图片
/// @param bitmap 位图数据
/// @param height 图片高度
+ (UIImage *)image:(NSData *)bitmap height:(size_t)height;

/// 动态获取二值化的阈值
/// @param data 256灰阶数据
/// @param width 图片宽度
/// @param height 图片高度
+ (NSUInteger)getThresholdForBinaryByGrayData:(NSData *)data width:
(NSUInteger)width height:(NSUInteger)height;

/// 获取图像的灰度数据
/// @param image 输入数据
+ (NSData *)grayscaleImageForImage:(CGImageRef)image;

```

2.9 PTLabel

使用电子面单模板，只需要填充相应的表单数据，即可发送打印出一张面单

- 注意事项

1. 当使用模板打印时，您必须填充我们提供的模板使用范例中所填充的所有表单项
2. 如果有空数据项，比如申明价值为空，则传入@""空字符串
3. 不同的模板，所要填充的数据项是不同的，具体以我们的范例为准

- 属性

```

@property(strong, nonatomic, readwrite) NSString *express_company;    // 快递公司
@property(strong, nonatomic, readwrite) NSString *delivery_number;    // 运单号
@property(strong, nonatomic, readwrite) NSString *order_number;    // 订单号

@property(strong, nonatomic, readwrite) NSString *distributing;    // 集散地
@property(strong, nonatomic, readwrite) NSString *barcode;    // 条形码
@property(strong, nonatomic, readwrite) NSString *barcode_text;    // 条形码下方的字符
@property(strong, nonatomic, readwrite) NSString *qrcode;    // 二维码
@property(strong, nonatomic, readwrite) NSString *qrcode_text;    // 二维码下方的字符

@property(strong, nonatomic, readwrite) NSString *receiver_name;    // 收件人 名字
@property(strong, nonatomic, readwrite) NSString *receiver_phone;    // 收件人 电话
@property(strong, nonatomic, readwrite) NSString *receiver_address;    // 收件人 地址
@property(strong, nonatomic, readwrite) NSString *receiver_message;    // 收件人 信息

@property(strong, nonatomic, readwrite) NSString *sender_name;    // 发件人 名字
@property(strong, nonatomic, readwrite) NSString *sender_phone;    // 发件人 电话
@property(strong, nonatomic, readwrite) NSString *sender_address;    // 发件人 地址
@property(strong, nonatomic, readwrite) NSString *sender_message;    // 发件人 信息

@property(strong, nonatomic, readwrite) NSString *article_name;    // 物品名
@property(strong, nonatomic, readwrite) NSString *article_weight;    // 物品重量

@property(strong, nonatomic, readwrite) NSString *amount_declare;    // 申明价值
@property(strong, nonatomic, readwrite) NSString *amount_paid;    // 到付金额
@property(strong, nonatomic, readwrite) NSString *amount_paid_advance;    // 预付金额

```

- 接口

```
/// 由模板数据生成下发给打印机的数据
/// @param filePath 模板路径
- (NSData *)datawithSourceFile:(NSString *)filePath;

/// 生成TSPL指令的数据
- (NSData *)datawithTSPL;

/// 由模板数据生成下发给打印机的数据
/// @param source 资源
/// @param labelDict 模板定义的key
/// @param orderDetails 订单详情
- (NSData *)getTemplateData:(NSString *)source labelDict:(NSDictionary *)labelDict orderDetails:(NSArray *)orderDetails;

/// 由模板数据生成下发给打印机的数据
/// @param source 资源
/// @param labelDict 模板定义的key
- (NSData *)getTemplateData:(NSString *)source labelDict:(NSDictionary *)labelDict;
```

2.10 PTOldCommandCPCL

该类是保留SDK3.0.0之前的版本的旧CPCL接口

2.11 PTOldCommandTSPL

该类是保留SDK3.0.0之前的版本的旧TSPL接口

2.12 PTCommandCommon

该类是共用的指令接口，以下三个接口需要打印机支持才能返回数据

- 1.OTA升级蓝牙固件一般用不到
- 2.升级打印机固件有新的方式和旧的方式，新的升级方式需要打印机支持

```
/// 获取打印机型号,回的数据格式: 51333142 5400
- (void)getPrinterModelName;

/// 获取打印机固件版本号,版本号返回格式为 x.xx.xx或x.x.x 【如1.01.01或1.0.3】
```

```

- (void)getPrinterFirmwareVersion;

/// OTA蓝牙固件升级,该功能需要打印机支持
- (void)updateOTABLEFirmwareWithData:(NSData *)data;

//=====固件升级旧的方式=====

/// 打印机固件升级 (固定包大小)
/// @param data 固件数据
- (void)updatePrinterFirmwareWithData:(NSData *)data;

//=====固件升级新的方式=====
/// 获取升级固件每包的大小
- (void)getUpdatePrinterFirmwarePackage;

/// 升级打印机固件
/// @param data 固件数据
/// @param package 每包大小
- (void)updateDeviceFirmwareWithData:(NSData *)data package:
(NSInteger)package;

```

三、如何连接外设说明

连接外设用到的几个方法，具体情况参考Demo

- BLE

```

//Swift5:

//获取蓝牙状态
PTDispatcher.share().getBluetoothStatus()

//开始扫描蓝牙
PTDispatcher.share().scanBluetooth()

//扫描到的蓝牙，以数组的形式返回
PTDispatcher.share()?.whenFindAllBluetooth({ (array) in

})

//关闭扫描，连接成功后会自动关闭
PTDispatcher.share().stopScanBluetooth()

//连接打印机
PTDispatcher.share().connect(printer)

```

```

//断开连接
PTDispatcher.share().disconnectPrinter(PTDispatcher.share().printerConnected)

//连接成功
PTDispatcher.share().whenConnectSuccess {
    //事务处理
}

//连接失败
PTDispatcher.share().whenConnectFailureWithErrorBlock { (error) in
    //事务处理
}

```

- WiFi

```

//要先连接路由器，这边需要判断下系统版本
let systemVersion = UIDevice.current.systemVersion

    if systemVersion.compare("13.0").rawValue == 0 ||
systemVersion.compare("13.0").rawValue == 1 {

        PTDispatcher.share().scanWiFi({ [weak self](ptArray) in

            })
        }else {

            let router = PTRouter.init()
            if router.connected {
                PTDispatcher.share().scanWiFi({ [weak self](ptArray) in

                    })
            }else {
                /// 未连接路由器
            }
        }

//连接打印机
PTDispatcher.share().connect(printer)

//断开连接
PTDispatcher.share().disconnectPrinter(PTDispatcher.share().printerConnected)

//连接成功
PTDispatcher.share().whenConnectSuccess {
}

```

```
//连接失败
PTDispatcher.share().whenConnectFailurewithErrorBlock { (error) in
}
```

- 发送数据

```
PTDispatcher.share().send(data)
//进度
PTDispatcher.share()?.whenSendProgressUpdate({ (progress) in

})
//发送成功
PTDispatcher.share().whenSendSuccess {
}
//发送失败
PTDispatcher.share().whenSendFailure { [weak self] in

}
// 接收蓝牙返回数据
PTDispatcher.share().whenReceiveData { (temp) in

}
```

四、指令使用案例

4.0 SDK提供的功能

- 打印格式条形码
- 打印二维码
- 打印文本
- 打印图片（黑白、灰阶抖动）
- 打印小票

通过本 Demo，您可以了解到：

- 如何导入、链接和使用 `PrinterSDK.framework` 框架
- 如何通过 Bluetooth 4.0和 便携式 BLE 蓝牙打印机进行通讯
- 如何通过 WiFi 和便携式 WiFi 打印机进行通讯
- 如何使用打印机指令集中的基础指令，并根据自己的需求分装成为功能

4.1 通过模板打印

通过模板打印，电子面单的样式已经预先编辑好，只需要填充相应的数据项，即可打印输出一张面单。这里以申通快递为例，具体也可参考Demo。

1.填充数据

```
// 使用说明：
// 1. 初始化一个 NSMutableDictionary，在相应的键值下塞入对应的数据，键值必须是下面样例中用到的键值。
// 2. 如果一个数据项没有数据 那么也需要设置成空字符串@""，比如 [templateDict setObject:@"" forKey:kCollection];
PTLabelTemplate *template = [[PTLabelTemplate alloc] init];
NSMutableDictionary *templateDict = [[NSMutableDictionary alloc] init];

[templateDict setObject:@"363604310467" forKey:LTBarcode];
[templateDict setObject:@"上海 上海市 长宁区" forKey:LTDistributing];

[templateDict setObject:@"申大通" forKey:LTRceiver];
[templateDict setObject:@"13826514987" forKey:LTRceiverContact];
[templateDict setObject:@"上海市宝山区共和新路4719弄共和小区12号306室"
forKey:LTRceiverAddress];

[templateDict setObject:@"快小宝" forKey:LTSender];
[templateDict setObject:@"13826514987\r\n" forKey:LTSenderContact];
[templateDict setObject:@"上海市长宁区北翟路1178号（鑫达商务楼）1号楼305室"
forKey:LTSenderAddress];

[templateDict setObject:@"SHENTONG" forKey:LTEExpressCompany];

NSData *cmdData = [template getShenTongTemplate:templateDict];
```

2.读入模板

使用时需要把模板文件拖入你的工程中。模板文件是 TXT 纯文本文件。

```
NSString *path = [[NSBundle mainBundle] pathForResource:@"ShenTong"
ofType:@"txt"];
NSData *templateData = [template getTemplateDatawithFilePath:path];
```

3.结合模板和数据

```
NSMutableData *finalData = [[NSMutableData alloc] initWithData:templateData];
[finalData appendData:cmdData];
```

4.发送数据

```
[[PTDispatcher share] sendData:finalData];
```


4.2 SDK打印图片案例(参考Demo)

- CPCL

```
let cmd = PTCommandCPCL.init()
cmd.cpclLabel(withOffset: 0, hRes: PTCPCLLabelResolution.resolution200,
vRes: PTCPCLLabelResolution.resolution200, height:
Int(zybImage.size.height) + 10, quantity: 1)
cmd.cpclPrintBitmap(withXPos: 0, yPos: 0, image: zybImage.cgImage,
bitmapMode: imageMode, compress: imagePressMode, isPackage: isPackage)
cmd.cpclPrint()
PTDispatcher.share()?.send(cmd.cmdData as Data)
```

- ESC

```
let cmd = PTCommandESC.init()
cmd.appendRasterImage(zybImage.cgImage, mode: imageMode, compress:
imagePressMode, package: isPackage)
PTDispatcher.share()?.send(cmd.getCommandData() as Data)
```

- TSPL

```
let imagewidth = Int((zybImage.size.width.truncatingRemainder(dividingBy:
8))) == 0 ? Int((zybImage.size.width / 8)) : Int((zybImage.size.width + 8)
/ 8)
let imageHeight =
Int((zybImage.size.height.truncatingRemainder(dividingBy: 8))) == 0 ?
Int((zybImage.size.height / 8)) : Int((zybImage.size.height + 8) / 8)
let cmd = PTCommandTSPL.init()
cmd.setCLS()
//这边指的是纸的毫米，不是像素点
cmd.setPrintDirection(PTTSCPrintDirection.normal, mirror:
PTTSCPrintStyle.normal)
cmd.setPrintAreaSizeWithWidth(imagewidth, height: imageHeight)
let ret = cmd.addBitmap(withXPos: 0, yPos: 0, mode: PTTSCBitmapMode.OR,
image: zybImage.cgImage, bitmapMode: imageMode, compress: imagePressMode)
cmd.print(withSets: 1, copies: printCopiesCount)
PTDispatcher.share()?.send(cmd.cmdData as Data)
```

- ZPL

```
/// DG~是先下载到打印机，建议用这条，可以用压缩方式
```

```

let cmd = PTCommandZPL()
cmd.xa_FormatStart()
cmd.mc_MapClear(PTZplBool.Y)
cmd.pm_PrintLabelMirrorImage(PTZplBool.N)
cmd.pw_PrintWidth(kUserDefaults.integer(forKey: PDPrintDots))
cmd.lh_LabelHome(withXPos: 0, yPos: 0)
cmd.lr_LabelReversePrint(PTZplBool.N)
cmd.xz_FormatEnd()

/// 下载
cmd.dg_DownloadGraphics(with: cgImage, bitmapMode: imageMode, compress:
imagePressMode, deviceLocation: PTZplFileLocation.R, imageName: "iZPL",
extension: "GRF")

cmd.xa_FormatStart()
cmd.pq_PrintQuantity(printCopiesCount)
cmd.fo_FieldOrigin(withXAxis: 10, yAxis: 10)
/// 打印
cmd.xg_RecallGraphic(withSourceDevice: PTZplFileLocation.R, imageName:
"iZPL", extension: "GRF", xAxisMagnification: 1, yAxisMagnification: 1)
cmd.fs_FieldSeparator()
cmd.xz_FormatEnd()

cmd.xa_FormatStart()
/// 删除
cmd.id_ImageDelete(withObjectLocation: PTZplFileLocation.R, objectName:
"iZPL", extension: "GRF")
cmd.xz_FormatEnd()
PTDispatcher.share()?.send(cmd.cmdData as Data)

```