一、SDK导入说明

```
1.把压缩包里面的PLPrinterSDK.xcframework导入你的项目中
2.由于SDK变成动态库，在导入时需要在TARGETS->General->Frameworks中选择对应的SDK，在Embed中选择Embed&Sign
3.由于用到蓝牙，需要在info.plist增加蓝牙授权的Key:<Privacy - Bluetooth Always Usage Description>和<Privacy - Bluetooth Peripheral Usage Description>
4.注释警告解除方法：Build Settings -> Documentations Comments -> 将YES改为NO
5.后台如果需要保持蓝牙连接，在info.plist的Required background modes下，需要增加App communicates using CoreBluetooth、App communicates with an accessory，详细参考Demo
6.如果需要图片增强处理，需要把img.framework导入项目，这个也是动态库需要Embed&Sign
7.ble只是用于辅助搜索，mfi用于连接
8.mfi连接需要在info.plist增加EAP协议,可参考Demo
```

二、SDK连接模块

2.0 PLConnectInterface

- 作用

  处理连接事务

- 属性

```
@property (nonatomic, weak) id<PLPrinterInterfaceDelegate>  delegate;
///< 协议
@property (nonatomic, assign) BOOL                          isConnected;
 ///< 是否连接
@property (nonatomic, strong) PLPrinterMessage              *deviceConnected;
///< 是否连接，连接后对象不为nil
@property (nonatomic, assign) PLConnectMode                 mode;
///< 连接方式 ble还是mfi
```

- 枚举

```
typedef NS_ENUM(NSInteger, PLConnectMode) {
    /// 未知类型
    PLConnectModeUnconnect  = 0,
    /// BLE
    PLConnectModeBLE        = 1,
    /// MFI
    PLConnectModeMFI        = 2,
};

typedef NS_ENUM(NSInteger, PLBluetoothState) {
    /// 未授权，请前往系统设置授权
    PLBluetoothStateUnauthorized  = 0,
    /// 蓝牙未开
    PLBluetoothStatePoweredOff  = 1,
    /// 正常
    PLBluetoothStatePoweredOn  = 2,
};

/// 连接错误类型
typedef NS_ENUM(NSUInteger,PLConnectError){
    PLConnectErrorTimeout = 0,
    PLConnectErrorNoAccessory,
    PLConnectErrorNoPeripheral,
    PLConnectErrorValidateTimeout,
    PLConnectErrorUnknownDevice,
    PLConnectErrorSystemFailed,
    PLConnectErrorValidateFailed,
    PLConnectErrorCodeCheckFailed
};

/// mfi返回错误
typedef void(^PLAccessoryPickerCompletion)(NSError *error);
```

- 接口

```
/** 单例对象 */
+ (PLConnectInterface *)shared;

/** 开启扫描 ble */
- (void)scanBluetooth;

/** 开启扫描 mfi */
- (void)scanAccessory;

/** 停止扫描，连接成功会自动停止扫描 */
- (void)stopScanBluetooth;

/**
 * 连接打印机
 * @param printer 外设对象
 */
- (void)connectPrinter:(PLPrinterMessage *)printer;

/** 断开打印机 */
- (void)disconnect;

/** 获取手机的蓝牙状态 */
- (PLBluetoothState)getBluetoothStatus;

/** 注册MFI通知 */
- (void)registerEAAccessoryManagerNotifications;

/** 移除MFI通知 */
- (void)removeEAAccessoryManagerNotifications;

/** 展示可连接的配件列表，该方法会弹出连接窗口 */
- (void)showBluetoothAccessorys:(NSString *)name completion:
(PLAccessoryPickerCompletion)completion;
```

2.1 PLPrinterMessage

- 作用

```
用作连接对象、外设属性
```

- 属性

```
@property (nonatomic, copy) NSString        *name;  ///< 外设名称
@property (nonatomic, copy) NSString        *mac;   ///< mfi和ble的可能不一样
@property (nonatomic, copy) NSString        *identifier; ///< 外设的唯一标识符
@property (nonatomic, strong) NSNumber      *distance;  ///< 距离
@property (nonatomic, strong) EAAccessory   *accessory; ///< mfi连接对象
@property (nonatomic, strong) CBPeripheral  *peripheral; ///< ble连接对象
@property (nonatomic, assign) PLConnectMode  mode;      ///< 连接类型
```

2.2 PLPrinterInterfaceDelegate

- 作用

  发现设备、连接状态、监听系统设置连接和断开回调

- 接口

```
/** BLE:发现附近蓝牙设备 */
- (void)didDiscoverBLEPrints:(NSArray<PLPrinterMessage *> *)devices;

/** MFI:发现附近配件 */
- (void)didDiscoverMFIPrints:(NSArray<PLPrinterMessage *> *)devices;

/** 连接失败 */
- (void)didConnectFail:(NSError *)error;

/** 连接成功 */
- (void)didConnectSuccess;

/** 断开连接 */
- (void)didDisconnect;

/** 监听系统设置有配件连接，注册mfi通知后才会回调 */
- (void)observerAccessoryDidConnectedNotification:(PLPrinterMessage *)device;

/** 监听系统设置有配件断开，注册mfi通知后才会回调 */
- (void)observerAccessoryDidDisconnectNotification:(PLPrinterMessage *)device;
```

三、数据调度模块

3.0 PLDataDispatcher

> 1.该类是处理数据发送和接收蓝牙数据
>
> 2.A4的打印机已经通过PLDataResolver解析协议在SDK里面处理
>
> 3.其他机型通过`receiveBluetoothDataBlock`这个接口返回的数据进行解析
>
> 4.数据发送除了A4图片、含有高清灰阶图片、固件升级有特定接口外，其他数据都可通过`writeData`接口发送
>
> 5.对于A4机型，通过resolver属性的resolvedBlock得到解析模型，前提是需要初始化`PLMT800Resolver`类

- 属性

```
@property (nonatomic, strong) id<PLDataResolver>          _Nullable  resolver; ///< 协议属性
@property (nonatomic, copy) PLReceiveDataAction           _Nullable  receiveDataBlock; ///< 接收蓝牙数据
@property (nonatomic, copy) PLAutoMT800StateAction        _Nullable  autoA4StateBlock; ///< A4系列自动回传状态
```

- 接口

```
+ (PLDataDispatcher *)shared;

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

/** 发送数据 */
- (void)writeData:(NSData *)data progress:(void(^_Nullable)(NSProgress *))block fail:(void(^_Nullable)(void))failBlock;

/** A4 打印图片 */
- (void)writeA4ImageDatas:(NSMutableArray *)datas progress:(void(^_Nullable)(NSProgress *))block fail:(void(^_Nullable)(void))failBlock;

/** 固件升级数据 */
- (void)writeFirmwareData:(NSData *)data progress:(void(^_Nullable)(NSProgress *))block fail:(void(^_Nullable)(void))failBlock;

/** 接收蓝牙返回数据回调 A4机型不需要使用这个回调 */
- (void)receiveBluetoothDataBlock:(PLReceiveDataAction)receiveDataBlock;

/** A4自动返回异常状态回调 */
- (void)autoA4DeviceStateBlock:(PLAutoMT800StateAction)automaticStateBlock;
```

- 协议

```
/// MT800解析协议
@protocol PLDataResolver <NSObject>

@property (nonatomic, copy) void (^_Nullable resolvedBlock)(id result);

- (void)readDeviceInput:(NSData *)data;

@end
```

3.1 PLResolverCentral

> A4机型数据解析处理

- PLCommonResolveModel

  > 解析模型，已经通过`PLMT800ResolverModel`继承，开发者使用`PLMT800ResolverModel`模型即可

  - 属性

```
///< \~chinese 是否丢弃 \~english discardable
@property (nonatomic, assign) BOOL isDiscardable;

///< \~chinese 数据内容 \~english content
@property (nonatomic, strong) NSData *data;

///< \~chinese 消耗的数据量 \~english data length costed
@property (nonatomic, assign) NSInteger cost;

///< \~chinese 源数据 \~english raw data
@property (nonatomic, strong) NSData *rawData;
```

- PLDataRouterBlock

  > 通过路由表得到公共模型，开发者不用理会

- PLDataRouter

  > 解析路由表，存储键值对，开发者不用理会

- PLCommonResolver

  > 解析类，得到解析公共模型，已经由`PLMT800Resolver`继承，开发者初始化PLMT800Resolver类即可

3.2 PLMT800Resolver

> 继承解析模型和解析公共类

- PLMT800ResolverModel

  > A4解析模型

  - 属性type类型

```
typedef NS_ENUM(NSUInteger, PLMT800CmdType) {
    PLMT800CmdTypePrinterInfo,                ///< 打印机信息
    PLMT800CmdTypeCarbonRibbonBrand,          ///< 碳带品牌
    PLMT800CmdTypeCarbonRibbonRemainCount,    ///< 碳带余量
    PLMT800CmdTypeSn,                         ///< 序列号
    PLMT800CmdTypeFirmwareVersion,            ///< 固件版本
    PLMT800CmdTypeClearBuffer,                ///< 清空缓存
    PLMT800CmdTypeAutoStatus,                 ///< 自动回传状态
    PLMT800CmdTypeImageSliceAck,              ///< 图片分包ACK
    PLMT800CmdTypeFirmwareSliceAck,           ///< 固件分包ACK
    PLMT800CmdTypeCommonSet,                  ///< 设置指令（浓度 关机时间指令 打印机型号）
    PLMT800CmdTypeCommonGet,                  ///< 获取指令（浓度 关机时间指令 打印机型号）
    PLMT800CmdTypeAuthorizedKey,              ///< 授权key获取，暂用不上
    PLMT800CmdTypeAuthorizedRibbon,           ///< 授权里程，暂用不上
    PLMT800CmdTypeCarbonRibbonProperty,       ///< 碳带相关信息，暂用不上
    PLMT800CmdTypeNotResolvedData = 0xff,     ///< 失败
};
```

- PLMT800Resolver

  > 解析类，需要初始化该类才能解析，建议是在连接时初始化

四、指令接口模块

4.0 PLCmdMTGenerator

> 详情看对应指令文档

4.1 PLEncode

> 解码和编码，默认kCFStringEncodingGB_18030_2000

4.2 PLEnumList

> 定义一些枚举

五、位图数据模块

5.0 PLBitmapManager

> 该类用于生成打印机识别的图片数据

- 枚举（图片效果和是否压缩）

```
typedef NS_ENUM(NSInteger,PLBitmapCompressMode) {
    ///图片不压缩
    PLBitmapCompressModeNone = 0,
    ///图片压缩
    PLBitmapCompressModeLZO = 48
};

typedef NS_ENUM(NSInteger, PLBitmapMode) {
    /// 黑白图片，二值算法
    PLBitmapModeBinary = 0,
    /// 扩散抖动算法
    PLBitmapModeDithering = 1,
    /// 聚集抖动
    PLBitmapModeCluster = 2,
    /// 灰阶算法
    PLBitmapModeGray = 3
};
```

- 接口

```
/// 生成二值抖动图片数据
/// @param image 图片
/// @param watermark 是否增加水印
/// @param mode 图片效果，这边如果选择灰阶模式，那么则默认SDK处理图片
/// @param compress 压缩模式
/// @param package 是否需要分包，一般都需要分包
+ (NSData *)generateGenralDataWithImage:(UIImage *)image watermark:(BOOL)watermark
mode:(PLBitmapMode)mode compress:(PLBitmapCompressMode)compress package:
(BOOL)package;
```

```
/// 灰阶高清数据，PLCmdPoooliGenerator再调用appendGrayHDImageBitmap
/// @param image 图片，图片高度不能超过5000
/// @param watermark 是否需要增加水印
/// @param gamma 伽码系数，调节明暗度
/// @param factor 锐化程度，默认14，取值1-20，值越大锐化程度越低
/// @param gammaType 是否需要调节伽马系数，YES表示需要，NO表示不需要，200dpi的机型建议选择NO
+ (NSData *)generateHDDataWithImage:(UIImage *)image watermark:(BOOL)watermark
fGamma:(CGFloat)gamma factor:(NSInteger)factor gammaType:(BOOL)gammaType;


/// 预览图
/// @param image 原图图片
/// @param mode 预览的图片模式
/// @param gamma 伽码系数，调节明暗度,范围：0.6-2.4，默认值1.2，只对灰阶图起作用
/// @param gammaType 是否需要调节伽马系数，YES表示需要，NO表示不需要，只对灰阶图起作用
+ (UIImage *)generateRenderingWithImage:(UIImage *)image mode:(PLBitmapMode)mode
fGamma:(CGFloat)gamma gammaType:(BOOL)gammaType;


/// 生成位图数据(该数据需要指令类的对应接口再处理，不可直接下发)
/// @param image 图片
/// @param watermark 是否增加白色水印
/// @param mode 图片效果
+ (NSData *)generateBitmapDataWithImage:(UIImage *)image watermark:(BOOL)watermark
mode:(PLBitmapMode)mode;

/// 锐化后的图片
/// @param image 原图图片
+ (UIImage *)imageToSharpen:(UIImage *)image;


/// 生成透明图片
/// @param image 原图图片
+ (UIImage *)imageToTransparent:(UIImage *)image;


/// 自定义阈值的黑白图数据
+ (NSData *)printableBinaryData:(UIImage *)image threshold:(Byte)threshold
compress:(PLBitmapCompressMode)compress;


/// 重绘打印图片
/// @param image 原图
/// @param destSize 目标大小 A4->2336x3288
/// @param aotuRotate 是否自动旋转
+ (UIImage *)drawPrintImage:(UIImage *)image destSize:(CGSize)destSize aotuRotate:
(BOOL)aotuRotate;
```

六、PDF转图片

6.0 PLPDFConverter

> 用于PDF转图片

- 属性

```
@property (nonatomic, assign, readonly) NSInteger pageCount;
```

- 接口

```
/// 初始化文件路径
/// @param path 文件路径
/// @param autoRotate 是否需要自动旋转，false表示不用旋转，true表示要自动旋转
- (instancetype)initWithPath:(NSURL *)path autoRotate:(BOOL)autoRotate;

/// 将某页PDF转图片
/// @param index 下标
/// @param size 图片大小
- (UIImage * _Nullable)imageAtIndex:(NSInteger)index size:(CGSize)size;
```

七、更新历史

- **v2.0.0**

```
更新时间:2020-12-30
1.SDK重构
```

- **v2.0.1**

```
更新时间:2021-07-19
1.优化SDK
2.优化图片处理类的预览图片接口
```

- **v2.0.2**

```
更新时间:2021-08-31
1.增加PDF转图片功能
2.增加客户代码授权
3.SDK文件夹中增加图片增强库，增强图片效果(注：由于A4的尺寸较大，用增强库时需要耗时数秒)
```

- **2.0.3**

```
更新时间:2021-11-24
1.优化客户代码验证,在连接之前要先初始化授权Key
```

- **2.0.4**

```
更新时间:2021-11-29
1.增加日志，默认是false(不写入)
```

- **2.0.5**

```
更新时间:2021-12-03
1.增加设置和获取纸张类型
```

- **2.0.6**

```
更新时间:2021-12-11
1.图片处理类PLBitmapManager增加重绘打印图片功能
```