

1. SDK Import Instruction
2. SDK Open Class Instruction
 - 2.0 PTPrinter
 - 2.1 PTDispatcher
 - 2.2 PTCommandCPCL
 - 2.3 PTCommandESC
 - 2.4 PTCommandTSPL
 - 2.5 PTCommandZPL
 - 2.6 PTEncode
 - 2.7 PTBitmapManager
 - 2.9 PTLabel
 - 2.10 PTOldCommandCPCL
 - 2.11 PTOldCommandTSPL
 - 2.12 PTCommandCommon
3. How to connect peripherals
4. Command Use Cases
 - 4.0 SDK function
 - 4.1 Printing by Template
 - 4.2 SDK printing picture cases (refer to demo)

1. SDK Import Instruction

- 1.Import the PrinterSDK.xcframework from zip to your project
- 2.If it is a Bluetooth connection, two Bluetooth permissions need to be added:
Privacy - Bluetooth Always Usage Description
Privacy - Bluetooth Peripheral Usage Description
- 3.When packaging an app, you need to set the option of enable bitcode to no to prevent packaging timeout
- 4.Add -ObjC in Build Settings -> Linking -> Other Linker Flags
- 5.Add #import <CoreBluetooth/CoreBluetooth.h> where Bluetooth is used, but it depends
- 6.Method of warning release: Build Settings -> Documentations Comments -> change YES to NO
- 7.After the optimization of the CPCL and TSPL interfaces, if the old users use this version of the SDK, they will modify a lot. Therefore, the SDK reserves PTOldCommandCPCL and PTOldCommandTSPL which respectively retain the old interface
- 8.Each time data is sent, SDK will set receiveDataBlock, sendSuccessBlock, sendFailureBlock and sendProgressBlock in ptdispatcher to null
- 9.There are two SDKs in the compression package, one with only the real machine architecture for packaging and putting on the shelf, and the other with the real machine and simulator architecture for debugging

2. SDK Open Class Instruction

2.0 PTPrinter

The property class of the peripheral defines the name of the peripheral, MAC address, Bluetooth broadcast package advertisement, Bluetooth UUID, signal strength, WiFi related router and IP, etc. when scanning to the peripheral, connecting to the peripheral or disconnecting from the connection, the transmitted parameter is the attribute class

- Property

```
/// Printer name
@property(strong, nonatomic, readwrite) NSString *name;
/// Printer mac address
@property(strong, nonatomic, readwrite) NSString *mac;
/// Printer Bluetooth module
@property(assign, nonatomic, readwrite) PTPrinterModule module;
/// Broadcast information obtained when Bluetooth is found
@property(strong, nonatomic, readwrite) NSDictionary *advertisement;
/// Bluetooth peripheral UUID
@property(strong, nonatomic, readwrite) NSString *uuid;
/// Signal strength value and unit decibel obtained when peripheral is found
@property(strong, nonatomic, readwrite) NSNumber *rssi;
/// Signal strength level is divided into 0-5 levels
@property(strong, nonatomic, readwrite) NSNumber *strength;
/// Distance calculated by signal strength
@property(strong, nonatomic, readwrite) NSNumber *distance;
/// Peripheral Bluetooth
@property(strong, nonatomic, readwrite) CBPeripheral *peripheral;
/// Peripheral ip address
@property(strong, nonatomic, readwrite) NSString *ip;
/// Port
@property(strong, nonatomic, readwrite) NSString *port;
```

2.1 PTDispatcher

Printer communication class

1. Define the enumeration of connection mode, mobile Bluetooth status, connection failure type, firmware upgrade status, etc
2. Define the block protocol and properties. These block properties are not handled by app
3. Define the interface to interact with the printer

- Enumeration

```

/// Connection mode
typedef NS_ENUM(NSInteger, PTDispatchMode) {

    /// Unknow
    PTDispatchModeUnconnect = 0,

    /// Bluetooth
    PTDispatchModeBLE = 1,

    /// WiFi
    PTDispatchModeWiFi = 2
};

/// Mobile Bluetooth status
typedef NS_ENUM(NSInteger, PTBluetoothState) {
    /// Unauthorized, please go to the system to set up authorization
    PTBluetoothStateUnauthorized = 0,
    /// Bluetooth off
    PTBluetoothStatePoweredOff = 1,
    /// Normal
    PTBluetoothStatePoweredOn = 2,
};

/// Status returned by printer after printing
typedef NS_ENUM(NSInteger, PTPrintState) {

    /// print succeeded
    PTPrintStateSuccess = 0xcc00,

    /// Print failed (paper empty)
    PTPrintStateFailurePaperEmpty = 0xcc01,

    /// Print failed (Cover open)
    PTPrintStateFailureLidOpen = 0xcc02
};

/// Returns the error type of the connection
typedef NS_ENUM(NSInteger, PTConnectError) {

    /// Connect timeout
    PTConnectErrorBleTimeout = 0,

    /// Get service timeout
    PTConnectErrorBleDisvokerServiceTimeout = 1,

    /// Verification timeout
    PTConnectErrorBleValidateTimeout = 2,

    /// Unknown device

```

```

PTConnectErrorBleUnknownDevice            = 3,

/// System error, returned by the core Bluetooth framework
PTConnectErrorBleSystem                    = 4,

/// validation failed
PTConnectErrorBleValidateFail              = 5,

/// WiFi connection timeout
PTConnectErrorWifiTimeout                  = 6,

/// socket error
PTConnectErrorWifiSocketError              = 7
};

/// Return firmware upgrade error
typedef NS_ENUM(NSInteger, PTUpgradeFirmwareState) {

    /// Upgraded successfully
    PTUpgradeFirmwareStateSuccess            = 0,

    /// Upgrade failed, wrong data length
    PTUpgradeFirmwareStateFailureDataLengthError,

    /// Upgrade failed, validation failed
    PTUpgradeFirmwareStateFailureValidateFail,

    /// Upgrade failed, write timeout
    PTUpgradeFirmwareStateFailureWriteTimout,

    /// Upgrade failed, package number error
    PTUpgradeFirmwareStateFailurePackageNumberError,

    /// Upgrade failed, package length error
    PTUpgradeFirmwareStateFailurePackageLengthError,

    /// Upgrade failed, write failed
    PTUpgradeFirmwareStateFailureWriteFail,

    /// Upgrade failed,
    PTUpgradeFirmwareStateFail
};

```

- Property

```

@property (assign, nonatomic) PTDispatchMode mode;
//This property indicates the printer object after connection

```

```

@property (strong, nonatomic, readwrite) PTPrinter
    *printerConnected;
@property (copy, nonatomic, readwrite) PToSendSuccessParameterBlock
    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;

```

- Method

```

/// Create singleton
+ (instancetype)share;

/// Send data
- (void)sendData:(NSData *)data;

/// Pause send
- (void)pausewriteData;

/// Continue sending
- (void)resumewriteData;

/// Scan Bluetooth
- (void)scanBluetooth;

```

```

/// Stop scanning Bluetooth, SDK will automatically stop scanning after connecting
successfully
- (void)stopScanBluetooth;

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

/// Get all found printers. Every time a new printer is found or called every
three seconds, an array of printers is returned
- (void)whenFindAllBluetooth:(PTPrinterMutableArrayBlock)bluetoothBlock;

/// When a Bluetooth block is found, the core Bluetooth framework will call every
time it finds a printer and return an array of printers
- (void)whenFindBluetooth:(PTPrinterParameterBlock)bluetoothBlock;

/// Get the RSSI signal strength of Bluetooth
- (void)whenReadRSSI:(PTNumberParameterBlock)readRSSIBlock;

/// Connect the printer and enter the printer object
- (void)connectPrinter:(PTPrinter *)printer;

/// Disconnect printer
- (void)disconnectPrinter:(PTPrinter *)printer;

/// This method is invoked after the connection is successful, the code scanning
device will be stopped
- (void)whenConnectSuccess:(PTEmptyParameterBlock)connectSuccessBlock;

/// This method is invoked after connect failed
- (void)whenConnectFailureWithErrorBlock:
(PTBluetoothConnectFailBlock)connectFailBlock;

/// Disconnected block. This method will be invoked after the disconnectPrinter are
invoked to disconnect the printer
- (void)whenUnconnect:(PTUnconnectBlock)unconnectBlock;

/// Block of data sent successfully. When the data is sent, this method is invoked
- (void)whenSendSuccess:(PTSendSuccessParameterBlock)sendSuccessBlock;

/// Block of data sending failed block
- (void)whenSendFailure:(PTEmptyParameterBlock)sendFailureBlock;

/// Block of data sending progress
- (void)whenSendProgressUpdate:(PTNumberParameterBlock)sendProgressBlock;

/// When receiving the data blocked by the printer, for example, getting the
printer name is returned through this method
- (void)whenReceiveData:(PTDataParameterBlock)receiveDataBlock;

```

```

/// Block of receive print status. Before using this method, it is necessary to
ensure that the printer has turned on the switch of status block, such as the
cpclTurnOnPrintStatusCallback method in the CPCL command and the
turnOnPrintStatusCallback method in ESC command
- (void)whenUpdatePrintState:(PTPrintStateBlock)printStateBlock;

/// Set Bluetooth connection timeout
- (void)setupBleConnectTimeout:(double)timeout;

/// Set the peripheral filter, filter in the block of the printer device, and
return to the desired model
- (void)setupPeripheralFilter:(PTPeripheralFilterBlock)block;

/// Sign up for the Bluetooth center, which is compatible with your own core
Bluetooth framework
- (void)registerCentralManager:(CBCentralManager *)manager delegate:
(id<CBCentralManagerDelegate>)delegate;

/// Unregister delegate
- (void)unregisterDelegate;

/// Block of upgrade firmware status
- (void)whenUpgradeFirmwareStateBlock:
(PTUpgradeFirmwareStateBlock)upgradeFirmwareStateBlock;

/// The Bluetooth status of the mobile phone. In order to get the Bluetooth status
of the mobile phone accurately, you need to initialize PTDispatcher when the app
is started
- (PTBluetoothState)getBluetoothStatus;

/// SDK build time
- (NSString *)SDKBuildTime;

```

2.2 PTCommandCPCL

CPCL command interface class, details [in](#) the CPCL command interface document

2.3 PTCommandESC

ESC command interface class, details [in](#) the ESC command interface document

2.4 PTCommandTSPL

TSPL command interface class, details [in](#) the TSPL command interface document

2.5 PTCommandZPL

ZPL command interface class, details [in](#) the ZPL command interface document

2.6 PTEncode

Encoding class, the default is kCFStringEncodingGB_18030_2000

- Interface

```
/// Code, GBK by default
+ (NSData *)encodeDataWithString:(NSString *)string;
+ (NSData *)encodeDataWithString:(NSString *)string encodingType:
(CFStringEncoding)encodingType;

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

2.7 PTBitmapManager

Image processing class, which is usually processed in SDK

- Enumeration

```
typedef NS_ENUM(NSInteger, PTBitmapCompressMode) {
    /*! *~chinese 不压缩 *~english None */
    PTBitmapCompressModeNone = 0,
    /*! *~chinese LZ0压缩算法 *~english LZ0 compress */
    PTBitmapCompressModeLZ0 = 48
};

typedef NS_ENUM(NSInteger, PTBitmapMode) {
    /*! *~chinese 黑白二值图像 *~english Binary */
    PTBitmapModeBinary = 0,
    /*! *~chinese 扩散抖动 *~english Diffusion dithering algorithm */
    PTBitmapModeDithering = 1,
```



```

    /*! *\~chinese 聚集抖动算法 *\~english Aggregate dithering algorithm */
    PTBitmapModeCluster = 2
};

```

- Interface

```

/// Generate binary jitter image data
/// @param image image
/// @param mode Picture effect
/// @param compress Compression mode
/// @param package Do you need to subcontract
/// @param reverse Whether the data needs to be reversed, eg: TSPL pictures need
to be reversed
+ (NSData *)generateGenralDataWithImage:(UIImage *)image mode:(PTBitmapMode)mode
compress:(PTBitmapCompressMode)compress package:(BOOL)package reverse:
(BOOL)reverse;

```

2.9 PTLabel

Using the shipping label template, you only need to fill in the corresponding data to send and print out the list

- Tips:

1. When printing with template, you must fill in all items in the template provided by us
2. If there is an empty data item, for example, if the claim value is empty, an empty string of @ '' will be entered
3. For different templates, the data items to be filled in are different. Please refer to our example for details

- Property

```

@property(strong, nonatomic, readwrite) NSString *express_company;    // Express
company
@property(strong, nonatomic, readwrite) NSString *delivery_number;    // Delivery
number
@property(strong, nonatomic, readwrite) NSString *order_number;    // Order
number

```

```

@property(strong, nonatomic, readwrite) NSString *distributing;      //
Distributing
@property(strong, nonatomic, readwrite) NSString *barcode;           // Barcode
@property(strong, nonatomic, readwrite) NSString *barcode_text;      // Characters
below barcode
@property(strong, nonatomic, readwrite) NSString *qrcode;             // QR Code
@property(strong, nonatomic, readwrite) NSString *qrcode_text;       // Characters
below QR Code

@property(strong, nonatomic, readwrite) NSString *receiver_name;      // Receiver
name
@property(strong, nonatomic, readwrite) NSString *receiver_phone;     // Receiver
phone
@property(strong, nonatomic, readwrite) NSString *receiver_address;   // Receiver
address
@property(strong, nonatomic, readwrite) NSString *receiver_message;   // Receiver
message

@property(strong, nonatomic, readwrite) NSString *sender_name;        // Sender name
@property(strong, nonatomic, readwrite) NSString *sender_phone;       // Sender
phone
@property(strong, nonatomic, readwrite) NSString *sender_address;     // Sender
address
@property(strong, nonatomic, readwrite) NSString *sender_message;     // Sender
message

@property(strong, nonatomic, readwrite) NSString *article_name;       // Article
name
@property(strong, nonatomic, readwrite) NSString *article_weight;     // Article
weight

@property(strong, nonatomic, readwrite) NSString *amount_declare;     // Declare
amount
@property(strong, nonatomic, readwrite) NSString *amount_paid;        // Paid amount
@property(strong, nonatomic, readwrite) NSString *amount_paid_advance; // Paid
advance amount

```

- Interface

```

/// Data generated from template data and sent to printer
/// @param filePath Tempalt file path
- (NSData *)datawithSourceFile:(NSString *)filePath;

/// Generate data for the TSPL command
- (NSData *)datawithTSPL;

```

```

/// Data generated from template data and sent to printer
/// @param source Source
/// @param labelDict Key defined by template
/// @param orderDetails Order details
- (NSData *)getTemplateData:(NSString *)source labelDict:(NSDictionary *)labelDict
orderDetails:(NSArray *)orderDetails;

/// Data generated from template data and sent to printer
/// @param source Source
/// @param labelDict Key defined by template
- (NSData *)getTemplateData:(NSString *)source labelDict:(NSDictionary
*)labelDict;

```

2.10 PTOldCommandCPCL

This class is an old CPCL interface before SDK3.0.0

2.11 PTOldCommandTSPL

This class is an old TSPL interface before SDK3.0.0

2.12 PTCommandCommon

This class is a common command interface. The following three interfaces need printer support to return data

```

/// Get the printer model, return the data format: 51333142 5400
- (void)getPrinterModelName;

/// Obtain the firmware version number of the printer. The return format of the version
number is x.xx.xx or x.x.x 【such as 1.01.01 or 1.0.3】
- (void)getPrinterFirmwareVersion;

/// OTA Bluetooth firmware upgrade, which requires printer support
- (void)updateOTABLEFirmwareWithData:(NSData *)data;

```

3. How to connect peripherals

Several methods are used to connect peripherals, please refer to the Demo for details.

- BLE

```

//Swift5:

//Get the Bluetooth status This interface needs to be initialized in AppDelegate
PTDispatcher.share()
PTDispatcher.share().getBluetoothStatus()

//Start scanning Bluetooth
PTDispatcher.share().scanBluetooth()

//Scanned Bluetooth, returned as an array
PTDispatcher.share()?.whenFindAllBluetooth({ (array) in

})

//Turn off scanning, it will automatically close after connecting
PTDispatcher.share().stopScanBluetooth()

//Connect printer
PTDispatcher.share().connect(printer)

//Disconnect printer
PTDispatcher.share().disconnectPrinter(PTDispatcher.share().printerConnected)

//Connected successfully
PTDispatcher.share().whenConnectSuccess {
    //processing
}

//Connection failed
PTDispatcher.share().whenConnectFailureWithErrorBlock { (error) in
    //processing
}

```

- WiFi

```

let printer = PTPrinter()
printer.ip = "xxx.xxx.xxx.xxx"
printer.module = .wifi
printer.port = "9100"

//Connect printer
PTDispatcher.share().connect(printer)

//Disconnect printer
PTDispatcher.share().disconnectPrinter(PTDispatcher.share().printerConnected)

//Connect printer

```

```
PTDispatcher.share().whenConnectSuccess {
}
//Connect failed
PTDispatcher.share().whenConnectFailureWithErrorBlock { (error) in
}
```

- Send data

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

})
//Sent successfully
PTDispatcher.share().whenSendSuccess {
}
//Sending failed
PTDispatcher.share().whenSendFailure { [weak self] in

}
// Receive data returned by Bluetooth
PTDispatcher.share().whenReceiveData { (temp) in

}

/// POS ESC command support
PTDispatcher.share().whenESCPrintSuccess { _ in

}
```

4. Command Use Cases

4.0 SDK function

- Print Format Barcode
- Print 2D code
- Printed text
- Print images (binary and gray-scale dithering)
- Printing tickets

Through this Demo, you can learn that:

- How to import, link and use the 'PrinterSDK.framework' framework
- How to communicate through Bluetooth 4.0 and portable BLE Bluetooth printer
- How to communicate through WiFi and mobile WiFi printers

- How to use the basic commands in the printer command set and subpackage them into functions according to their own needs

4.1 Printing by Template

The type of the shipping label has been pre-edited. You only need to fill the items in corresponding, and a shipping label can be printed out. Take Shentong Express as an example, please refer to Demo for detail.

1.Fill data

```
// Description:
// 1. Initialize an NSMutableDictionary and insert the corresponding data under the
// corresponding key value. The key value must be used in the following example.
// 2. If a data item has no data, then it needs to be set to an empty string @"", such
// as [templateDict setObject:@"" forKey:kCollection];
NSMutableDictionary *templateDict = [[NSMutableDictionary alloc] init];

[templateDict setObject:@"363604310467" forKey:LTBarcode];
[templateDict setObject:@"Changning District, Shanghai City, Shanghai"
forKey:LTDistributing];

[templateDict setObject:@"Shendatong" forKey:LTRceiver];
[templateDict setObject:@"13826514987" forKey:LTRceiverContact];
[templateDict setObject:@"Room 306,No.12, Gonghe Community, Lane 4719, Gonghexin Road,
Baoshan District, Shanghai City" forKey:LTRceiverAddress];

[templateDict setObject:@"Kuaixiaobao" forKey:LTSender];
[templateDict setObject:@"13826514987\r\n" forKey:LTSenderContact];
[templateDict setObject:@"Room 306,No.12, Gonghe Community, Lane 4719, Gonghexin Road,
Baoshan District, Shanghai City" forKey:LTSenderAddress];

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

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

2.Read-in template

You need to drag the template file into the project when you use it. The template file is a TXT plain text file.

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

3.Combine templates and data

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

4.Send data

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

4.2 SDK printing picture cases (refer to 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: PTBitmapCompressMode.none)
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()
//This is the millimeter of paper, not the pixel
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~It is recommended to use this one. It can be compressed
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()

/// Download
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)
/// Print
cmd.xg_RecallGraphic(withSourceDevice: PTZplFileLocation.R, imageName: "iZPL",
extension: "GRF", xAxisMagnification: 1, yAxisMagnification: 1)
cmd.fs_FieldSeparator()
cmd.xz_FormatEnd()

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

```