

# iOS SWIFT SDK

## User Manual<sub>(sv2.3.0)</sub>



# iOS Swift Bluetooth/WIFI SDK

## 1. fe\_scan (callback)

- Description: Search for Bluetooth BLE devices and send back
- Parameter:
  - ➔ **callback(devices: [FPeripheral]):** function, return all found BLE devices.

## 2. openPort()

### ★ Bluetooth

openPort (connectDecive, callback)

- Description: Connect after specifying the Bluetooth device and returning the status function
- Parameter:
  - ➔ **connectDecive:** FPeripheral type , Specify the connected Bluetooth Peripheral device
  - ➔ **callback(msg: String):** function , return connection status

Return String
"disconnected"
"connecting"
"connected"
"disconnecting"
"time_out"

### ★ WIFI

openPort(IP, port, callback)

- Description: After specifying the IP address and port of the printer and returning the status function , open output port
- Parameter:
  - ➔ **IP:** String type , Specify the IP address of the connection , Example: "192.168.1.109"
  - ➔ **port:** Int type , Specifies the port for the connection , Example: 9100
  - ➔ **callback(msg: String):** function , return connection status

Return String
"connecting"
"connected"
"time_out"

### 3. closePort()

#### ★ Bluetooth

closePort (connectDecive, callback)

■ Description: bluetooth disconnected

■ Parameter:

→ **connectDecive**: CBPeripheral type , Specify the connected Bluetooth Peripheral device

→ **callback(msg: String)**: function , Return disconnection message "disconnected"

#### ★ WIFI

closePort (callback)

■ Description: close output port

■ Parameter:

→ **callback(msg: String)**: function , Return disconnection message "disconnected"

### 4. retrieveDevice(uuid, callback)

■ **For Bluetooth**

■ Description: Input the UUID, if it can find the device with the same UUID, and then the function will send back the device information.

■ Parameter:

Parameter	Type	Description
uuid	UUID	The specific UUID user wants.
callback	function	If it can scan and get the device with the UUID, it will return the device information(FEPeripheral). If it can find the device, will return nil.

## 5. **setup(width, height, speed, density, sensor, sensorDistance, sensorOffset, callback)**

- Description: Setup label width , label height, print speed, density, sensor type, gap/blackmark vertical distance and gap/blackmark offset distance.
- Parameter:

Parameter	Type	Description
width	Int	Set up label width; unit: mm.
height	Int	Set up label height; unit: mm.
speed	Int	Set up print speed (1~15: print speed at 1"~15"/sec). Selectable print speeds depend on different printer models, and maximum speed is 15"/sec.
density	Int	Set up print density(0~15);If the number is bigger, the printout will be darker.
sensor	Int	Set up the sensor type for the media. 0: Gap sensor 1: Black mark sensor
sensorDistance	Int	Set up vertical gap height of the gap/black mark; unit: mm
sensorOffset	Int	Set up offset distance of the gap/black mark, unit: mm, this parameter is set to 0 when the general label is used.
callback	function	Return when not connected"Connection doesn't exist."

## 6. **clearBuffer()**

- Description: Clear the image buffer.

## 7. **printBarcode( x, y, type, height, readable, rotation, narrow, wide, content, callback)**

- Description: Use built-in barcode formats to print barcodes.
- Parameter:

Parameter	Type	Description
x	Int	Specify the x-coordinate bar code on the label, Unit: dot
y	Int	Specify the y-coordinate bar code on the label,

		Unit: dot
type	String	Set up Code Type , <a href="#">refer to Appendix</a>
height	Int	Set up bar code height (in dots)
readable	Int	Set up whether to print human recognizable interpretation (text) or not. 0: Do not print 1: Print barcode document left 2: Print barcode code document in the center 3: Print barcode document right
rotation	Int	Set up barcode rotation 0 : No rotation 90 : Rotate 90 degrees clockwise 180 : Rotate 180 degrees clockwise 270 : Rotate 270 degrees clockwise
narrow	Int	Set up narrow bar ratio (in dots), <a href="#">refer to Appendix</a>
wide	Int	Set up wide bar ratio (in dots), <a href="#">refer to Appendix</a>
content	String	Content of barcode. Please note that the maximum number of digits of bar code content. <a href="#">refer to Appendix</a>
callback	function	Return when not connected"Connection doesn't exist."

## 8. printQRCode(x, y, eccLevel, cellWidth, rotation, content, callback)

- Description: Use built-in QR code formats to print QR code
- Parameter:

Parameter	Type	Description
x	Int	The upper left corner x-coordinate of the QR code
y	Int	The upper left corner y-coordinate of the QR code
eccLevel	String	Error correction recovery level L: 7%, M: 15%, Q: 25%, H: 30%
cellWidth	Int	QR code shape width 1~10
rotation	Int	Set up QR code rotation: 0: 0 degree, 90: 90 degree, 180: 180 degree, 270: 270 degree
content	String	The encodable character set is described as below,

		<p>*Encodable character set:</p> <p>1) Numeric data: (digits 0~9)</p> <p>2) Alphanumeric data</p> <p>Digits 0-9</p> <p>Upper case letters A-Z</p> <p>Nine other characters: space, \$ % * + - . / : )</p> <p>3) 8-bit byte data</p> <p>JIS 8-bit character set (Latin and Kana) in accordance with JIS X 0201</p> <p>4) Kanji characters</p> <p>Shift JIS values 8140HEX –9FFCHEX and E040HEX – EAA4 HEX. These are values shifted from those of JIS X 0208. Refer to JIS X 0208 Annex 1</p> <p>Shift Coded Representation for detail.</p>
callback	Function	Return when not connected"Connection doesn't exist."

## 9. formFeed()

- Description: Feed label to the top of next label.

This function should be used after setup function.

## 10.noBackFeed()

- Description: Set the paper not to back feed.

## 11.sendCommand (\_ string)

- Description: Sent built-in commands to the printer.
- Parameter:
  - ➔ **string**: String type, Refer to TSPL programming manual commands for details.

## 12. sendByteCmd(cmdData)

- Description: Sent built-in commands in byte format to the printer.
- Parameter:
  - **cmdData**: Data type

## 13. printFont(x, y, fontName, rotation, x\_scale, y\_scale, content, callback)

- Description: Use printer built-in fonts to print
- Parameter:

Parameter	Type	Description
x	Int	The x-coordinate of the text
y	Int	The y-coordinate of the text
fontName	String	Built-in font type  1: 8*/12 dots  2: 12*20 dots  3: 16*24 dots  4: 24*32 dots  5: 32*48 dots  TST24.BF2: Traditional Chinese 24*24  TST16.BF2: Traditional Chinese 16*16  TSS24.BF2: Simplified Chinese 24*24  TSS16.BF2: Simplified Chinese 16*16
rotation	Int	The rotation angle of text  0 : No rotation

		90: degrees, in clockwise direction  180 : degrees, in clockwise direction  270 : degrees, in clockwise direction
x_scale	Int	Horizontal multiplication, Available factors: 1~10
y_scale	Int	Vertical multiplication, Available factors: 1~10
content	String	Content of text string
callback	Function	Return when not connected"Connection doesn't exist."



#### 14. printLabel(set, copy)

- Description: Print the label format currently stored in the image buffer.

- Parameter:

➔ **set**: Int type , Specifies how many sets of labels will be printed.

$$1 \leq \text{set} \leq 999999999$$

➔ **copy**: Int type , Specifies how many copies should be printed for each particular label set.

$$1 \leq \text{copy} \leq 999999999$$

#### 15. downloadPCX(filePath, filename, callback)

- Description: Download the monochrome PCX file to main board flash memory

- Parameter:

➔ **filePath**: URL type , Bring in the location of the PCX file

➔ **fileName**: String type , PCX file name (must include file extension)

➔ **callback(msg: String)**: function, Return when not connected"Connection doesn't exist."

#### 16. downloadBMP(filePath, filename, callback)

- Description: Download the monochrome BMP file to main board flash memory

- Parameter:

➔ **filePath**: URL type , Bring in the location of the BMP file



- ➔ **fileName**: String type , BMP file name. (must include file extension)
- ➔ **callback(msg: String)**: function, Return when not connected"Connection doesn't exist."

## 17.getSdkVersion ( )

- Description: Returns the SDK version.
- Parameter: None
- **Return**: SDK version(String)

## 18.printerStatus(callback)

- Description: Obatin the printer status.
- Parameter:
  - ➔ **callback(status: String)**: function, returns the status of the printer
- return string description:

return string	printer status
00	Normal
01	Head opened
02	Paper Jam
03	Paper Jam and head opened
04	Out of paper
05	Out of paper and head opened
08	Out of ribbon
09	Out of ribbon and head opened
0A	Out of ribbon and paper jam
0B	Out of ribbon, paper jam and head opened
0C	Out of ribbon and out of paper
0D	Out of ribbon, out of paper and head opened
10	Pause
20	Printing
80	Other error
	time_out
	Connection doesn't exist.

## 19.setDirectionAndMirror(direction,mirror)

- Description: Set direction and mirror.
- Parameter:

Parameter	Type	Description
direction	int	Set direction, default: 0 0: Top out 1: Bottom out
mirror	int	Set mirror 0: No 1: Yes

## 20. setShift(shiftY)

- Description: Set the vertical displacement distance, when value is positive, it will shift in the printing direction, otherwise, it will shift in opposite direction.
- Parameter:
  - ➔ **shiftY**: int type, vertical displacement distance, the unit is dot.

## 21. printReverse(x\_start, y\_start, x\_width, y\_height)

- Description: Reverse the designated area.
- Parameter:

Parameter	Type	Description
x_start	int	The x-coordinate of the area(in dots)
y_start	int	The y-coordinate of the area (in dots)
x_width	int	The width of area in dots
y_height	int	The height of area in dots

## 22. setOffset(offset)

- Description: Set feed offset(Usually use with peel mode and cut mode)
- Parameter:
  - ➔ **offset**: double type, extra feed offset, the unit is mm

## 23. setCutMode(mode, piece)

- Description: Set cut mode and cut number
- Parameter:

Parameter	Type	Description
mode	int	Set cut mode, default: 1
		0: Backward 1: Forward
piece	int	Set cut number

## 24. **setAfterPrintAction(mode)**

- Description: Set action after print
- Parameter:

Parameter	Type	Description
mode	int	Set action after print, default: 1  0: Normal  1: Tear Mode  2: Peel Mode  3: Cut Mode

## 25. **genericDefault()**

- Description: Initialize the general setting value
- Parameter: none

## 26. **sensorDefault()**

- Description: Initialize the sensor setting value
- Parameter: none

## 27. **switchWifiFrequency(frequency, callback)**

- Description: When using a compatible 5G frequency band WiFi module, it can be used to switch between frequency bands.
- Parameter:
  - ➔ **Frequency**: String type, Module Frequency Setting ;
    - 2.4G: Use the 2.4GHz frequency band.,
    - 5G: Use the 5GHz frequency band.
    - BOTH: Use dual-band frequency
  - ➔ **callback**: function, Return when not connected"Connection doesn't exist."

## 28. printBitmap(imageData, x, y, width, height, mode, callback)

- Description: Print pictures(jpg and png only)
- Parameter:

Parameter	Type	Description
imageData	Data	The Data Type of image.
x	Int	The printing x-coordinate of the area(in dots)
y	Int	The printing y-coordinate of the area(in dots)
width	Int	Modify image width(in dots, can be empty)
height	Int	Modify image height(in dots, can be empty)
callback	Function	Return when not connected"Connection doesn't exist."

## 29. setRealTimeStatus(status, callback)

- Description: Turn on or off the real-time command, The default state of the printer when it is turned on for the first time is off
- Parameter:

Parameter	Type	Description
Status	String	Real-time command open status  0: Disabled  1: Enable
callback	Function	Return when not connected"Connection doesn't exist."

### 30. getRealTimeStatus(callback)

- Description: Read the on or off status of the real-time command
- Parameter:
  - ➔ **callback(msg: String):** function, return the read data or display a timeout
- Return instructions:

Return	Type	Description
0	String	Real-time command shutdown status feedback FB: 0\r\n
1	String	Real-time command open status feedback FB: 1\r\n
time_out	String	Overtime
Connection doesn't exist.	String	Not connected

### 31. getStatusNumber() -> String

- Description: Read real-time status numbers
- Parameter: None
- return numbers description:

Return string	Printer status
00	Normal
01	Head opened
02	Paper Jam
03	Paper Jam and head opened
04	Out of paper
05	Out of paper and head opened
08	Out of ribbon
09	Out of ribbon and head opened
0A	Out of ribbon and paper jam
0B	Out of ribbon, paper jam and head opened
0C	Out of ribbon and out of paper
0D	Out of ribbon, out of paper and head opened
10	Pause
20	Printing
80	Other error

# iOS Swift Bluetooth/WIFI SDK(BT Config WIFI)

## 1. fe\_checkIP(handler)

- Description: Check WIFI IP Address, need a 5-second delay after using the function.
- Parameter:
  - ➔ handler(\_ message: String, \_ IPAddress: String?, \_ error: Error?): function, return the result.
- Return instructions:

Return Parameter	Type	Description
message	String	State description message
IPAddress	String?	option, If it is empty, query failed
error	Error?	option, existed when error

## 2. fe\_btConfigNetwork(isDynamic, ssid, password, staticIp, gw, mask, dns, reconnect, complete)

- Description: Configure WIFI via bluetooth connection, need a 5-second delay after using the function.
- Parameter:

Parameter	Type	Description
isDynamic	Bool	Is it dynamic distribution network? False means static distribution network.
ssid	String	WIFI SSID
password	String?	option, WIFI Password. if the network does not require a password.
staticIp	String	IP address in static network.
gw	String?	option, gateway
mask	String?	option, subnet mask
dns	String?	option, domain name system
reconnect	Bool	Set reconnect or not
complete	function	return results

- Return instructions:

Return Parameter	Type	Description
message	String	State description message
IPAddress	String?	option, If it is empty, query failed
error	Error?	option, existed when error

# iOS Swift Bluetooth/WIFI SDK(RFID-GEN2)

15

## 1. writeUHF(dataFormat, startBlockNo, byteSize, Gen2MemoryBank, dataString, callback)

- Description: Write data to UHF tag memory.
- Parameter:

Parameter	Type	Description
dataFormat	String	Define data format , default is"H"  A: ASCII  H: Hexadecimal
startBlockNo	Int	Secify the 16-bit starting block number , Default : 2
byteSize	Int	Values: 1 to n, where n is the maximum number of bytes for the tag.  Default: 1
Gen2MemoryBank	String	Select Gen2 memory bank  R: Reserved  E: EPC (Default)  T: TID(Tag ID)  U: User
dataString	String	String to write.
callback	Function	Return when not connected"Connection doesn't exist."



## 2. EPCPWD\_Action(action, password)

- Description: Lock or unlock EPC memory with password for UHF GEN2 tag.
- Parameter:

Parameter	Type	Description
action	String	Action type
		U: unlock EPC memory bank
		L: lock EPC memory bank
		O: permanently unlock EPC memory bank
password	String	P: permanently lock EPC memory bank
		password , 8 HEX characters. (0~9, A,B,C,D,E,F)

## 3. TIDPWD\_Action(action, password)

- Description: Lock or unlock TID memory with password for UHF GEN2 tag.
- Parameter:

Parameter	Type	Description
action	String	Action type
		U: unlock TID memory bank
		L: lock TID memory bank
		O: permanently unlock TID memory bank
password	String	P: permanently lock TID memory bank
		password , 8 HEX characters. (0~9, A,B,C,D,E,F)

#### 4. USERPWD\_Action(action, password)

- Description: Lock or unlock USER memory with password for UHF GEN2 tag.
- Parameter:

Parameter	Type	Description
action	String	Action type  U: unlock USER memory bank  L: lock USER memory bank  O: permanently unlock USER memory bank  P: permanently lock USER memory bank
password	String	password ,  8 HEX characters. (0~9, A,B,C,D,E,F)

#### 5. accessPWD\_Action(action, password)

- Description: Lock or unlock access password with password for UHF GEN2 tag.
- Parameter:

Parameter	Type	Description
action	String	Action type  U: unlock the access password*  L: lock the access password*  O: permanently unlock the access password  P: permanently lock the access password  S: Set Password
password	String	password ,  8 HEX characters. (0~9, A,B,C,D,E,F)

## 6. killPWD\_Action(action, password)

- Description: Lock or unlock kill password with password for UHF GEN2 tag.
- Parameter:

Parameter	Type	Description
action	String	Action type  U: unlock the kill password*  L: lock the kill password*  O: permanently unlock the kill password  P: permanently lock the kill password  S: Set Password
password	String	password ,  8 HEX characters. (0~9, A,B,C,D,E,F)

## 7. set\_RFIDPorcedure(tagType, rw\_position, void\_printout, tryEncode\_times, error\_handle, speed, retry\_times, callback)

- Description: Set RFID procedure
- Parameter:

Parameter	Type	Description
tagType	Int	Set Tag type , accepted value: 1~10 ,  For UHF:  1 = ISO 18000 6C/Class 1 Gen2 (Q command)  8 = ISO 18000 6C/Class 1 Gen 2 (default)  For HF  10 = UHF-J
rw_position	Int	Move the media to the specified position on the label, measured in mm rows from the label top, before

		<p>encoding</p> <p>Accept value: 0~9999(dot) , default is 0</p>
void_printout	Int	<p>Set the length of the void printout in vertical (Y axis) dot rows.</p> <p>Accepted values: 0 to label length</p> <p>Default: label length</p>
tryEncode_times	Int	<p>The number of labels that will be attempted in case of read/encode failure. Accepted values: 1 to 10</p> <p>Default: 3</p>
error_handle	String	<p>If an error persists after the specified number of labels are tried, perform this error handling action.</p> <p>N: No action (Default)</p> <p>P: Pause mode</p> <p>E: Error mode</p>
speed	Int	<p>If a label is voided, the speed at which "VOID" will be printed across the label.</p> <p>Accepted value: 2~10(IPS),</p> <p>Default is 2.</p>
retry_times	Int	<p>The retry times of a tag that will be attempted in case of read/encode failure.</p> <p>Accepted value: 0~10 , Default is 6</p>
callback	Function	Return when not connected"Connection doesn't exist."

## 8. `set_RFIDPorcedure(tagType, rw_position, void_printout, tryEncode_times, error_handle, speed, retry_times, dpi, callback)`

- Description: Set RFID procedure
- Parameter:

Parameter	Type	Description
tagType	Int	<p>Set Tag type , accepted value: 1~10 ,</p> <p>For UHF:</p> <p>1 = ISO 18000 6C/Class 1 Gen2 (Q command)</p> <p>8 = ISO 18000 6C/Class 1 Gen 2 (default)</p> <p>For HF</p> <p>10 = UHF-J</p>
rw_position	Int	<p>Move the media to the specified position on the label, default is 0</p> <p>measured in mm rows from the label top, before encoding</p> <p>Accept value:</p> <p>203dpi: 0 ~ 1251 (mm),</p> <p>300dpi: 0 ~ 846 (mm),</p> <p>600dpi: 0 ~ 423 (mm)</p>
void_printout	Int	<p>Set the length of the void printout in vertical (Y axis) dot rows.</p> <p>Accepted values: 0 to label length</p> <p>Default: label length</p>
tryEncode_times	Int	<p>The number of labels that will be attempted in case of read/encode failure. Accepted values: 1 to 10</p> <p>Default: 3</p>
error_handle	String	<p>If an error persists after the specified number of labels are tried, perform this error handling action.</p>

		N: No action (Default)  P: Pause mode  E: Error mode
speed	Int	If a label is voided, the speed at which "VOID" will be printed across the label.  Accepted value: 2~10(IPS),  Default is 2.
retry_times	Int	The retry times of a tag that will be attempted in case of read/encode failure.  Accepted value: 0~10 , Default is 6
dpi	String	Set DPI of the printer 203: 203 dpi  300: 300 dpi  600: 600 dpi
callback	Function	Return when not connected"Connection doesn't exist."

## 9. writeHF(dataFormat, startBlockNo, byteSize, dataString, callback)

- Description: Write data to HF tag memory.
- Parameter:

Parameter	Type	Description
dataFormat	String	Define data format , default is "H"  A: ASCII  H: Hexadecimal
startBlockNo	Int	Secify the 16-bit starting block number , Default : 2
byteSize	Int	Values: 1 to n, where n is the maximum number of bytes for the tag.  Default: 1
dataString	String	Data string
callback	Function	Return when not connected"Connection doesn't exist."

## 10. printFontBlock (x, y ,width, height, fontName, rotation, x\_scale, y\_scale, space, align, content, callback)

- Description: Use printer built-in fonts to print paragraph.
- Parameter:

Parameter	Type	Description
x	Int	The x-coordinate of the text (in dots)
y	Int	The y-coordinate of the text (in dots)
width	Int	The width of block for the paragraph in dots
height	Int	The height of block for the paragraph in dots
fontName	String	<p>Built-in font type</p> <p>1: 8*12 dots</p> <p>2: 12*20 dots</p> <p>3: 16*24 dots</p> <p>4: 24*32 dots</p> <p>5: 32*48 dots</p> <p>TST24.BF2: Traditional Chinese 24*24</p> <p>TST16.BF2: Traditional Chinese 16*16</p> <p>TSS24.BF2: Simplified Chinese 24*24</p> <p>TSS16.BF2: Simplified Chinese 16*16</p>
rotation	Int	<p>The rotation angle of text</p> <p>0 : No rotation</p> <p>90 : degrees, in clockwise direction</p> <p>180 : degrees, in clockwise direction</p> <p>270 : degrees, in clockwise direction</p>
x_scale	Int	Horizontal multiplication, Available factors: 1~10
y_scale	Int	Vertical multiplication, Available factors: 1~10
space	Int	Add or delete the space between lines (in dots)

align	Int	Text alignment  0 : default (Left)  1 : Left  2 : Center  3 : Right
content	String	Data in block. The maximum data length is 4092 bytes.
callback	Function	Return when not connected"Connection doesn't exist."

## 11.readUHF(dataFormat, startBlockNo, byteSize, Gen2MemoryBank, callback)

- Description: Read data from UHF tag memory (R command)
- Parameter:

Parameter	Type	Description
dataFormat	String	Setting callback returned data format ,  A: ASCII  H: Hexadecimal (Default)
startBlockNo	Int	Secify the 16-bit starting block number to read , Default is 0
byteSize	Int	Secify the data lengths to read , default is 1
Gen2MemoryBank	String	Gen2 memory bank ,  R = Reserved  E = EPC  T = TID  U = UESR  Default: E
callback	Function	Return when not connected"Connection doesn't exist."



		Return on connection: <ol style="list-style-type: none"> <li>1. Label data</li> <li>2. "Error code (Refer to the error code page)"</li> <li>3. "time_out"</li> </ol>
--	--	--

■ Return Value(Label information):

dataFormat	Return string(example)
A	Label data is displayed in ASCII (ex: 24051324000103456400)
H	Label data is displayed in Hexadecimal (ex: 3234303531333234303030313033343536343030)

## 12. query\_UHF(dataFormat, PCReturnStatus, CRCReturnStatus, callback)

- Description: Read data from UHF tag memory (Q command)
- Parameter:

Parameter	Type	Description
dataFormat	String	Set up callback return data format , A: ASCII H: Hexadecimal (Default)
PCReturnStatus	Int	enable/disable PC value returned 0: read epc data not include PC value 1: read epc data include PC value
CRCReturnStatus	Int	enable/disable CRC-16 value returned 0: read epc data not include CRC-16 value 1: read epc data include CRC-16 value
callback	Function	Return when not connected"Connection doesn't exist."

		Return on connection: <ol style="list-style-type: none"> <li>1. Label data</li> <li>2. "Error code (Refer to the error code page)"</li> <li>3. "time_out"</li> </ol>
--	--	--

■ Return Value:

Take query\_UHF\_USB( "A", 1, 1) and query\_UHF\_USB( "H", 1, 1) as an example: (Both PC value and CRC-16 are returned)

dataFormat	Return string(example)
A	Label data is displayed in ASCII (ex: 24051324000103456400)
H	Label data is displayed in Hexadecimal (ex: 3234303531333234303030313033343536343030)

Take query\_UHF\_USB( "A", 0, 0) and query\_UHF\_USB( "H", 0, 0) as an example: (PC value and CRC-16 are not returned)

dataFormat	Return string(example)
A	Label data is displayed in ASCII (ex: 0513240001034564)
H	Label data is displayed in Hexadecimal (ex: 30353133323430303031303334353634)

### 13. rfid\_calibration(callback)

- Description: Auto calibration for RFID label
- Parameter: none

### 14. rfidSetupDefault()

- Description: Initialize the RFID setting value
- Parameter: none

## 15. rfid\_labelCalibration(width, height, sensor, sensorDistance, tagType)

- Description: Set up label width, label height, sensor type, gap/black mark vertical distance, tag type, after setting, auto calibration for RFID label
- Only supports Printer:GE2408DE1168
- Parameter:

Parameter	Type	Description
width	CGFloat	Set up label width; unit: mm.
height	CGFloat	Set up label height; unit: mm.
sensor	Int	Set up the sensor type. 0: Gap sensor  1: Black mark sensor
sensorDistance	CGFloat	Set up vertical gap height of the gap/black mark; unit: mm
tagType	Int	Set Tag type , accepted value:1~10 , For UHF: 1 = ISO 18000 6C/Class 1 Gen2 (Q command) 8 = ISO 18000 6C/Class 1 Gen 2 (default) For HF  10 = UHF-J

# iOS Swift Bluetooth/WIFI SDK(RFID-GJB)

27

## 1. set\_GJB\_Pwd\_Action(passwordArea, newPassword, writePassword)

- Description: Set Write/Read/Status/Kill Password to UHF GJB tag.
- Parameter:

Parameter	Type	Description
passwordArea	String	Set password area  K=Kill,  W=Write ( Default),  R=Read,  S=Status
newPassword	String	New password for password area setting above ,  8 HEX characters. (0~9, A,B,C,D,E,F)
writePassword	String	Writing password ,  8 HEX characters. (0~9, A,B,C,D,E,F)

## 2. writeGJB\_UHF(dataFormat, startBlockNo, byteSize, GJBMemoryBank, dataString, writePassword, callback)

- Description: Write data to UHF GJB tag memory
- Parameter:

Parameter	Type	Description
dataFormat	String	Define data format , default is"H"  A: ASCII  H: Hexadecimal
startBlockNo	Int	Secify the 16-bit starting block number , Default : 1
byteSize	Int	Values: 1 to n, where n is the maximum number of

		bytes for the tag.  Default: 1
GJBMemoryBank	String	Select GJB memory bank to write  R = Reserved  E = EPC  T = TID  U = User  2 = User 2  3 = User 3  Default: E
dataString	String	Data string
writePassword	String	Writing password ,  8 HEX characters. (0~9, A,B,C,D,E,F)
callback	Function	Return when not connected"Connection doesn't exist."

### 3. readGJB\_UHF(dataFormat, startBlockNo, byteSize, GJBMemoryBank, readPassword, callback)

- Description: Read data from UHF GJB tag.
- Parameter:

Parameter	Type	Description
dataFormat	String	Set up callback returned data format ,  A: ASCII  H: Hexadecimal (Default)
startBlockNo	Int	Secify the 16-bit starting block number to read , Default is 0
byteSize	Int	Secify the data lengths to read , default is 1
GJBMemoryBank	String	Select GJB memory bank to read ,  E = EPC  T = TID  U = User  2 = User 2  3 = User 3  Default: E
readPassword	String	Reading password ,  8 HEX characters. (0~9, A,B,C,D,E,F)
callback	Function	Return when not connected"Connection doesn't exist."  Return on connection:  1. Label data 2. "Error code (Refer to the error code page)" 3. "time_out"

- Return Value:

dataFormat	Return string(Example)
A	Label data is displayed in ASCII (ex: 24051324000103456400)
H	Label data is displayed in Hexadecimal (ex: 3234303531333234303030313033343536343030)

#### 4. set\_GJB\_Status\_UHF(GJBMemoryBank, action, statusPassword, callback)

- Description: Set memory status with password for UHF GJB tag.
- Parameter:

Parameter	Type	Description
GJBMemoryBank	String	Select GJB memory bank for set,  E = EPC (default)  T = TID  U = User  2 = User 2  3 = User 3
action	String	Memory Status type  A: Lock0( readable and writable )  B: Lock1(read only)  C: Lock2(write only)  D: Lock3( non-readable and non-writable )  *Each Memory support status:  <ul style="list-style-type: none"> <li>• EPC area:  A: read and write    B: read only</li> </ul>

		<ul style="list-style-type: none"> <li>USER area(include User, User2, User3): A: read and write B: read only C: write only D: not allow access</li> <li>TID area: B: read only D: not allow access</li> <li>SAFE area: C: write only D: not allow access</li> </ul>
statusPassword	String	Status password , 8 HEX characters. (0~9, A,B,C,D,E,F)
callback	Function	Return when not connected"Connection doesn't exist."

## 5. Kill\_GJB\_Tag\_UHF(killPassword)

- Description: Kill UHF GJB tag.
- Parameter:

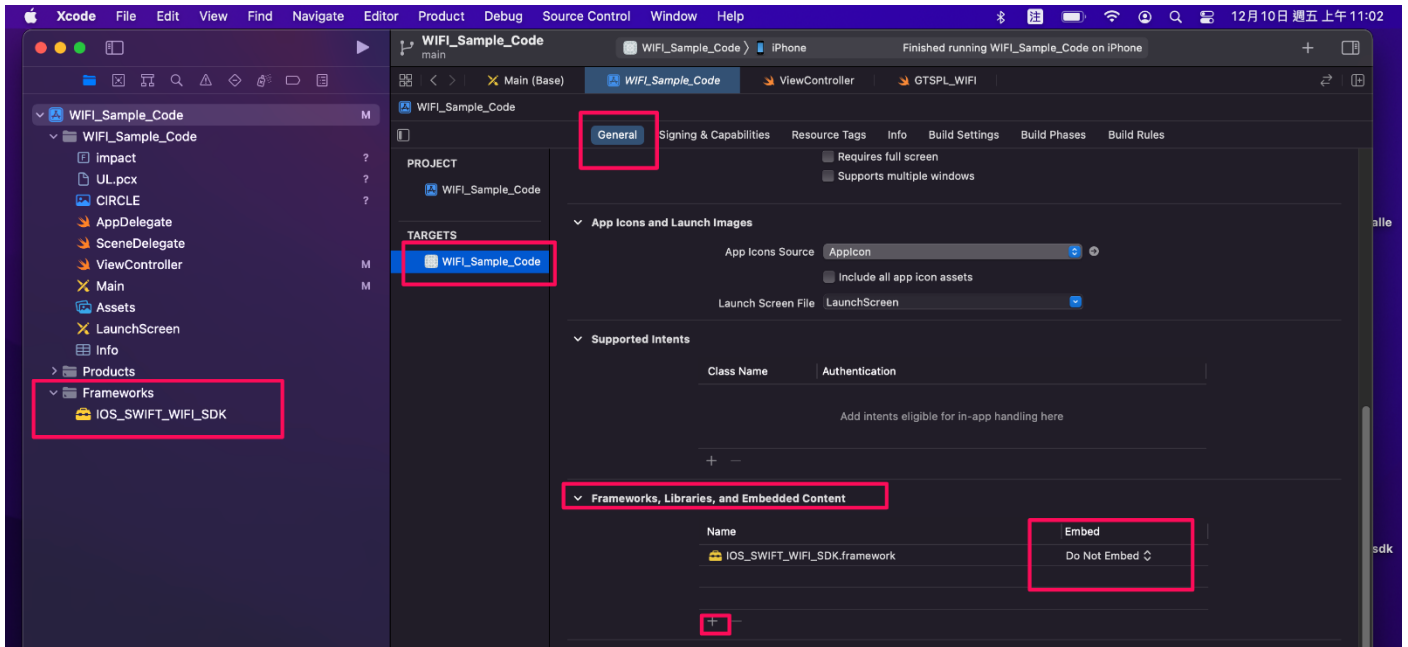
Parameter	Type	Description
killPassword	String	Killing password , 8 HEX characters. (0~9, A,B,C,D,E,F)



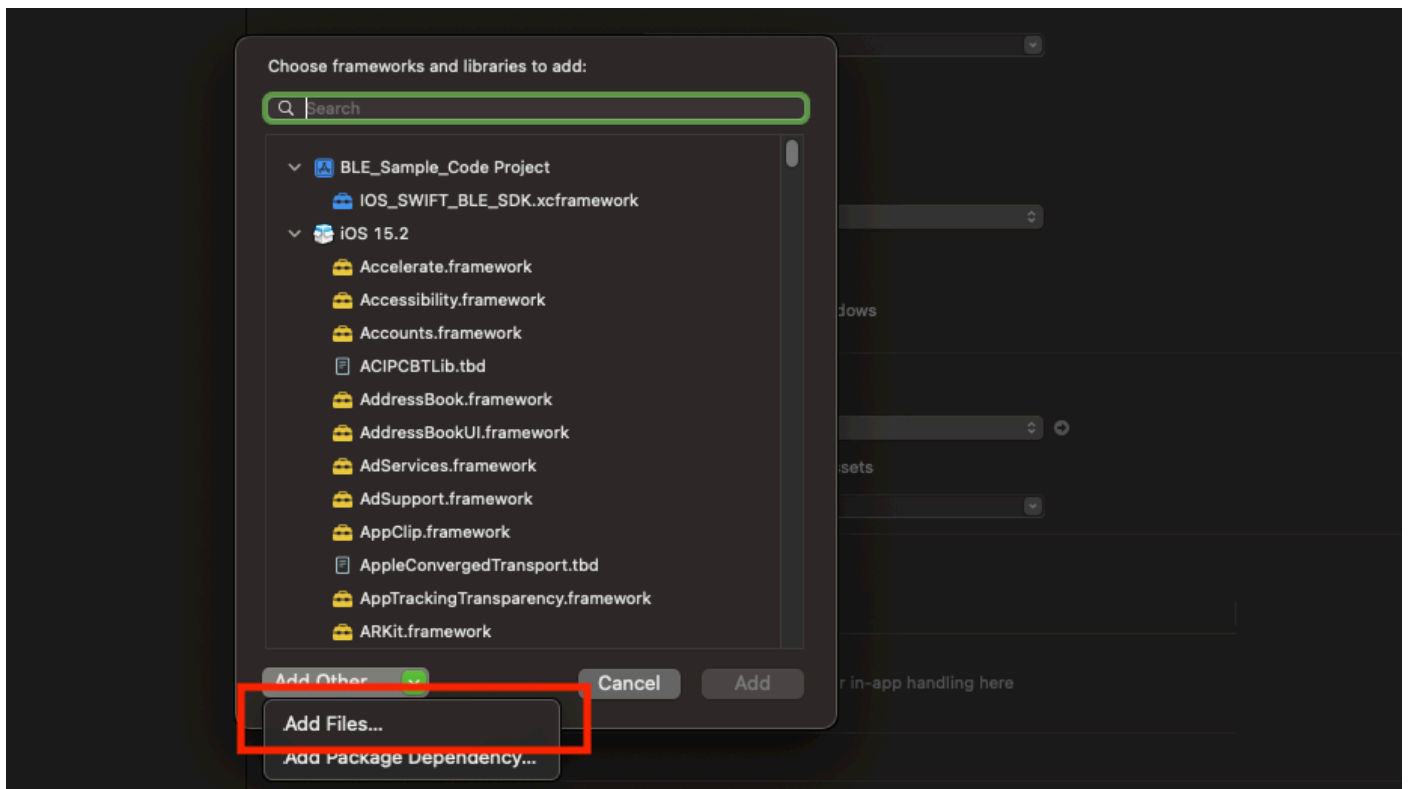
# Swift SDK Import Description

32

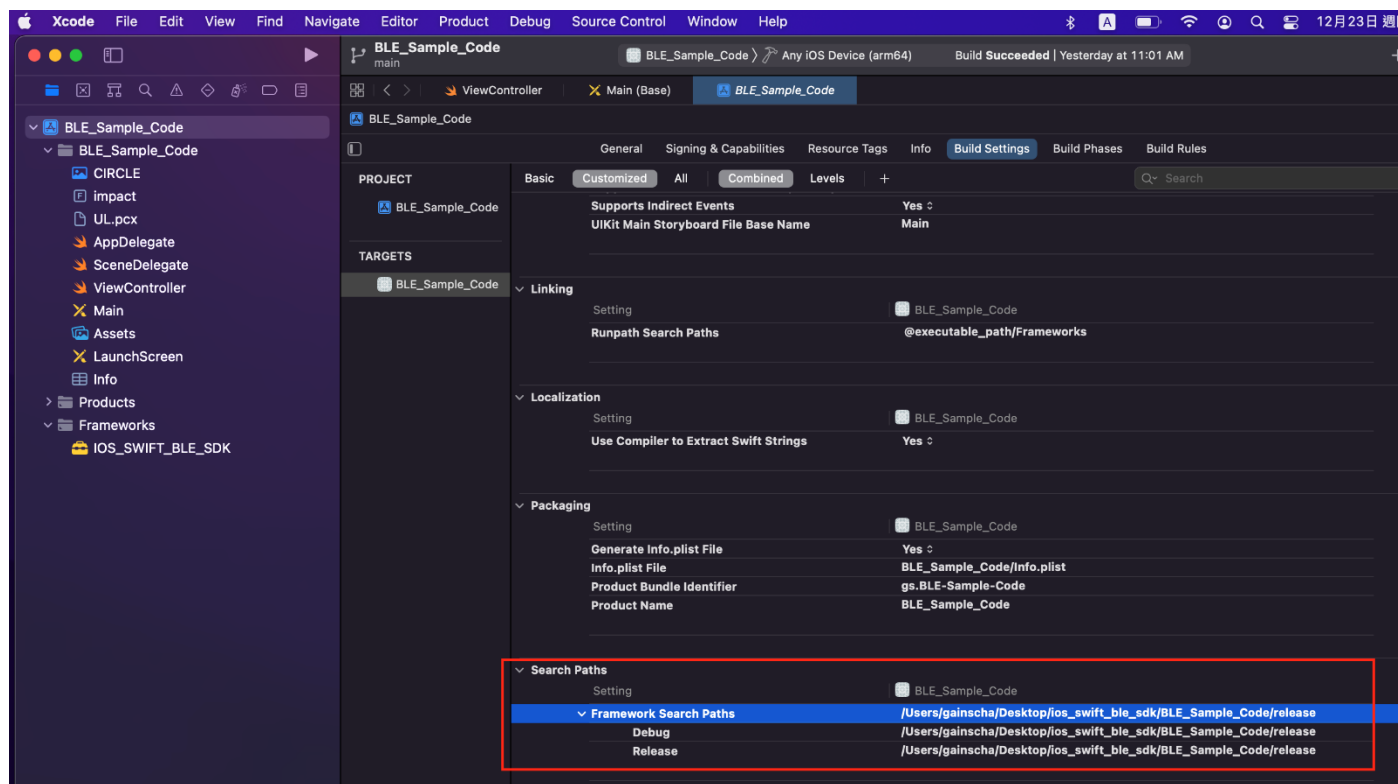
1.Import Framework: Target => General => Frameworks, Libraires, and Embedded Content add WIFI and BLE XCFramework, Note that Embed should choose Do Not Embed.



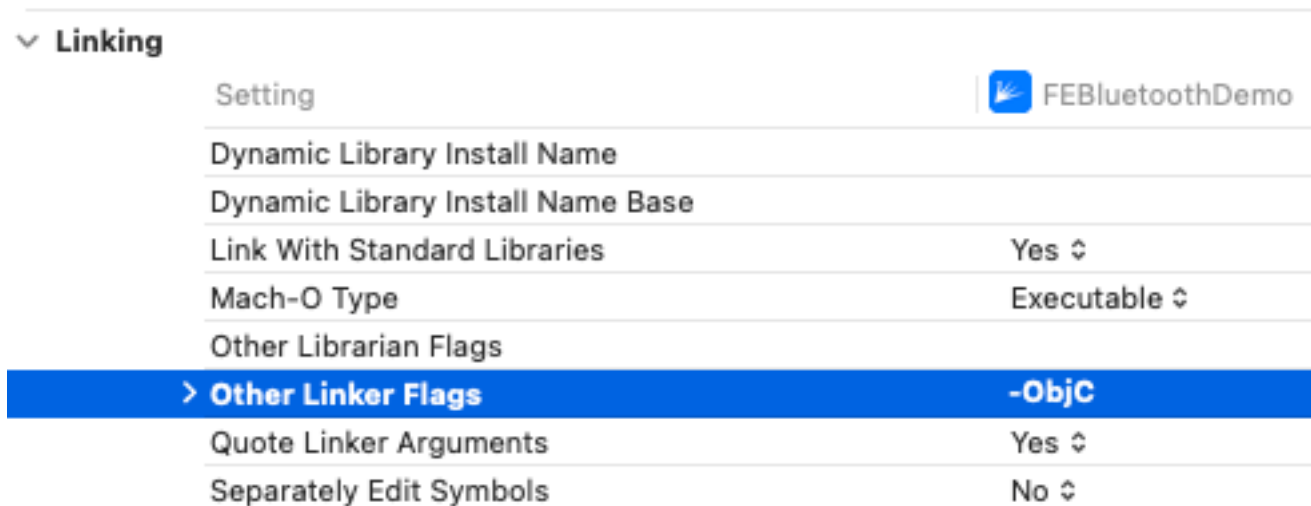
2.Choose Add files, and choose data in .xcframework, Press open to import



3. Setting Search Paths: Target => Building Settings => Combined => Search Paths, Set the path of Framework



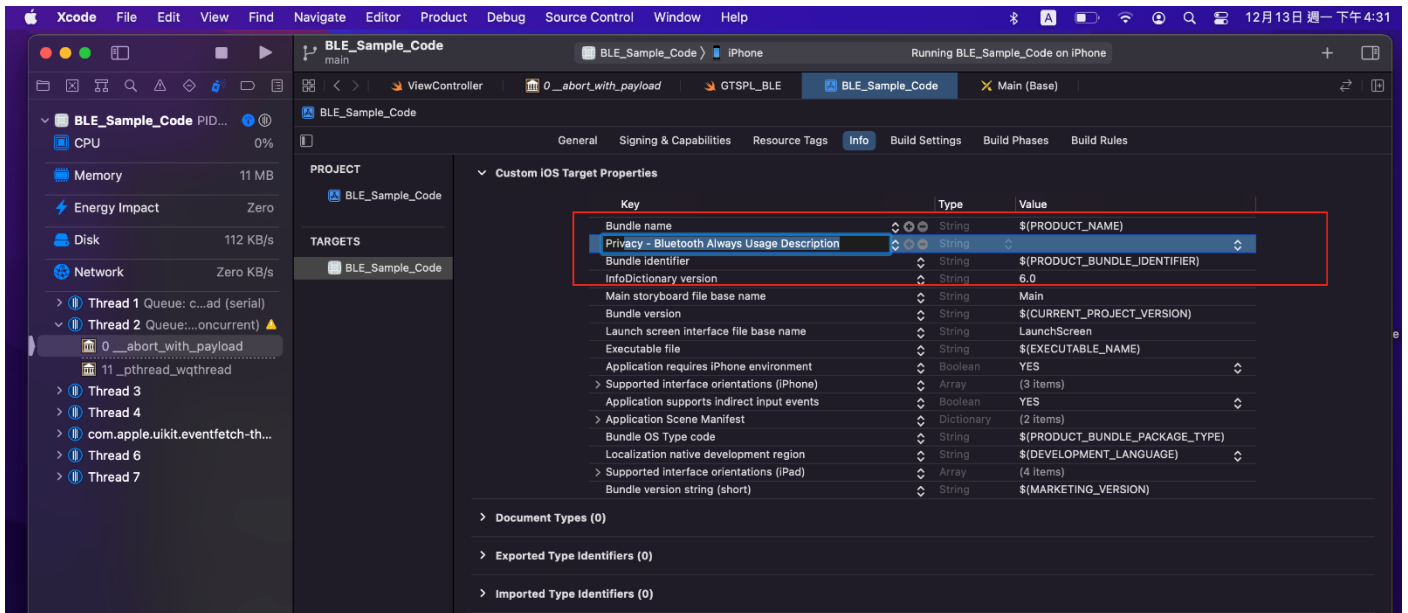
4. Set linker flags, TARGETS => Build Settings => Linking => Other Linker Flags, add "-ObjC".



# Swfit Bluetooth SDK Example Description

34

## 1.First Target=> Info set Properties: Privacy – Bluetooth Always Usage Description



## 2.Import GTSPB\_SDK:

```
import IOS_SWIFT_BLE_SDK
```

## 3.Example:

```
//Initialize SDK

let gsble = GTSPB_BLE()

// Array of bluetooth devices

var mFEPipherals = [String: FEPipheral]()

// Selected bluetooth device name

var mBleName: String?

// set BLE delegate to self

gsble.mDelegate = self
```

// inherit protocol GTSPL\_BLE\_Delegate to get BLE connection status

```
extension ViewController: GTSPL_BLE_Delegate {
    func sendBackStatus(_ object: IOS_SWIFT_BLE_SDK.GTSPL_BLE, status: String)
    {
        DispatchQueue.main.async {
            self.statusLabel.text = status // connected or disconnected
        }
    }
}
```

// Search BLE devices

```
gsble.fe_scan { devices in
    for device in devices {
        if let name = device.peripheral.name {
            self.mFEPipherals[name] = device
        }
    }
}
```

//BLE devices name

```
let btdevice = mFEPipherals [mBleName!]
```

//Connect BLE device

```
gsble.openPort(connectDevice: btdevice!){(msg) in
    self.textView.text = msg
}
```

// Disconnect BLE device

```
gsble.closePort(){(msg) in
    self.textView.text = msg
}
```

```
}
```

```
// reconnect the device
```

```
if let uuid = UUID(uuidString: uuidString) {
    gsble.retrieveDevice(uuid: uuid) { device in
        if let device = device {
            self.gsble.openPort(connectDevice: device) { state in
                var statusMessage = ""
                switch state {
                    case self.mTimeOutString:
                        statusMessage = "time out"
                    default:
                        statusMessage = state
                }
                self.statusLabel.text = statusMessage
            }
        }
    }
}
```

```
//Set printer
```

```
gsble.setup(width: 105, height: 80, speed: 4, density: 6, sensor: 0,
sensorDistance: 5, sensorOffset: 5) { msg in
    self.textView.text = msg
}
```

//Print barcode, text, bitmap, fontblock

// Clear data first, then print Specified items

gsble.clearBuffer()

//barcode

gsble.printBarcode(x: 30, y: 30, type: "128", height: 100, human\_readable: 1, rotation: 0, narrow: 2, wide: 2, content: "barcode987654321")

//text

gsble.printFont( x: 100, y: 180, fontname: "3", rotation: 0, xScale: 3, yScale: 3, content: "12345678 print test")

//fontblock

let paragraph="We stand behind our products with one of the most comprehensive support programs in the Auto-ID industry."

gsble.printFontBlock(x: 15,y: 15,width: 790,height: 90,fontname: "1",rotation: 0,xScale: 8,yScale: 8,space: 20,align: 0,content: paragraph)

// Specifies the file of the bitmap url

let url = Bundle.main.url(forResource: "CIRCLE", withExtension: "bmp")!

// Send the bitmap file to the printer

gsble.downloadBMP(filepath: url, filename: "CIRCLE.bmp")

//Command of print bitmap

gsble.sendcommand("PUTBMP 30,30,\"CIRCLE.bmp\"\r\n")

// Execute print action

gsble.printLabel(set: 1, copy: 1)

//Print QRcode

```
// Clear data first
```

```
gsble.clearBuffer()
```

```
//formed a QRcode
```

```
gsble.printQRCode(x: 50, y: 50, eccLevel: "H", cellWidth: 4, mode: "A",  
rotation: 0, content: "QRcode987654321")
```

```
// Print action
```

```
gsble.printLabel(set: 1, copy: 1)
```

```
// Print Simplified Chinese
```

```
cn: String = "默认简体中文测试"
```

```
gsble.clearBuffer()
```

```
gsble.printFont(x: 200, y: 200, fontName: "TSS24.BF2", rotation: 0, x_scale:  
1, y_scale: 1, content: cn)
```

```
gsble.printLabel(set: 1, copy: 1)
```

```
// Print traditional Chinese
```

```
cn: String = "預設繁體中文測試"
```

```
gsble.clearBuffer()
```

```
gsble.printFont(x: 200, y: 200, fontName: "TST24.BF2", rotation: 0, x_scale:  
1, y_scale: 1, content: cn)
```

```
gsble.printLabel(set: 1, copy: 1)
```

```
//Get SDK version
```

```
let version = gsble.getSdkVersion()
```

//Get printer status

```
var statusStr = ""

gsble.printerStatus() {(status) in

switch status{

    case "00":

        statusStr = "Normal"

    case "01":

        statusStr = "Head opened"

    case "02":

        statusStr = "Paper Jam"

    case "03":

        statusStr = "Paper Jam and head opened"

    case "04":

        statusStr = "Out of paper"

    case "05":

        statusStr = "Out of paper and head opened"

    case "08":

        statusStr = "Out of ribbon"

    case "09":

        statusStr = "Out of ribbon and head opened"

    case "0A":

        statusStr = "Out of ribbon and paper jam"

    case "0B":
```



```

        statusStr = "Out of ribbon, paper jam and head opened"

    case "0C":

        statusStr = "Out of ribbon and out of paper"

    case "0D":

        statusStr = "Out of ribbon, out of paper and head opened"

    case "10":

        statusStr = "Pause"

    case "20":

        statusStr = "Printing"

    case "80":

        statusStr = "Other error"

    default:

        break

    }

}

```

// Set direction and mirror.

```
gsble.setDirectionAndMirror(direction: 0, mirror: 0)
```

// Set the vertical displacement distance, example: 36 dots

```
gsble.setShift(shiftY: 36)
```

// Reverse the designated area.

```
gsble.printReverse(x_start: 0, y_start: 0, x_width: 100, y_height: 100)
```

```
// Set feed offset, example: 9.9mm
```

```
gsble.setOffset(offset: 9.9)
```

```
// Set cut mode and cut number, example: cut forward + 3 pieces
```

```
gsble.setCutMode(mode: 1, piece: 3)
```

```
// Set action after print, example: Tear Mode
```

```
gsble.setAfterPrintAction(mode: 1)
```

```
// Initialize the general setting value
```

```
gsble.genericDefault()
```

```
// Initialize the sensor setting value
```

```
gsble.sensorDefault()
```

```
// switch wifi frequency 2.4G/5G/Both , example: 2.4G
```

```
gsble.switchWifiFrequency(frequency: "2.4G") { msg in
```

```
    print("Error message: \(msg)")
```

```
}
```

```
// convert image to black and white bitmap file and printing.
```

// example: mode 1, modifies width and height to 200 dots and 100 dots

```
gsble.printBitmap(imageData: Data, x: 0, y: 0, width: 200, height: 100, mode: 1) { msg in
    print("Error message: \(msg)")
}
```

// check WIFI IP address

```
gsble.fe_checkIP { message, IPAddress, error in
    print(">>> " + message)
    if error != nil {
        print(">>> error: \(error!.localizedDescription)")
        return
    }
    DispatchQueue.main.async {
        self.showDialog(title: "IP Address", msg: IPAddress ?? "unknown")
    }
}
```

// Configure dynamic WIFI / Network

```
gsble.fe_btConfigNetwork(isDynamic: true, ssid: "Sample", password: "12345678",
staticIp: "", gw: nil, mask: nil, dns: nil, reconnect: true) { message, IPAddress, error in
    print(">>> \(message)")
    guard error == nil else {
        DispatchQueue.main.async {
            self.showDialog(title: "Error", msg: error!.localizedDescription)
        }
        return
    }
    if let IPAddress = IPAddress {
        DispatchQueue.main.async {
```

```

        self.showDialog(title: "Message", msg: "Succesed!, ip: \(IPAddress)")
    }
} else {
    DispatchQueue.main.async {
        self.showDialog(title: "Alert", msg: "Config Error!")
    }
}
}
}

```

#### // Configure static WIFI / Network

```

gsble.fe_btConfigNetwork(isDynamic: false, ssid: "Sample", password: "12345678",
staticIp: "192.168.66.118", gw: "192.168.66.1", mask: "255.255.255.0", dns: nil, reconnect:
true) { message, IPAddress, error in
    print(">>> \(message)")
    guard error == nil else {
        DispatchQueue.main.async {
            self.showDialog(title: "Error", msg: error!.localizedDescription)
        }
        return
    }
    if let IPAddress = IPAddress {
        DispatchQueue.main.async {
            self.showDialog(title: "Message", msg: "Succesed!, ip: \(IPAddress)")
        }
    } else {
        DispatchQueue.main.async {
            self.showDialog(title: "Alert", msg: "Config Error!")
        }
    }
}
}

```

#### // RFID Setting , example: read position 10 dots

```

gsble.setRFIDProcedure(tagType: 8, rw_position: 10, void_printout: 10, tryEncode_times: 3,
error_handle: "E", speed: 2, retry_times: 6) { msg in

    print("Error message: \(msg)")

}

```

// RFID Setting , example: read position 10 mm

```
gsble.setRFIDProcedure(tagType: 8, rw_position: 10, void_printout: 10, tryEncode_times: 3,
error_handle: "E", speed: 2, retry_times: 6, dpi: "203") { msg in

    print("Error code: \(msg)")

}
```

// Auto calibration for RFID label

```
gsble.rfid_calibration() { msg in

    print("Error message" \(msg))

}
```

//UHFReaderE710 auto calibration

```
gsble.rfid_labelCalibration(width: 62.0, height: 45.0, sensor: 0, sensorDistance: 3.0, tagType: 8)
```

// Initialize the RFID setting value

```
gsble.rfidSetupDefault()
```

//UHF GEN2 EPC lock data use L, unlock use U

```
gsble.EPCPWD_Action(action: "L", password: "12345678")
```

//UHF GEN2 TID lock data use L, unlock use U

```
gsble.TIDPWD_Action(action: "L", password: "12345678")
```

//UHF GEN2 USER lock data use L, unlock use U

```

gsble.USERPWD_Action(action: "L", password: "12345678")

//UHF GEN2 access password lock data use L, unlock use U

gsble.accessPWD_Action(action: "L", password: "12345678")

//UHF GEN2 kill password lock data use L, unlock use U

gsble.killPWD_Action(action: "L", password: "12345678")


//UHF GEN2 set access password

gsble.accessPWD_Action(action: "S", password: "12345678")

//UHF GEN2 set kill password

gsble.killPWD_Action(action: "S", password: "12345678")

//UHF GEN2 write data

gsble.clearBuffer()

gsble.writeUHF(dataFormat: "H", startBlockNo: 2, byteSize: 12,
Gen2MemoryBank: "E", dataString: "414142424343444445454646") { msg in

    self.textView.text = msg // Indicates that the printer isn't connected

}

gsble.printLabel(set: 1, copy: 1)

// UHF GEN2 read data

gsble.readUHF(dataFormat: "H", startBlockNo: 0, byteSize: 12,
Gen2MemoryBank: "E") { msg in

    if msg == "Connection doesn't exist." {} // Indicates that the printer isn't connected

    else if msg == "time_out" {} //time out

    else {}//Other return, Indicates read data

```

```
// UHF GEN2 read data(Q Command)
```

```
gsble.query_UHF(dataFormat: "H", PCReturnStatus: 1, CRCReturnStatus: 1) { msg
```

in

```
    if msg == "Connection doesn't exist." {
```

```
        self.textView.text = msg // Indicates that the printer isn't connected
```

```
        return
```

```
    } else if msg == "time_out" {
```

```
        self.textView.text = "time out" //time out
```

```
        return
```

```
    } else {
```

```
        Self.textView.text = msg // Other return, Indicates read data
```

```
    }
```

```
}
```

```
// UHF GJB set password
```

```
// Use write password, set new read password
```

```
gsble.set_GJB_Pwd_Action(passwordArea: "R", newPassword: "87654321",  
writePassword: "12345678")
```

```
// Use write password, set new write password
```

```
gsble.set_GJB_Pwd_Action(passwordArea: "W", newPassword: "87654321",  
writePassword: "12345678")
```

```
// Use write password, set new kill password
```

```
gsble.set_GJB_Pwd_Action(passwordArea: "K", newPassword: "87654321",
```

```
writePassword: "12345678")
```

```
// Use write password, set new status password
```

```
gsble.set_GJB_Pwd_Action(passwordArea: "S", newPassword: "87654321",
writePassword: "12345678")
```

```
// UHF GJB use status password, set status
```

```
gsble.statusGJB_UHF(GJBMemoryBank: "E", action: "C", statusPassword:
"11112222")
```

```
// UHF GJB use write password, write data
```

```
gsble.clearBuffer()
```

```
gsble.writeGJB_UHF(dataFormat: "H", startBlockNo: 1, byteSize: 12,
Gen2MemoryBank: "E", dataString: "404041414242434344444545", writePassword:
"12345678") { msg in
```

```
    self.textView.text = msg // Indicates that the printer isn't connected
```

```
}
```

```
gsble.printLabel(set: 1, copy: 1)
```

```
// UHF GJB use read password, read data
```

```
gsble.readGJB_UHF(dataFormat: "H", startBlockNo: 0, byteSize: 12,
Gen2MemoryBank: "E", readPassword: "33334444") { msg in
```

```
    if msg == "Connection doesn't exist." {} // Indicates that the printer isn't connected
```

```
    else if msg == "time_out" {} // time out
```

```
    else {} // Other return, Indicates read data
```

```
// UHF GJB use kill password, delete tag
```

```
gsble.killGJB_Tag_UHF(killPassword: "11224455")
```

```
// Open real-time
```

```
gsble.setRealTimeStatus("1")
```



```
isRealTimeOn = true // Set true in real-time
```

```
timer = Timer.scheduledTimer(withTimeInterval: 0.5, repeats: true, block: { _ in // Start
counting, read every 0.5 seconds
```

```
    if let statusNumber = self.gsble.getStatusNumber() {
```

```
        var statusStr = ""
```

```
        switch statusNumber {
```

```
            case 0:    statusStr = "Normal"
```

```
            // omitted..... get status code, Check the table to get the current status.
```

```
            self.realTimeShowLabel.text = statusStr // Displayed on the UILabel screen
```

```
// Close real-time
```

```
gsble.setRealTimeStatus("0")
```

```
isRealTimeOn = false // Set false in real-time
```

```
timer.invalidate() // Close timer
```

```
// Read error codes list
```

```
let errorCode = gsble.getRFIDErrorCode()
```

# Swfit WIFI SDK Example Description

1.Import GTSPL\_SDK:

```
import IOS_SWIFT_WIFI_SDK
```

2.Example:

```
// Initialize SDK
```

```
let gswifi = GTSPL_WIFI()
```

```
//Connect WIFI device
```

```
let ip = IPText.text
```

```
let port: Int? = Int(PortText.text!)
```

```
gswifi.openport(IP: ip!, Port: port!){(msg) in
```

```
    print(msg)
```

```
}
```

```
// Disconnect WIFI device
```

```
gswifi.closePort()
```

```
//Set printer
```

```
gswifi.setup(width: 105, height: 80, speed: 4, density: 6, sensor: 0,  
sensorDistance: 3, sensorOffset: 3)
```

```
//Print barcode, text, bitmap
```

```
// Clear data first, then print Specified items
```

```
gswifi.clearBuffer()
```

```
//barcode
```

```
gswifi.printBarcode(x: 30, y: 30, type: "128", height: 100, human_readable:  
1, rotation: 0, narrow: 2, wide: 2, content: "barcode987654321")
```

```
//text
```

```
gswifi.printFont( x: 100, y: 180, fontname: "3", rotation: 0, xScale: 3, yScale: 3, content: "12345678 print test")
```

```
//fontblock
```

```
let paragraph="We stand behind our products with one of the most comprehensive support programs in the Auto-ID industry."
```

```
gswifi.printFontBlock(x: 15,y: 15,width: 790,height: 90,fontname: "1",rotation: 0,xScale: 8,yScale: 8,space: 20,align: 0,content: paragraph)
```

```
// Specifies the file of the bitmap url
```

```
let url = Bundle.main.url(forResource: "CIRCLE", withExtension: "bmp")!
```

```
// Send the bitmap file to the printer
```

```
gswifi.downloadBMP(filePath: url, fileName: "CIRCLE.bmp")
```

```
// Command of print bitmap
```

```
gswifi.sendCommand("PUTBMP 30,30,\"CIRCLE.bmp\"\\r\\n")
```

```
// Execute print action
```

```
gswifi.printLabel(set: 1, copy: 1)
```

```
//Print QRcode
```

```
// Clear data first
```

```
gswifi.clearBuffer()
```

```
// formed a QRCode
```

```
gswifi.printQRCode(x: 50, y: 50, eccLevel: "H", cellWidth: 4, rotation: 0,
content: "QRcode987654321")
```

```
gswifi.printLabel(set: 1, copy: 1)
```

```
// Print simplified Chinese
```

```
String stString="默认简体中文测试";
```

```
gswifi.clearBuffer()
```

```
gswifi.printFont(x: 200, y: 200, fontName: "TSS24.BF2", rotation: 0, x_scale:
1, y_scale: 1, content: cn)
```

```
gswifi.printLabel(set: 1, copy: 1)
```

```
// Print traditional Chinese
```

```
cn: String = "預設繁體中文測試"
```

```
gswifi.clearBuffer()
```

```
gswifi.printFont(x: 200, y: 200, fontName: "TST24.BF2", rotation: 0, x_scale:
1, y_scale: 1, content: cn)
```

```
gswifi.printLabel(set: 1, copy: 1)
```

```
//Get sdk version
```

```
let version = gswifi.getSdkVersion()
```

```
//Get printer status
```

```
let status = gswifi.printerStatus()
```

```
var statusStr = ""
```

```
switch status{
```

```
    case "00":
```

```
        statusStr = "Normal"
```

```
case "01":  
    statusStr = "Head opened"  
  
case "02":  
    statusStr = "Paper Jam"  
  
case "03":  
    statusStr = "Paper Jam and head opened"  
  
case "04":  
    statusStr = "Out of paper"  
  
case "05":  
    statusStr = "Out of paper and head opened"  
  
case "08":  
    statusStr = "Out of ribbon"  
  
case "09":  
    statusStr = "Out of ribbon and head opened"  
  
case "0A":  
    statusStr = "Out of ribbon and paper jam"  
  
case "0B":  
    statusStr = "Out of ribbon, paper jam and head opened"  
  
case "0C":  
    statusStr = "Out of ribbon and out of paper"  
  
case "0D":  
    statusStr = "Out of ribbon, out of paper and head opened"  
  
case "10":
```

```
        statusStr = "Pause"

    case "20":

        statusStr = "Printing"

    case "80":

        statusStr = "Other error"

    default:

        break

}
```

// Set direction and mirror.

```
gswifi.setDirectionAndMirror(direction: 0, mirror: 0)
```

// Set the vertical displacement distance, example: 36 dots

```
gswifi.setShift(shiftY: 36)
```

// Reverse the designated area.

```
gswifi.printReverse(x_start: 0, y_start: 0, x_width: 100, y_height: 100)
```

// Set feed offset, example: 9.9mm

```
gswifi.setOffset(offset: 9.9)
```

// Set cut mode and cut number, example: cut forward + 3 pieces

```
gswifi.setCutMode(mode: 1, piece: 3)
```

// Set action after prin, example: Tear Mode

```
gswifi.setAfterPrintAction(mode: 1)
```

// Initialize the general setting value

```
gswifi.genericDefault()
```

// Initialize the sensor setting value

```
gswifi.sensorDefault()
```

// switch wifi frequency 2.4G/5G/Both , example: 2.4G

```
gswifi.switchWifiFrequency(frequency: "2.4G") { msg in
```

```
    print("Error message: \(msg)")
```

```
}
```

// convert image to black and white bitmap file and printing.

// example: mode 1, modifies width and height to 200 dots and 100 dots

```
gswifi.printBitmap(imageData: Data, x: 0, y: 0, width: 200, height: 100, mode: 1) { msg in
```

```
    print("Error message: \(msg)")
```

```
}
```

// RFID Setting , example: read position 10 dots

```
gswifi.setRFIDProcedure(tagType: 8, rw_position: 10, void_printout: 10, tryEncode_times: 3,
error_handle: "E", speed: 2, retry_times: 6) { msg in
```

```
    print("Error message: \(msg)")
```

```
}
```

```
// RFID Setting , example: read position 10 mm
```

```
gswifi.setRFIDProcedure(tagType: 8, rw_position: 10, void_printout: 10, tryEncode_times: 3,
error_handle: "E", speed: 2, retry_times: 6, dpi: "203") { msg in
```

```
    print("Error code: \(msg)")
```

```
}
```

```
// Auto calibration for RFID label
```

```
gswifi.rfid_calibration() { msg in
```

```
    print("Error message" \(msg) )
```

```
}
```

```
//UHFReaderE710 auto calibration
```

```
gswifi.rfid_labelCalibration(width: 62.0, height: 45.0, sensor: 0, sensorDistance: 3.0, tagType: 8)
```

```
// Initialize the RFID setting value
```

```
gswifi.rfidSetupDefault()
```



//UHF GEN2 EPC lock data use L, unlock use U

```
gswifi.EPCPWD_Action(action: "L", password: "12345678")
```

//UHF GEN2 TID lock data use L, unlock use U

```
gswifi.TIDPWD_Action(action: "L", password: "12345678")
```

//UHF GEN2 USER lock data use L, unlock use U

```
gswifi.USERPWD_Action(action: "L", password: "12345678")
```

//UHF GEN2 access password lock data use L, unlock use U

```
gswifi.accessPWD_Action(action: "L", password: "12345678")
```

//UHF GEN2 kill password lock data use L, unlock use U

```
gswifi.killPWD_Action(action: "L", password: "12345678")
```

//UHF GEN2 set access password

```
gswifi.accessPWD_Action(action: "S", password: "12345678")
```

//UHF GEN2 set kill password

```
gswifi.killPWD_Action(action: "S", password: "12345678")
```

//UHF GEN2 write data

```
gswifi.clearBuffer()
```

```
gswifi.writeUHF(dataFormat: "H", startBlockNo: 2, byteSize: 12,
Gen2MemoryBank: "E", dataString: "414142424343444445454646") { msg in
```

```
    self.textView.text = msg // Indicates that the printer isn't connected
```

```
}
```

```
gswifi.printLabel(set: 1, copy: 1)
```

// UHF GEN2 read data

```

    gswifi.readUHF(dataFormat: "H", startBlockNo: 0, byteSize: 12,
Gen2MemoryBank: "E") { msg in

    if msg == "Connection doesn't exist." {} // Indicates that the printer isn't connected

    if msg == "time_out" {} // time out

    else {} // Other return, Indicates read data

// UHF GJB set password

    // Use write password, set new read password

    gswifi.set_GJB_Pwd_Action(passwordArea: "R", newPassword: "87654321",
writePassword: "12345678")

    // Use write password, set new write password

    gswifi.set_GJB_Pwd_Action(passwordArea: "W", newPassword: "87654321",
writePassword: "12345678")

    // Use write password, set new kill password

    gswifi.set_GJB_Pwd_Action(passwordArea: "K", newPassword: "87654321",
writePassword: "12345678")

    // Use write password, set new status password

    gswifi.set_GJB_Pwd_Action(passwordArea: "S", newPassword: "87654321",
writePassword: "12345678")

// UHF GJB use status password, set status

    gswifi.statusGJB_UHF(GJBMemoryBank: "E", action: "C", statusPassword:
"11112222")

// UHF GJB use write password, write data

    gswifi.clearBuffer()

    gswifi.writeGJB_UHF(dataFormat: "H", startBlockNo: 1, byteSize: 12,
Gen2MemoryBank: "E", dataString: "404041414242434344444545", writePassword:
"12345678") { msg in

```

```

        self.textView.text = msg {} // Indicates that the printer isn't connected
    }

    gswifi.printLabel(set: 1, copy: 1)

    // UHF GJB use read password, read data

    gswifi.readGJB_UHF(dataFormat: "H", startBlockNo: 0, byteSize: 12,
    Gen2MemoryBank: "E", readPassword: "33334444") { msg in

        if msg == "Connection doesn't exist." {} // Indicates that the printer isn't connected

        if msg == "time_out" {} // time out

        else {} // Other return, Indicates read data

    // UHF GJB use kill password, delete tag

    gswifi.killGJB_Tag_UHF(killPassword: "11224455")

    // Open real-time

    gswifi.setRealTimeStatus("1")

    isRealTimeOn = true // Set true in real-time

    timer = Timer.scheduledTimer(withTimeInterval: 0.5, repeats: true, block: { _ in // Start
    counting, read every 0.5 seconds

        if let statusNumber = self.gsble.getStatusNumber() {

            var statusStr = ""

            switch statusNumber {

            case 0: statusStr = "Normal"

            // omitted..... get status code, Check the table to get the current status.

            self.realTimeShowLabel.text = statusStr // Displayed on the UILabel screen

        }

    // Close real-time

```

```
gswifi.setRealTimeStatus("0")  
  
isRealTimeOn = false // Set false in real-time  
  
timer.invalidate() // Close timer  
  
// Read error codes list  
  
let errorCode = gswifi.getRFIDErrorCode()
```

Code Type	Description	Narrow : Width					Max. data length
		1: 1	1: 2	1: 3	2: 5	3: 7	
128	Code 128, switching code subset automatically.	V					
128M	Code 128, switching code subset manually.	V					
EAN128	EAN128, switching code subset automatically.	V					
EAN128 M	EAN128M, switching code subset manually.	V					
25	Interleaved 2 of 5.		V	V	V		Length is even
25C	Interleaved 2 of 5 with check digit.		V	V	V		Length is odd
25S	Standard 2 of 5.		V	V	V		
25I	Industrial 2 of 5.		V	V	V		
39	Code 39, switching standard and full ASCII mode automatically.		V	V	V		
39C	Code 39 with check digit.		V	V	V		
93	Code 93.			V			
EAN13	EAN 13.	V					12
EAN13+2	EAN 13 with 2 digits add-on.	V					14
EAN13+5	EAN 13 with 5 digits add-on.	V					17
EANB	EAN 8.	V					7
EANB+2	EAN 8 with 2 digits add-on.	V					96
EANB+5	EAN 8 with 5 digits add-on.	V					12
CODA	Codabar.		V	V	V		
POST	Postnet.	V					5,9,11
UPCA	UPC-A.	V					11
UPCA+2	UPC-A with 2 digits add-on.	V					13
UPA+5	UPC-A with 5 digits add-on.	V					16
UPCE	UPC-E.	V					6
UPCE+2	UPC-E with 2 digits add-on.	V					8
UPE+5	UPC-E with 5 digits add-on.	V					11
MSI	MSI.		V	V	V		
MSIC	MSI with check digit.		V	V	V		
PLESSE Y	PLESSEY.		V	V	V		

CPOST	China post.					V	
ITF14	ITF14.		V	V	V		13
EAN14	EAN14.	V					13
11	Code 11.		V	V	V		
TELEPE N	Telepen. *Since V6.89EZ.		V	V	V		
TELEPE NN	Telepen number. *Since V6.89EZ.		V	V	V		
PLANET	Planet. *Since V6.89EZ.	V					
CODE49	Code 49. *Since V6.89EZ.	V					
DPI	Deutsche Post Identcode. *Since V6.91EZ.		V	V	V		11
DPL	Deutsche Post Leitcode. *Since V6.91EZ.		V	V	V		13
LOGMAR S	A special use of Code 39. *Since V6.88EZ.		V	V	V		

## Appendix Two: RFID Error Code

Error Code	Error Description
00	No electronic tag found.
05	Wrong access password.
FA	There is an electronic tag, but the communication is poor and the operation failed.
FB	No electronic tag is operable.
FC	Electronic tag returned error.
FD	The command length is incorrect.
FE	Illegal command.
FF	Parameter error.