

Windows SDK

说明手册 (V1.25)



GTSP.L.DLL 函式库支持开发平台/语言

开发平台/语言	范例代码说明
C Sharp	第 26~36 页
Java	第 37~50 页
Javascript	第 50~52 页
ASP.NET	第 52~54 页
VB.NET	第 54~56 页
Access-VBA	第 56~57 页
Excel-VBA	第 58 页
Visual C++	第 59~69 页
FoxPro	第 70~71 页
Python	第 72~76 页
VB6	第 77~82 页
PHP	第 83~89 页
Qt C++	第 90~143 页
JSP	第 144~150 页

GTSP.L.DLL 函式库使用说明(V1.25)

1. openport(name)、openport_USB()、openports_USB(PrinterModel)、openport_Ethernet(IP, port)

■ 函式说明：指定并开启计算机端的输出端口

■ 参数说明：

➔ name：字符串型别

(1) 单机打印时，请指定打印机驱动程序名称

(2) 若连接网络打印机，请指定打印机路径及共享打印机名称

如：\\192.168.1.104\ Printer Model

➔ PrinterModel：字符串型别，打印机机型名称

➔ IP：字符串型别，打印机 IP 位址

➔ port：int 型别，打印机连接埠号码

2. closeport()、closeport_USB()、closeport_Ethernet()

■ 函式说明：关闭输出埠

■ 参数说明：无

3. detectUSB_USB()

■ 函式说明：侦测打印机 USB 接口插拔状况

■ 参数说明：无

■ 回传说明：字符串数组型别，以 USB 连接所有开启的打印机

4. detectUSBStr_USB()

■ 函式说明：侦测打印机 USB 接口插拔状况

■ 参数说明：无

■ 回传说明：字符串型别，以 USB 连接所有开启的打印机

5. sendcommand(command)、sendcommand_USB(command)、sendcommand_Ethernet(command)

■ 函式说明：将内建命令传送至打印机(结尾会自动加入\r\n 命令结束用语)

■ 参数说明：

➔ command：字符串型别，设定指令内容，详细指令请参考 TSPL 使用手册

6. `sendcommand_Array(command)`、`sendcommand_Array_USB(command)`、

`sendcommand_Array_Ethernet(command)`

■ 函式说明：将内建命令传送至打印机(结尾会自动加入\r\n 命令结束用语)

■ 参数说明：

➔ `command`：字节数组型别，设定指令内容，详细指令请参考 TSPL 使用手册

7. `sendcommand_noCIRf(command)`、`sendcommand_noCIRf_USB(command)`、

`sendcommand_noCIRf_Ethernet(command)`

■ 函式说明：将内建命令传送至打印机(结尾不会自动加入\r\n 命令结束用语)

■ 参数说明：

➔ `command`：字符串型别，设定指令内容，详细指令请参考 TSPL 使用手册

8. `clearbuffer()`、`clearbuffer_USB()`、`clearbuffer_Ethernet()`

■ 函式说明：清除图像缓冲

■ 参数说明：无

9. `setup(width, height, speed, density, sensor, sensorDistance, sensorOffset)`、

`setup_USB(width, height, speed, density, sensor, sensorDistance, sensorOffset)`、

`setup_Ethernet(width, height, speed, density, sensor, sensorDistance, sensorOffset)`

■ 函式说明：设定卷标的宽度、高度、打印速度、打印热度、传感器类别、间隙/黑标垂直间距、间隙/黑标偏移距离

■ 参数说明：

参数	型别	说明
<code>width</code>	字符串	设定标签宽度，单位 mm
<code>height</code>	字符串	设定标签高度，单位 mm
<code>speed</code>	字符串	设定打印速度，1~15，代表每秒 1~15 吋打印速度(随机型不同会有不同打印最高上限，最高为每秒 15 吋打印速度)
<code>density</code>	字符串	设定打印浓度，0~15，数字越大打印结果越黑

sensor	字符串	设定使用传感器之类别： 0: 表示使用间隙传感器(gap sensor) 1: 表示使用黑标传感器(black mark sensor)
sensorDistance	字符串	设定间隙/黑标垂直间距高度，单位 mm
sensorOffset	字符串	设定间隙/黑标垂直间距高度，单位 mm，此参数若使用一般标签时均设为 0

10. barcode(x, y, type, height, readable, rotation, narrow, wide, content)、

barcode_USB(x, y, type, height, readable, rotation, narrow, wide, content)、

barcode_Ethernet(x, y, type, height, readable, rotation, narrow, wide, content)

■ 函式说明：使用打印机内建条形码打印

■ 参数说明：

参数	型别	说明
x	字符串	条形码 X 方向起始点，以点(dot)表示
y	字符串	条形码 Y 方向起始点，以点(dot)表示
type	字符串	设定条形码类型(Code Type) ， 请参考附件
height	字符串	设定条形码高度，高度以点来表示
readable	字符串	设定是否打印条形码码文 0:不打印 1:打印条形码码文置左 2:打印条形码码文置中 3:打印条形码码文置右
rotation	字符串	设定条形码旋转角度 0: 旋转0度 90: 旋转90度 180: 旋转180度 270: 旋转270度
narrow	字符串	设定条形码窄 bar 比例因子， 请参考附件
wide	字符串	设定条形码宽 bar 比例因子， 请参考附件
content	字符串	设定欲打印之条形码内容

11. `qrcode(x, y, ECCLevel, cellWidth, mode, rotation, content)`、
`qrcode_USB(x, y, ECCLevel, cellWidth, mode, rotation, content)`、
`qrcode_Ethernet(x, y, ECCLevel, cellWidth, mode, rotation, content)`

■ 函式说明：使用打印机打印 QRcode

■ 参数说明：

参数	型别	说明
x	字符串	QRCode X 方向起始点，以点(dot)表示
y	字符串	QRCode Y 方向起始点，以点(dot)表示
ECCLevel	字符串	容错率 L : 7% M : 15% Q : 25% H : 30%
cellWidth	字符串	设定 QRCode 大小，1~10
mode	字符串	设定自动或手动编码 A : 自动 M : 手动
rotation	字符串	设定QRCode旋转角度 0 : 旋转0度 90 : 旋转90度 180 : 旋转180度 270 : 旋转270度
content	字符串	设定数据内容 数据内容限制： 1) 数字数据: (数字 0~9) 2) 字母数据 数字 0-9 大写字母 A-Z 9 种其它字符: 空格, \$ % * + - . / :) *如果” A” 是数据内容的第一个字符，那么数据内容将会被设置为字母数据。 *如果” N” 是数据内容的第一个字符，那么数据内容将会被设置为数字数据。 * “！” 用来转换数据的格式，” N” 、” A” 等数据类型可通过” ！” 来转换。

12. `printerfont(x, y, size, rotation, x_scale, y_scale, content)`、

`printerfont_USB(x, y, size, rotation, x_scale, y_scale, content)`、

`printerfont_Ethernet(x, y, size, rotation, x_scale, y_scale, content)`

■ 函式说明：使用打印机内建文字打印

■ 参数说明：

参数	型别	说明
x	字符串	文字 X 方向起始点，以点(dot)表示
y	字符串	文字 Y 方向起始点，以点(dot)表示
size	字符串	内建字型名称 1: 8*/12 dots 2: 12*20 dots 3: 16*24 dots 4: 24*32 dots 5: 32*48 dots TST24.BF2: 繁体中文24*24 TST16.BF2: 繁体中文16*16 TSS24.BF2: 简体中文24*24 TSS16.BF2: 简体中文16*16
rotation	字符串	设定文字旋转角度 0: 旋转0度 90: 旋转90度 180: 旋转180度 270: 旋转 270 度
x_scale	字符串	设定文字 X 方向放大倍率，1~10
y_scale	字符串	设定文字Y方向放大倍率，1~10
content	字符串	设定欲打印之文字内容

13. `formfeed()`、`formfeed_USB()`、`formfeed_Ethernet()`

■ 函式说明：跳页，该函式需在 `setup` 后使用

■ 参数说明：无

14. nobackfeed()、nobackfeed_USB()、nobackfeed_Ethernet()

- 函式说明：设定纸张不回吐
- 参数说明：无

15. printlabel (set, copy)、printlabel _USB(set, copy)、printlabel _Ethernet(set, copy)

- 函式说明：打印标签内容
- 参数说明：
 - ➔ set: 字符串型别，设定打印标签式数(set)
 - ➔ copy: 字符串型别，设定打印标签份数(copy)

16. downloadpcx (filename,memoryname)、downloadpcx_USB (filename,memoryname)、downloadpcx_Ethernet(filename,memoryname)

- 函式说明：下载单色 PCX 格式图文件至打印机
- 参数说明：
 - ➔ filename: 字符串型别，文件名(可包含路径)
 - ➔ memoryname: 下载至打印机内存内之文件名(请使用大写档名)

17. downloadbmp (filename, memoryname)、downloadbmp_USB (filename, memoryname)、downloadbmp_Ethernet (filename, memoryname)

- 函式说明：下载单色 BMP 格式图文件至打印机
- 参数说明：
 - ➔ filename: 字符串型别，文件名(可包含路径)
 - ➔ memoryname: 下载至打印机内存内之文件名(请使用大写档名)

18. windowsfont (x,y,height,rotation,fontstyle,underline,fontname,content)

windowsfont_USB (x,y,height,rotation,fontstyle,underline,fontname,content)

windowsfont_Ethernet(x,y,height,rotation,fontstyle,underline,fontname,content)

■ 函式说明：使用 Windows TTF 字型打印文字

■ 参数说明：

参数	型别	说明
x	int	文字 X 方向起始点，以点(dot)表示
y	int	文字 Y 方向起始点，以点(dot)表示
height	int	字体高度，以点(dot)表示
rotation	int	设定旋转角度，逆时钟方向旋转 0：旋转0度 90：旋转90度 180：旋转180度 270：旋转 270 度
fontstyle	int	设定字体外型 0：标准(Normal) 1：斜体(Italic) 2：粗体(Bold) 3：粗斜体(Bold and Italic)
underline	int	设定底线 0：无底线 1：加底线
fontname	字符串	Windows字体名称，如: Arial, Times new Roman, 细明体, 标楷体
content	字符串	设定欲打印之文字内容

19. getDLLVersion (returnWay)、getDLLVersion_USB(returnWay)、getDLLVersion_Ethernet(returnWay)

■ 函式说明：回传此 SDK 版本号

■ 参数说明：

➔ retrunWay: int 型别，输入 0 除返回 SDK 版本号外，会跳出 SDK 版本讯息

20. printerstatus_USB ()、printerstatus_Ethernet()

- 函式说明：回传打印机状态，需用字符串变量接收回传讯息
- 参数说明：无
- 回传字符串说明：

回传字符串	打印机状态
00	就绪
01	上盖开启
02	卡纸
03	卡纸且上盖开启
04	标签用尽
05	标签用尽且上盖开启
08	碳带用尽
09	碳带用尽且上盖开启
0A	碳带用尽且卡纸
0B	碳带用尽、卡纸且上盖开启
0C	碳带用尽且标签用尽
0D	碳带用尽、标签用尽且上盖开启
10	暂停
20	打印中
80	其他错误

21. writeUHF (dataFormat,startBlockNo,byteSize,Gen2MemoryBank,datastring)

writeUHF_USB (dataFormat,startBlockNo,byteSize,Gen2MemoryBank,datastring)

writeUHF_Ethernet (dataFormat,startBlockNo,byteSize,Gen2MemoryBank,datastring)

- 函式说明：将数据写入 UHF Gen2/UHF GJB 卷标内存中
- 参数说明：

参数	型别	说明
dataFormat	string	设定字符串数据编码格式，默认为 H A: ASCII

		H: Hexadecimal
startBlockNo	int	设定数据区块起始位置，Gen2 默认为 2，GJB 默认为 1
byteSize	int	设定写入数据byte长度，默认为1
Gen2Memory Bank	string	设定 Gen2/GJB 数据区段，默认为 E R: 保留 E: EPC T: TID(Tag ID) U: User
datastring	string	欲写入之字符串数据

22. EPCPWD_Action (action, password)、EPCPWD_Action_USB (action, password)、

EPCPWD_Action_Ethernet (action, password)

- 函式说明：将 UHF GNE2 的 EPC 资料区块上锁或解锁
- 参数说明：

参数	型别	说明
action	string	设定执行动作 U: 解锁资料区块 L: 上锁资料区块 O: 永久解锁资料区块 P: 永久上锁资料区块
password	string	密码，应为 8 hex 字符(0~9,A,B,C,D,E,F)

23. TIDPWD_Action(action, password)、TIDPWD_Action_USB (action, password)、

TIDPWD_Action_Ethernet(action, password)

- 函式说明：将 UHF GNE2 的 TID 资料区块上锁或解锁
- 参数说明：

参数	型别	说明
action	string	设定执行动作 U: 解锁资料区块 L: 上锁资料区块 O: 永久解锁资料区块

		P: 永久上锁资料区块
password	string	密码, 应为 8 hex 字符(0~9,A,B,C,D,E,F)

24. USERPWD_Action (action, password)、USERPWD_Action_USB (action, password)、

USERPWD_Action_Ethernet (action, password)

- 函式说明: 将 UHF GNE2 的 USER 资料区块上锁或解锁
- 参数说明:

参数	型别	说明
action	string	设定执行动作 U: 解锁资料区块 L: 上锁资料区块 O: 永久解锁资料区块 P: 永久上锁资料区块
password	string	密码, 应为 8 hex 字符(0~9,A,B,C,D,E,F)

25. AccessPWD_Action (action, password)、AccessPWD_Action_USB (action, password)、

AccessPWD_Action_Ethernet (action, password)

- 函式说明: 将 UHF GNE2/UHF GJB 的访问密码进行设定、上锁或解锁
- 参数说明:

参数	型别	说明
action	string	设定执行动作 UHF GNE2 : U: 解锁访问密码 L: 上锁访问密码 O: 永久解锁访问密码 P: 永久上锁访问密码 S: 设定访问密码 UHF GJB : U: 解锁访问密码 S: 设定访问密码
password	string	密码, 应为 8 hex 字符(0~9,A,B,C,D,E,F)

26. KillPWD_Action (action, password)、KillPWD_Action_USB (action, password)、

KillPWD_Action_Ethernet (action, password)

- 函式说明：将 UHF GNE2 的删除密码进行设定、上锁或解锁
- 参数说明：

参数	型别	说明
action	string	设定执行动作 U：解锁删除密码 L：上锁删除密码 O：永久删除访问密码 P：永久删除访问密码 S：设定删除密码
password	string	密码，应为 8 hex 字符(0~9,A,B,C,D,E,F)

27. Set_RFIDPorcedure (tagType, rw_position, void_printout, tryEncodie_times,error_handle, speed, retry_times)

Set_RFIDPorcedure_USB (tagType, rw_position, void_printout, tryEncodie_times,error_handle, speed, retry_times)

Set_RFIDPorcedure_Ethernet (tagType, rw_position, void_printout, tryEncodie_times,error_handle, speed, retry_times)

- 函式说明：RFID 设定
- 参数说明：

参数	型别	说明
tagType	int	设定卷标类型，1~10，默认值为 8 1：EPC Class 1 Generation 2-Q，8：EPC Class 1 Generation 2-R，10：UHF-J
rw_position	int	设标签读写位置(卷标顶部起算)，范围为 0~9999(dot)，预设 0
void_printout	int	设定无效打印长度(dot)，范围为0~卷标长度，默认为卷标长度
tryEncodie_times	string	设定最大无效标签数，范围为 0~10，预设 3
error_handle	string	设定无效时采取的动作，预设 N

		N: No action(继续) P: Pause mode(暂停) E: Error mode(停止)
speed	int	设定无效打印速度，范围 2~10(IPS)，默认值 2(IPS)
retry_times	int	设定标签重试次数，范围 0~10，默认值 6

28. Set_RFIDPorcedure (tagType, rw_position, void_printout, tryEncodie_times,error_handle, speed, retry_times,dpi)

Set_RFIDPorcedure_mm_USB (tagType, rw_position, void_printout, tryEncodie_times,error_handle, speed, retry_times,dpi)

Set_RFIDPorcedure_mm_Ethernet (tagType, rw_position, void_printout, tryEncodie_times,error_handle, speed, retry_times,dpi)

■ 函式说明：RFID 设定

■ 参数说明：

参数	型别	说明
tagType	int	设定卷标类型，1~10，默认值为 8 1：EPC Class 1 Generation 2-Q，8：EPC Class 1 Generation 2-R，10：UHF-J
rw_position	double	设卷标读写位置(卷标顶部起算)， 范围为 203dpi:0 ~ 1251 (mm)、 300dpi:0 ~ 846 (mm)、 600dpi:0 ~ 423 (mm)，预设为 0
void_printout	double	设定无效打印长度(mm)，范围为0~卷标长度，默认为卷标长度
tryEncodie_times	int	设定最大无效标签数，范围为 0~10，预设为 3
error_handle	字符串	设定无效时采取的动作，预设为 N N：No action(继续) P：Pause mode(暂停) E：Error mode(停止)
speed	int	设定无效打印速度，范围 2~10(IPS)，默认值 2(IPS)
retry_times	int	设定标签重试次数，范围 0~10，默认值 6
dpi	字串	设定打印机的 DPI 203: 203 dpi 300: 300 dpi 600: 600 dpi

29. writeHF (dataFormat,startBlockNo,byteSize,datastring)

writeHF_USB (dataFormat,startBlockNo,byteSize,datastring)

writeHF_Ethernet (dataFormat,startBlockNo,byteSize,datastring)

■ 函式说明：将数据写入 HF 卷标内存中

■ 参数说明：

参数	型别	说明
dataFormat	string	设定字符串数据编码格式，默认为 H A: ASCII H: Hexadecimal
startBlockNo	int	设定数据区块起始位置，默认为 2
byteSize	int	设定写入数据byte长度，默认为1
datastring	string	欲写入之字符串数据

30. printerfontblock (x, y, width, height, fontname, rotation, x_scale, y_scale, space, align, content) 、

printerfontblock_USB(x, y, width, height, fontname, rotation, x_scale, y_scale,space, align,content) 、

printerfontblock_Ethernet(x, y, width, height, fontname, rotation, x_scale, y_scale, space, align, content)

■ 函式说明：打印段落文字内容

■ 参数说明：

参数	型别	说明
x	字符串	文字 X 方向起始点，以点(dot)表示
y	字符串	文字 Y 方向起始点，以点(dot)表示
width	字符串	设定段落区块宽度，以点(dot)表示
height	字符串	设定段落区块高度，以点(dot)表示
fontname	字符串	内建字型名称 1: 8*12 dots 2: 12*20 dots 3: 16*24 dots 4: 24*32 dots 5: 32*48 dots TST24.BF2: 繁体中文24*24 TST16.BF2: 繁体中文16*16

		TSS24.BF2: 简体中文24*24 TSS16.BF2: 简体中文16*16
rotation	字符串	设定旋转角度，逆时钟方向旋转 0: 旋转0度 90: 旋转90度 180: 旋转180度 270: 旋转 270 度
x_scale	字符串	设定文字 X 方向放大倍率，1~10
y_scale	字符串	设定文字Y方向放大倍率，1~10
space	字符串	行距，以点(dot)表示
align	字符串	对齐位置 0: 预设(置左) 1: 置左 2: 置中 3: 置右
content	字符串	欲写入之字符串数据

31. readUHF_USB (dataFormat,startBlockNo,byteSize,Gen2MemoryBank)

readUHF_Ethernet (dataFormat,startBlockNo,byteSize,Gen2MemoryBank)

■ 函式说明：读取 UHF Gen2/UHF GJB 卷标内存数据

■ 参数说明：

参数	型别	说明
dataFormat	string	设定字符串数据编码格式，默认为 H A: ASCII H: Hexadecimal
startBlockNo	int	设定数据区块起始位置，默认为 0
byteSize	int	设定读取数据byte长度，默认为1
Gen2Memory Bank	string	设定 Gen2/GJB 数据区段，默认为 E R: 保留 E: EPC T: TID(Tag ID) U: User

■ 回传字符串说明(标签资料):

dataFormat	回传字符串(范例)
A	标签资料以 ASCII 显示 (ex: 24051324000103456400)
H	标签资料以 Hexadecimal 显示 (ex: 3234303531333234303030313033343536343030)

■ 回传字符串说明(错误代码)：

回传字符串	错误代码说明
64000000000000000000000000000000	其他错误
65000000000000000000000000000000	超过记忆体空间
66000000000000000000000000000000	记忆体被锁住
67000000000000000000000000000000	读取功率不足
68000000000000000000000000000000	非特定的错误
69000000000000000000000000000000	CRC错误
6A000000000000000000000000000000	写入中若发生错误时，回覆已写入多少 words 数
6B000000000000000000000000000000	写入中若 Tag 标签回覆错误时，错误码加上已写入多少 words 数
6C000000000000000000000000000000	没有标签存在
6D000000000000000000000000000000	指令格式错误
6E000000000000000000000000000000	设定电源强度失败
6F000000000000000000000000000000	设定法规失败

32. setPWD_Action(passwordArea, action, NewPassword, WritePassword)

setPWD_Action_USB(passwordArea, action, NewPassword, WritePassword)

setPWD_Action_Ethernet(passwordArea, action, NewPassword, WritePassword)

■ 函式说明：设定 UHF GJB 各密码区域新密码

■ 参数说明：

参数	型别	说明
passwordArea	string	设定密码区域，预设为 W K：Kill W：Write R：Read S：Status
action	string	设定动作

		S : Set Password
NewPassword	string	设定密码区域的新密码，应为8 hex字元(0~9,A,B,C,D,E,F)
WritePassword	string	写入密码，应为 8 hex 字元(0~9,A,B,C,D,E,F)

33. writeGJB_UHF(dataFormat, startBlockNo, byteSize, Gen2MemoryBank, datastring, WritePassword)
writeGJB_UHF_USB(dataFormat, startBlockNo, byteSize, Gen2MemoryBank, datastring, WritePassword)
writeGJB_UHF_Ethernet(dataFormat, startBlockNo, byteSize, Gen2MemoryBank, datastring, WritePassword)

■ 函式说明：将资料写入 UHF GJB 标签记忆体中

■ 参数说明：

参数	型别	说明
dataFormat	string	设定字串资料编码格式，预设 H A : ASCII H : Hexadecimal
startBlockNo	int	设定资料区块起始位置，GJB 预设 1
byteSize	int	设定写入资料byte长度，预设 1
Gen2MemoryBank	string	设定 GJB 资料区段，预设 E R : 安全区 E : EPC T : TID(Tag ID) U : User
datastring	string	欲写入之字串资料
WritePassword	string	写入密码，应为 8 hex 字元(0~9,A,B,C,D,E,F)

34. readGJB_UHF_USB(dataFormat, startBlockNo, byteSize, Gen2MemoryBank, ReadPassword)
readGJB_UHF_Ethernet(dataFormat, startBlockNo, byteSize, Gen2MemoryBank, ReadPassword)

■ 函式说明：读取 UHF GJB 标签记忆体资料，需用字符串变数接收回传讯息

■ 参数说明：

参数	型别	说明
dataFormat	string	设定字串资料编码格式，预设 H

		A : ASCII H : Hexadecimal
startBlockNo	int	设定资料区块起始位置，预设 0
byteSize	int	设定写入资料byte长度，预设 1
Gen2MemoryBank	string	设定 GJB 资料区段，预设 E R : 安全区 E : EPC T : TID(Tag ID) U : User
ReadPassword	string	读取密码，应为 8 hex 字元(0~9,A,B,C,D,E,F)

■ 回传字符串说明：

dataFormat	回传字符串(范例)
A	标签资料以 ASCII 显示 (ex: 24051324000103456400)
H	标签资料以 Hexadecimal 显示 (ex: 3234303531333234303030313033343536343030)

35. statusGJB_UHF(Gen2MemoryBank, action, StatusPassword)

statusGJB_UHF_USB(Gen2MemoryBank, action, StatusPassword)

statusGJB_UHF_Ethernet(Gen2MemoryBank, action, StatusPassword)

■ 函式说明：设定 UHF GJB 各资料区块读写状态

■ 参数说明：

参数	型别	说明
Gen2MemoryBank	string	设定 GJB 资料区段，预设 E F : 安全区 E : EPC T : TID(Tag ID) U : User
action	string	设定状态，预设 A A=Lock0(可读可写) B=Lock1(可读不可写) C=Lock2(不可读可写) D=Lock3(不可读不可写)
StatusPassword	string	状态密码，应为 8 hex 字元(0~9,A,B,C,D,E,F)

36. killGJB_UHF(KillPassword)

killGJB_UHF_USB(KillPassword)

killGJB_UHF_Ethernet(KillPassword)

■ 函式说明：删除 UHF GJB 标签

■ 参数说明：

参数	型别	说明
KillPassword	string	删除密码，应为 8 hex 字元(0~9,A,B,C,D,E,F)

37. query_UHF_USB(dataFormat, PCReturnStatus, CRCReturnStatus)

query_UHF_Ethernet (dataFormat, PCReturnStatus, CRCReturnStatus)

■ 函式说明：以 Q 指令读取 UHF Gen2 标签记忆体 EPC 资料区段资料，需用字串变数接收回传讯息

■ 参数说明：

参数	型别	说明
dataFormat	string	设定字串资料编码格式，预设 H A：ASCII H：Hexadecimal
PCReturnStatus	int	PC 返回状态，预设 0 0：不回传 PC 值 1：回传 PC 值
CRCReturnStatus	int	CRC-16 返回状态，预设 0 0：不回传 CRC-16 1：回传CRC-16

■ 回传字符串说明(标签资料)：

以 query_UHF_USB("A", 1, 1)和 query_UHF_USB("H", 1, 1)为例：(PC 值和 CRC-16 皆回传)

dataFormat	回传字符串(范例)
A	标签资料以 ASCII 显示 (ex: 24051324000103456400)
H	标签资料以 Hexadecimal 显示 (ex: 3234303531333234303030313033343536343030)

以 query_UHF_USB("A", 0, 0)和 query_UHF_USB("H", 0, 0)为例：(PC 值和 CRC-16 皆不回传)

dataFormat	回传字符串(范例)
------------	-----------

A	标签资料以 ASCII 显示 (ex: 0513240001034564)
H	标签资料以 Hexadecimal 显示 (ex: 30353133323430303031303334353634)

■ 回传字符串说明(错误代码)：

回传字符串	错误代码说明
64000000000000000000000000000000	其他错误
65000000000000000000000000000000	超过记忆体空间
66000000000000000000000000000000	记忆体被锁住
67000000000000000000000000000000	读取功率不足
68000000000000000000000000000000	非特定的错误
69000000000000000000000000000000	CRC错误
6A000000000000000000000000000000	写入中若发生错误时，回覆已写入多少 words 数
6B000000000000000000000000000000	写入中若 Tag 标签回覆错误时，错误码加上已写入多少 words 数
6C000000000000000000000000000000	没有标签存在
6D000000000000000000000000000000	指令格式错误
6E000000000000000000000000000000	设定电源强度失败
6F000000000000000000000000000000	设定法规失败

38. RFIDAutoCalibration()、RFIDAutoCalibration_USB()、RFIDAutoCalibration_Ethernet()

- 函式说明：进行 RFID 标签自动校准至最佳位置
- 参数说明：无

39. setDirectionAndMirror(direction,mirror)、setDirectionAndMirror_USB(direction,mirror)、setDirectionAndMirror_Ethernet(direction,mirror)

- 函式说明：设定标签打印时的出纸方向与是否使用镜像打印
- 参数说明：

参数	型别	说明
direction	int	设定出纸方向，预设为 0 0：顶端出纸 1：底端出纸
mirror	int	设定是否镜像打印

		0：否 1：是
--	--	------------

40. setShift(shiftY)、setShift_USB(shiftY)、setShift_Ethernet(shiftY)

- 函式说明：设定图像垂直位移距离，数值为正时，图像会往打印方向移动，数值为负时，图像会背离打印方向
- 参数说明：
 - ➔ shiftY：int 型别，垂直位移距离，单位为 dot

41. printReverse(x_start, y_start, x_width, y_height)、printReverse_USB(x_start, y_start, x_width, y_height)、printReverse_Ethernet(x_start, y_start, x_width, y_height)

- 函式说明：将指定的区域于打印时反白
- 参数说明：

参数	型别	说明
x_start	int	指定 X 起始坐标位置，以点(dot)表示
y_start	int	指定 Y 起始坐标位置，以点(dot)表示
x_width	int	指定 X 坐标宽度，以点(dot)表示
y_height	int	指定 Y 坐标高度，以点(dot)表示

42. setOffset(offset)、setOffset_USB(offset)、setOffset_Ethernet(offset)

- 函式说明：设定每次出纸后额外偏移的距离(通常与剥纸模式和裁切模式组合使用)
- 参数说明：
 - ➔ offset：double 型别，额外的出纸偏移，单位为 mm

43. setCutMode(mode, piece)、setCutMode_USB(piece)、setCutMode_Ethernet(mode, piece)

- 函式说明：设定裁切模式与张数
- 参数说明：

参数	型别	说明
mode	int	设定裁切方式，预设为 1 0：反切 1：正切
piece	int	设定裁切张数

44. setAfterPrintAction(mode)、setAfterPrintAction_USB(mode)、setAfterPrintAction_Ethernet(mode)

- 函式说明：设定打印后动作
- 参数说明：

参数	型别	说明
mode	int	设定打印后动作，预设值为 1 0：停在原地 1：撕纸 2：剥纸 3：裁切

45. genericDefault ()、genericDefault_USB ()、genericDefault_Ethernet()

- 函式说明：将打印机之一般设定值初始化
- 参数说明：无

46. sensorDefault ()、sensorDefault_USB ()、sensorDefault_Ethernet ()

- 函式说明：将打印机之传感器设定值初始化
- 参数说明：无

47. rfidSetupDefault ()、rfidSetupDefault_USB ()、rfidSetupDefault_Ethernet ()

- 函式说明：将 RFID 设定值初始化
- 参数说明：无

48. WifiFrequency(Frequency)、WifiFrequency_USB(Frequency)、WifiFrequency_Ethernet (Frequency)

- 函式说明：使用兼容 5G 频段 WIFI 模块时，可用于切换使用频段
- 参数说明：

参数	型别	说明
Frequency	string	设定模块频段 2.4G：使用 2.4G 频段 5G：使用 5G 频段 BOTH：使用双频频段

49. Bitmap(x,y, width,height,mode,filename)

Bitmap_USB(x,y, width,height,mode,filename)

Bitmap_Ethernet(x,y, width,height,mode,filename)

■ 函式说明：将图片转为单色点阵图，使用打印机直接打印

■ 参数说明：

参数	型别	说明
x	string	文字 X 方向起始点，以点(dot)表示
y	string	文字 Y 方向起始点，以点(dot)表示
width	int	图片宽度，以字节(byte)表示
height	int	图片高度，以点(dot)表示
mode	int	图片格式 0: OVERWRITE 1: OR 2: XOR
filename	string	档案名称(可包含路径) 图档仅支援以下格式： 1. BMP (Bitmap)：位图格式 2. JPG (JPEG)：压缩的图像格式 3. PNG (Portable Network Graphics)：无损压缩的图像格式 4. GIF (Graphics Interchange Format)：支援多张图片的格式，通常用于动画 5. TIFF (Tagged Image File Format)：高品质的无损压缩图像格式 6. ICO (Icon)：图示格式，用于显示档案、程式或资料夹的图示 7. WMF (Windows Metafile)：Windows 绘图文件格式 8. EMF (Enhanced Metafile)：扩展的 Windows 绘图文件格式

50. compressBitmap(x,y, width,height,filename)

compressBitmap_USB(x,y, width,height,filename)

compressBitmap_Ethernet(x,y, width,height,filename)

- 函式说明：将图片转为单色点阵图，压缩后再使用打印机打印
- 参数说明：

参数	型别	说明
x	string	文字 X 方向起始点，以点(dot)表示
y	string	文字 Y 方向起始点，以点(dot)表示
width	int	图片宽度，以字节(byte)表示
height	int	图片高度，以点(dot)表示
filename	string	档案名称(可包含路径) 图档仅支援以下格式： 1. BMP (Bitmap)：位图格式 2. JPG (JPEG)：压缩的图像格式 3. PNG (Portable Network Graphics)：无损压缩的图像格式 4. GIF (Graphics Interchange Format)：支援多张图片的格式，通常用于动画 5. TIFF (Tagged Image File Format)：高品质的无损压缩图像格式 6. ICO (Icon)：图示格式，用于显示档案、程式或资料夹的图示 7. WMF (Windows Metafile)：Windows 绘图文件格式 8. EMF (Enhanced Metafile)：扩展的 Windows 绘图文件格式

51. setResponse_USB(jobName, Mode)

- 函式说明：自动响应指令
- 参数说明：

参数	型别	说明
jobName	string	打印任务名称。设置任务 ID，默认值为 Null
Mode	string	自动响应模式： ON:打开自动响应功能，任务开始每打印一张的开始与结束或途中有异常状态皆返回打印任务状态 OFF:关闭自动响应功能 ※ 打印内容一定要使用 printlabel 函式打印，后续才能成功接收

52. setResponseReadFromPrinter_USB()

- 函式说明：当 setResponse 为 ON，接收回应状态
- 参数说明：无
- 回传字符串说明：

字符串型别，回传格式：{ 打印机状态,任务编号,任务名称,任务状态}

Ex: {20,000001,Test,START}，START: 开始打印时返回

{01,000001, ,ERR}，ERR: 进入错误时

{20,000001, ,ERR}，ERR: 退出错误时

{20,000001, ,FS}，FS: 开盖后开始校准

{20,000001,Test,START}，START: 错误后再次重新开始打印时返回

{20,000001,Test,STOP}，STOP: 结束打印时返回

任务编号：共 6 码，000001~999999(连续列印时，会从 000001 向上计数)

回传字符串	打印机状态
00	就绪
01	上盖开启
02	卡纸
03	卡纸且上盖开启
04	标签用尽
05	标签用尽且上盖开启
08	碳带用尽
09	碳带用尽且上盖开启
0A	碳带用尽且卡纸
0B	碳带用尽、卡纸且上盖开启
0C	碳带用尽且标签用尽
0D	碳带用尽、标签用尽且上盖开启
10	暂停
20	打印中
80	其他错误

53. setResponseReceiveSTOP_USB()

- 函式说明：判断 setResponse_USB 任务是否结束，停止接收
- 参数说明：无
- 回传说明：int 型别，若为 0，表示任务还未结束；若为 1，表示任务结束

● C#(.Netframework)

1.需先将 GTSPL_SDK.dll 加入参考

2.汇入 GTSPL_SDK:

```
using GTSPL_SDK;
```

3.范例程序:

```
//单机打印
```

```
Driver driver = new Driver();
```

```
driver.openport("Printer Model");
```

```
driver.setup("54", "30", "2", "3", "0", "3", "0");
```

```
driver.sendcommand("DIRECTION 1");
```

```
driver.setDirectionAndMirror(1, 1);
```

```
driver.setShift(50);
```

```
driver.setAfterPrintAction(2);
```

```
driver.setOffset(20);
```

```
driver.setCutMode(0,2);
```

```
driver.clearbuffer();
```

```
driver.sendcommand("TEXT 100,100,\"3\",0,1,1,\"REVERSE\");
```

```
driver.printReverse(90, 90, 128, 40);
```

```
driver.barcode("30", "30", "128", "100", "1", "0", "2", "2", "barcode1234567");
```

```
driver.qrcode("220", "260", "M", "3", "A", "0", "AABCB03abcN123");
```

```
driver.printerfont("50", "10", "3", "0", "1", "1", "Print Font 123456");
```

```
driver.printerfontblock("35", "15", "790", "90", "1", "0", "8", "8", "0", "1", "We stand behind our products  
with one of the most comprehensive support programs in the Auto-ID industry.");
```

```
string file = Environment.CurrentDirectory;
```

```

driver.downloadbmp(file + "\\CIRCLE.BMP", "CIRCLE.BMP");

driver.sendcommand("PUTBMP 150,30,\"CIRCLE.BMP\"");

driver.windowfont(100, 20, 48, 0, 0, 0, "arial", "C# Driver test");

//BitmapResult 为 Bitmap 处理结果，成功为 1；失败为 0

int BitmapResult = driver.Bitmap("-500", "70", 400, 350, 1, "Thunder.png ");

BitmapResult = driver.compressBitmap("-600", "70", 666, 357, "Cat1.jpg");

driver.printlabel("1", "1");

driver.genericDefault();

driver.sensorDefault();

driver.WifiFrequency("5G");

driver.writeUHF("H", 2, 12, "E", "414142424343444445454646"); //startBlockNo: Gen2 默认为 2

driver.printlabel("1", "1");

driver.EPCPWD_Action("U", "12345678");

driver.TIDPWD_Action("L", "12345678");

driver.USERPWD_Action("L", "12345678");

driver.AccessPWD_Action("S", "12345678");

driver.KillPWD_Action("L", "12345678");

driver.Set_RFIDPorcedure(8, 8, 32, 3, "N", 2, 2);

driver.Set_RFIDPorcedure_mm(5, 40, 32, 5, "N", 5, 5, "203");

driver.writeHF("H", 2, 12, "414142424343444445454646");

driver.printlabel("1", "1");

//RFID

driver.RFIDAutoCalibration();

driver.rfidSetupDefault();

//UHF GJB 写入资料

```

```
driver.writeGJB_UHF("H", 1, 12, "E", "414142424343444445454646", "12345678");
```

*带入写入密码，写入资料(startBlockNo: GJB 预设为 1)

//UHF GJB 设定各密码区域新密码

```
driver.setPWD_Action("S", "S", "11112222", "12345678"); //带入写入密码，设定新状态密码
driver.setPWD_Action("R", "S", "33334444", "12345678"); //带入写入密码，设定新读取密码
driver.setPWD_Action("K", "S", "55556666", "12345678"); //带入写入密码，设定新删除密码
driver.setPWD_Action("W", "S", "87654321", "12345678"); //带入旧写入密码，设定新写入密码
```

//UHF GJB 设定资料区块状态

```
driver.statusGJB_UHF("E", "B", "11112222"); //带入状态密码，设定状态
```

```
driver.printlabel("1", "1");
```

//UHF GJB 删除标签

```
driver.killGJB_UHF ("55556666"); //带入删除密码，删除标签
```

```
driver.printlabel("1", "1");
```

```
driver.closeport();
```

//指定 USB 传输接口

```
USB usb = new USB();
```

```
usb.openport_USB();
```

```
string[] PrinterName = usb.detectUSB_USB(); //假设侦测打印机有一台以上
```

```
string[] PrinterName = usb.detectUSBStr_USB() == null ? null : usb.detectUSBStr_USB().Split('\n');
```

*先判断 usb.detectUSBStr_USB()是否为 null，若不是，再以'\n'区分各个机型

```
usb.openports_USB(PrinterName[0]); //利用侦测到的第一台打印机连线
```

```
usb.setup_USB("54", "30", "2", "3", "0", "3", "0");
```

```
usb.sendcommand_USB("DIRECTION 1");
```

```
usb.setDirectionAndMirror_USB(1,1);
```

```
usb.setShift_USB(50);
```

```

usb.setAfterPrintAction_USB(2);

usb.setOffset_USB(20);

usb.setCutMode_USB(0,2);


usb.clearbuffer_USB();

usb.sendcommand_USB("TEXT 100,100,\"3\",0,1,1,\"REVERSE\");

usb.printReverse_USB(90,90,128,40);

usb.barcode_USB("30", "30", "128", "100", "1", "0", "2", "2", "barcode1234567");

usb.qrcode_USB("220", "260", "M", "3", "A", "0", "AABCB03abcN123");

usb.printerfont_USB("50", "10", "3", "0", "1", "1", "Print Font 123456");

string file = Environment.CurrentDirectory;

usb.downloadpcx_USB(file + "\\UL.PCX", "UL.PCX");

usb.windowfont_USB(10, 100, 48, 0, 0, 0, "arial", "C# WIN TEST");

//BitmapResult 为 Bitmap 处理结果，成功为 1；失败为 0

int BitmapResult = usb.Bitmap_USB("-500", "70", 400, 350, 1, "Thunder.png");

BitmapResult=usb.compressBitmap_USB("-600", "70", 666, 357, "Cat1.jpg");

usb.printlabel_USB("1", "1");

var status = usb.printerstatus_USB();

usb.genericDefault_USB();

usb.sensorDefault_USB();

usb.WifiFrequency_USB("5G");

//简中打印

string stString="默认简体中文测试";

usb.clearbuffer_USB();

usb.printerfont_USB ("100", "10", "TSS24.BF2", "0", "1", "1", stString);

usb.printlabel_USB (1, 1, this);

```

//繁中打印

```
string ttString="默認繁體中文測試";

usb.clearbuffer_USB();

usb.printerfont_USB("100", "10", " TST24.BF2", "0", "1", "1", ttString);

usb.printlabel_USB(1, 1, this);
```

//BLOCK 打印

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

usb.clearbuffer_USB();

usb.printerfontblock_USB("35","15","790","90","1","0","8","8","0","1", ttString);

usb.printlabel_USB(1, 1, this);
```

//RFID

```
usb.RFIDAutoCalibration_USB();

usb.rfidSetupDefault_USB();
```

//UHF GEN2

```
usb.writeUHF_USB("H", 2, 12, "E", "414142424343444445454646"); //startBlockNo: Gen2 默认为 2
string data = usb.readUHF_USB("H", 0, 12, "E");
string data_Q = usb.query_UHF_USB("A", 0, 0); //以 Q 指令读取 EPC 资料
usb.EPCPWD_Action_USB("L", "12345678");
usb.TIDPWD_Action_USB("L", "12345678");
usb.USERPWD_Action_USB("U", "12345678");
usb.AccessPWD_Action_USB("L", "12345678");
usb.KillPWD_Action_USB("L", "12345678");
usb.Set_RFIDPorcedure_USB(8,8,32,3,"N",2,2);
usb.Set_RFIDPorcedure_mm_USB(5, 40, 32, 5, "N", 5, 5, "203");
```

```
usb.writeHF_USB("H", 2, 12, "414142424343444445454646");
```

```
usb.printlabel_USB("1", "1");
```

```
//UHF GJB 写入资料
```

```
usb.writeGJB_UHF_USB ("H", 1, 12, "E", "414142424343444445454646", "12345678");
```

*带入写入密码，写入资料(startBlockNo: GJB 默认为 1)

```
//UHF GJB 设定各密码区域新密码
```

```
usb.setPWD_Action_USB ("S", "S", "11112222", "12345678"); //带入写入密码，设定新状态密码
```

```
usb.setPWD_Action_USB ("R", "S", "33334444", "12345678"); //带入写入密码，设定新读取密码
```

```
usb.setPWD_Action_USB ("K", "S", "55556666", "12345678"); //带入写入密码，设定新删除密码
```

```
usb.setPWD_Action_USB ("W", "S", "87654321", "12345678"); //带入旧写入密码，设定新写入密码
```

```
//UHF GJB 设定资料区块状态
```

```
usb.statusGJB_UHF_USB ("E", "B", "11112222"); //带入状态密码，设定状态
```

```
usb.printlabel_USB ("1", "1");
```

```
//UHF GJB 读取资料
```

```
string data =usb.readGJB_UHF_USB("H", 0, 12, "E", "33334444"); //带入读取密码，读取标签资料
```

```
//UHF GJB 删除标签
```

```
usb.killGJB_UHF_USB ("55556666"); //带入删除密码，删除标签
```

```
usb.printlabel_USB ("1", "1");
```

```
//setResponse 自动响应功能
```

```
Timer receiveResponseTimer; //接收响应指令计时器
```

```
private void SetResponseBtn_Click(object sender, RoutedEventArgs e)
```

```
{
```

```
    usb.openport_USB();
```

```
    usb.setResponse_USB("Test", "ON");
```

```
    usb.sendcommand_USB("SIZE 4,2");
```



```

usb.sendcommand_USB("GAP 0,0");

usb.printlabel_USB("1", "5");    //打印标签必须使用 printlabel 才可正常接收

Task.Factory.StartNew(() =>
{
    startReceiveResponseTimer();    //开启自动响应 Timer
});
}

//开启自动响应 Timer

public void startReceiveResponseTimer()
{
    receiveResponseTimer = new Timer();
    receiveResponseTimer.Interval = 800;
    receiveResponseTimer.Elapsed += timer_Elapsed;
    receiveResponseTimer.Start();
}

//Timer 的处理

public void timer_Elapsed(object sender, ElapsedEventArgs e)
{
    Dispatcher.BeginInvoke(new Action(() =>
    {
        string ReceiveStr=usb.getResponseReadFromPrinter_USB();
        if (usb.getResponseReceiveSTOP_USB() == 1)
        {
            usb.getResponse_USB("Test", "OFF");
            stopReceiveResponseTimer();
        }
    }
    ));
}

```

```

    });
}
//关闭自动响应 Timer
public void stopReceiveResponseTimer()
{
    if (receiveResponseTimer != null)
    {
        receiveResponseTimer.Stop();
        receiveResponseTimer.Dispose();
        receiveResponseTimer = null;
    }
}

```

```

usb.closeport_USB();
usb.getDLLVersion_USB(1);

```

//指定网口传输介面

```

SocketConnect socketconnect = new SocketConnect();
socketconnect.openport_Ethernet("192.168.66.177", 9100);
socketconnect.setup_Ethernet ("54", "30", "2", "3", "0", "3", "0");
socketconnect.sendcommand_Ethernet ("DIRECTION 1");
socketconnect.setDirectionAndMirror_Ethernet(1,1);
socketconnect.setShift_Ethernet(50);
socketconnect.setAfterPrintAction_Ethernet(2);
socketconnect.setOffset_Ethernet(20);
socketconnect.setCutMode_Ethernet(0,2);

```

```

socketconnect.clearbuffer_Ethernet ();

socketconnect.sendcommand_Ethernet("TEXT 100,100,\"3\",0,1,1,\"REVERSE\"");

socketconnect.printReverse_Ethernet(90, 90, 128, 40);

socketconnect.barcode_Ethernet ("30", "30", "128", "100", "1", "0", "2", "2", "barcode1234567");

socketconnect.qrcode_Ethernet ("220", "260", "M", "3", "A", "0", "AABCB03abcN123");

socketconnect.printerfont_Ethernet ("50", "10", "3", "0", "1", "1", "Print Font 123456");

string file = Environment.CurrentDirectory;

socketconnect.downloadpcx_Ethernet (file + "\\UL.PCX", "UL.PCX");

socketconnect.windowfont_Ethernet (10, 100, 48, 0, 0, 0, "arial", "C# WIN TEST");

//BitmapResult 为 Bitmap 处理结果，成功为 1；失败为 0

int BitmapResult = socketconnect.Bitmap_Ethernet("-500", "70", 400, 350, 1, "Thunder.png");

BitmapResult = socketconnect.compressBitmap_Ethernet("-600", "70", 666, 357, "Cat1.jpg");

socketconnect.printlabel_Ethernet ("1", "1");

var status = socketconnect.printerstatus_Ethernet ();

socketconnect.genericDefault_Ethernet();

socketconnect.sensorDefault_Ethernet();

socketconnect.WifiFrequency_Ethernet("5G");

//简中打印

string stString="默认简体中文测试";

socketconnect.clearbuffer_Ethernet ();

socketconnect.printerfont_Ethernet ("100", "10", "TSS24.BF2", "0", "1", "1", stString);

socketconnect.printlabel_Ethernet (1, 1, this);

//繁中打印

string ttString="默認繁體中文測試";

socketconnect.clearbuffer_Ethernet ();

socketconnect.printerfont_Ethernet ("100", "10", "TST24.BF2", "0", "1", "1", ttString);

```

```
socketconnect.printlabel_Ethernet (1, 1, this);
```

```
//BLOCK 打印
```

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

```
socketconnect.clearbuffer_Ethernet ();
```

```
socketconnect.printerfontblock_Ethernet ("35","15","790","90","1","0","8","8","0","1", ttString);
```

```
socketconnect.printlabel_Ethernet (1, 1, this);
```

```
//RFID
```

```
socketconnect.RFIDAutoCalibration_Ethernet();
```

```
socketconnect.rfidSetupDefault_Ethernet();
```

```
//UHF GEN2
```

```
socketconnect.writeUHF_Ethernet("H", 2, 12,"E","414142424343444445454646"); //startBlockNo :
```

```
Gen2 默认为 2
```

```
socketconnect.printlabel_Ethernet ("1", "1");
```

```
string data = socketconnect.readUHF_Ethernet ("H", 0, 12, "E");
```

```
string data_Q = socketconnect.query_UHF_Ethernet ( "A", 0, 0); //以 Q 指令读取 EPC 资料
```

```
socketconnect.EPCPWD_Action_Ethernet ("L","12345678");
```

```
socketconnect.TIDPWD_Action_Ethernet ("L", "12345678");
```

```
socketconnect.USERPWD_Action_Ethernet ("L", "12345678");
```

```
socketconnect.AccessPWD_Action_Ethernet ("U", "12345678");
```

```
socketconnect.KillPWD_Action_Ethernet ("U", "12345678");
```

```
socketconnect.Set_RFIDPorcedure_Ethernet (8,8,32,3,"N",2,2);
```

```
socketconnect.Set_RFIDPorcedure_mm_Ethernet(5, 40, 32, 5, "N", 5, 5, "203");
```

```
socketconnect.writeHF_Ethernet ("H", 0, 12, "414142424343444445454646");
```

```
socketconnect.printlabel_Ethernet ("1", "1");
```

//UHF GJB 写入资料

```
socketconnect. writeGJB_UHF_Ethernet ("H", 1, 12, "E", "414142424343444445454646", "12345678");
```

*带入写入密码，写入资料(startBlockNo: GJB 默认为 1)

//UHF GJB 设定各密码区域新密码

```
socketconnect. setPWD_Action_Ethernet ("S", "S", "11112222", "12345678"); //带入写入密码，设定  
新状态密码
```

```
socketconnect. setPWD_Action_Ethernet ("R", "S", "33334444", "12345678"); //带入写入密码，设定  
新读取密码
```

```
socketconnect. setPWD_Action_Ethernet ("K", "S", "55556666", "12345678"); //带入写入密码，设定  
新删除密码
```

```
socketconnect. setPWD_Action_Ethernet ("W", "S", "87654321", "12345678"); //带入旧写入密码，设定  
新写入密码
```

//UHF GJB 设定资料区块状态

```
socketconnect. statusGJB_UHF_Ethernet ("E", "B", "11112222"); //带入状态密码，设定状态
```

```
socketconnect. printlabel_Ethernet ("1", "1");
```

//UHF GJB 读取资料

```
string data = socketconnect. readGJB_UHF_Ethernet ("H", 0, 12, "E", "33334444"); //带入读取密码，  
读取标签资料
```

//UHF GJB 删除标签

```
socketconnect. killGJB_UHF_Ethernet ("55556666"); //带入删除密码，删除标签
```

```
socketconnect. printlabel_Ethernet ("1", "1");
```

```
socketconnect.closeport_Ethernet ();
```

```
socketconnect.getDLLVersion_Ethernet (1);
```

● Java

1.需先汇入 jan-5.5.0.jar:

```
import com.sun.jna.Library;

import com.sun.jna.Native;
```

2.建立调用 dll 的 class 调用 dll

```
class GTSPL {

    interface GTSPL_SDK extends Library {

        GTSPL_SDK INSTANCE = (GTSPL_SDK) Native.load("GTSPL_SDK_C", GTSPL_SDK.class);

        int openport_USB();

        int openport(String PrinterName);

        ....

    }

}
```

3.范例程序:

```
System.load(System.getProperty("user.dir") + "\\GTSPL_SDK_C.dll");
```

```
//单机打印
```

```
GTSPL.GTSPL_SDK.INSTANCE.openport("Printer Model");
```

```
GTSPL.GTSPL_SDK.INSTANCE.setup("54", "30", "2", "3", "0", "3", "0");
```

```
GTSPL.GTSPL_SDK.INSTANCE.sendcommand("DIRECTION 1");
```

```
GTSPL.GTSPL_SDK.INSTANCE.setDirectionAndMirror(1, 1);
```

```
GTSPL.GTSPL_SDK.INSTANCE.setShift(50);
```

```
GTSPL.GTSPL_SDK.INSTANCE.setAfterPrintAction(2);
```

```
GTSPL.GTSPL_SDK.INSTANCE.setOffset(20);
```

```
GTSPL.GTSPL_SDK.INSTANCE.setCutMode(0,2);
```

```
GTSPL.GTSPL_SDK.INSTANCE.clearbuffer();
```

```
GTSPL.GTSPL_SDK.INSTANCE.sendcommand("TEXT 100,100,\"3\",0,1,1,\"REVERSE\");
```

SDK Version 1.10.5 / Instructions Version 1.25

```

GTSP.LGTSP.L_SDK.INSTANCE.printReverse(90, 90, 128, 40);

GTSP.LGTSP.L_SDK.INSTANCE.barcode("30", "30", "128", "100", "1", "0", "2", "2",
"barcode1234567");

GTSP.LGTSP.L_SDK.INSTANCE.qrcode("50", "100", "H", "4", "A", "0", "QRcode1234567");

GTSP.LGTSP.L_SDK.INSTANCE.printerfont("50", "10", "2", "0", "1", "1", "Print Font 123456");

GTSP.LGTSP.L_SDK.INSTANCE.downloadbmp(System.getProperty("user.dir") + "\\CIRCLE.BMP",
"CIRCLE.BMP");

GTSP.LGTSP.L_SDK.INSTANCE.windowfont(5, 150, 48, 0, 0, 0, "arial", "JAVA WIN DRIVER");

GTSP.LGTSP.L_SDK.INSTANCE.sendcommand("PUTBMP 150,30,\"CIRCLE.BMP\"");

GTSP.LGTSP.L_SDK.INSTANCE.printerfontblock("15", "15", "790", "90", "0", "0", "8", "8", "20",
"2", "We stand behind our products with one of the most comprehensive support programs in the
Auto-ID industry.");

```

//初始化

```

GTSP.LGTSP.L_SDK.INSTANCE.genericDefault();

GTSP.LGTSP.L_SDK.INSTANCE.sensorDefault();

GTSP.LGTSP.L_SDK.INSTANCE.rfidSetupDefault();

```

//修改 Wifi 频段

```

GTSP.LGTSP.L_SDK.INSTANCE.WifiFrequency("5G");

```

//BLOCK 打印

```

WString str = new WString("-- 默认简体中文测试：海天米醋白米醋 1 瓶 450ml --");

GTSP.LGTSP.L_SDK.INSTANCE.printerfontblockUnicode("15", "15", "790", "90", "TSS24.BF2", "0", "1",
"1", "0", "1", str);

GTSP.LGTSP.L_SDK.INSTANCE.printlabel("1", "1");

```

```

GTSP.LGTSP.L_SDK.INSTANCE.closeport();

GTSP.LGTSP.L_SDK.INSTANCE.getDLLVersion(0);

//UHF GEN2

GTSP.LGTSP.L_SDK.INSTANCE.writeUHF("H", 2, 12, "E", "11223344556677889900AABB");

GTSP.LGTSP.L_SDK.INSTANCE.printlabel("1", "1");

GTSP.LGTSP.L_SDK.INSTANCE.EPCPWD_Action("L", "12345678");

GTSP.LGTSP.L_SDK.INSTANCE.TIDPWD_Action("L", "12345678");

GTSP.LGTSP.L_SDK.INSTANCE.USERPWD_Action("L", "12345678");

GTSP.LGTSP.L_SDK.INSTANCE.AccessPWD_Action("U", "12345678");

GTSP.LGTSP.L_SDK.INSTANCE.KillPWD_Action("U", "12345678");

GTSP.LGTSP.L_SDK.INSTANCE.Set_RFIDPorcedure(8,8,32,"3","N",2,2);

GTSP.LGTSP.L_SDK.INSTANCE.Set_RFIDPorcedure_mm(5, 40, 32, 5, "N", 5, 5, "203");

GTSP.LGTSP.L_SDK.INSTANCE.writeHF("H", 2, 12, "414142424343444445454646");

GTSP.LGTSP.L_SDK.INSTANCE.printlabel("1", "1");

//UHF GJB 写入资料

GTSP.LGTSP.L_SDK.INSTANCE.writeGJB_UHF("H", 1, 12, "E", "414142424343444445454646",
"12345678");

*带入写入密码，写入资料(startBlockNo: GJB 默认为 1)

//UHF GJB 设定各密码区域新密码

GTSP.LGTSP.L_SDK.INSTANCE.setPWD_Action("S", "S", "11112222", "12345678");

//带入写入密码，设定新状态密码

GTSP.LGTSP.L_SDK.INSTANCE.setPWD_Action("R", "S", "33334444", "12345678");

//带入写入密码，设定新读取密码

GTSP.LGTSP.L_SDK.INSTANCE.setPWD_Action("K", "S", "55556666", "12345678");

//带入写入密码，设定新删除密码

GTSP.LGTSP.L_SDK.INSTANCE.setPWD_Action("W", "S", "87654321", "12345678");

```


//带入旧写入密码，设定新写入密码

//UHF GJB 设定资料区块状态

GTSP.LGTSP.L_SDK.INSTANCE.statusGJB_UHF("E", "B", "11112222"); //带入状态密码，设定状态

GTSP.LGTSP.L_SDK.INSTANCE.printlabel("1", "1");

//UHF GJB 删除标签

GTSP.LGTSP.L_SDK.INSTANCE.killGJB_UHF ("55556666"); //带入删除密码，删除标签

GTSP.LGTSP.L_SDK.INSTANCE.printlabel("1", "1");

GTSP.LGTSP.L_SDK.INSTANCE.closeport()

//简中打印

WString str=new WString("默认简体中文测试");//需 jni 的 WString 才能正确传送中文

GTSP.LGTSP.L_SDK.INSTANCE.clearbuffer();

GTSP.LGTSP.L_SDK.INSTANCE.printerfontUnicode("100", "10", "TSS24.BF2", "0", "1", "1", str);

GTSP.LGTSP.L_SDK.INSTANCE.printlabel(1, 1);

//繁中打印

WString str=new Wstring("默认繁体中文测试");//需 jni 的 WString 才能正确传送中文

GTSP.LGTSP.L_SDK.INSTANCE.clearbuffer();

GTSP.LGTSP.L_SDK.INSTANCE. printerfontUnicode("100", "10", " TST24.BF2", "0", "1", "1", str);

GTSP.LGTSP.L_SDK.INSTANCE.printlabel(1, 1);

GTSP.LGTSP.L_SDK.INSTANCE.closeport();

GTSP.LGTSP.L_SDK.INSTANCE.getDLLVersion(0);

//RFID 自动校准

GTSP.LGTSP.L_SDK.INSTANCE.clearbuffer();

GTSP.LGTSP.L_SDK.INSTANCE.RFIDAutoCalibration();

GTSP.LGTSP.L_SDK.INSTANCE.closeport();

//指定 USB 传输接口

//侦测多台打印机

```
String PrinterName = GTSPL.GTSPL_SDK.INSTANCE.detectUSBStr_USB();
```

```
String[] PrinterNameStringArray = PrinterName.split("\\n+");
```

```
GTSPL.GTSPL_SDK.INSTANCE.openport_USB("Printer Model");
```

//单机打印

```
GTSPL.GTSPL_SDK.INSTANCE.openport_USB();
```

```
GTSPL.GTSPL_SDK.INSTANCE.setup_USB("54", "30", "2", "3", "0", "3", "0");
```

```
GTSPL.GTSPL_SDK.INSTANCE.sendcommand_USB("DIRECTION 1");
```

```
GTSPL.GTSPL_SDK.INSTANCE.setDirectionAndMirror_USB(1, 1);
```

```
GTSPL.GTSPL_SDK.INSTANCE.setShift_USB(50);
```

```
GTSPL.GTSPL_SDK.INSTANCE.setAfterPrintAction_USB(2);
```

```
GTSPL.GTSPL_SDK.INSTANCE.setOffset_USB(20);
```

```
GTSPL.GTSPL_SDK.INSTANCE.setCutMode_USB(0,2);
```

```
GTSPL.GTSPL_SDK.INSTANCE.clearbuffer_USB();
```

```
GTSPL.GTSPL_SDK.INSTANCE.barcode_USB("30", "30", "128", "100", "1", "0", "2", "2",  
"barcode1234567");
```

```
GTSPL.GTSPL_SDK.INSTANCE.qrcode_USB("50", "100", "H", "4", "A", "0", "QRcode1234567");
```

```
GTSPL.GTSPL_SDK.INSTANCE.printerfont_USB("50", "10", "2", "0", "1", "1", "Print Font 123456");
```

```
GTSPL.GTSPL_SDK.INSTANCE.windowfont_USB(5, 150, 48, 0, 0, 0, "arial", "JAVA WIN DRIVER");
```

```
GTSPL.GTSPL_SDK.INSTANCE.downloadpcx_USB(System.getProperty("user.dir")+ "\\UL.PCX",  
"UL.PCX");
```

```
GTSPL.GTSPL_SDK.INSTANCE.sendcommand_USB("PUTPCX 50,10,\"UL.PCX\");
```

```
GTSPL.GTSPL_SDK.INSTANCE.printerfontblock_USB("15", "15", "790", "90", "0", "0", "8", "8", "20",  
"2", "We stand behind our products with one of the most comprehensive support programs in the  
Auto-ID industry.");
```

//初始化

```
GTSP.LGTSP.L_SDK.INSTANCE.genericDefault_USB();
GTSP.LGTSP.L_SDK.INSTANCE.sensorDefault_USB();
GTSP.LGTSP.L_SDK.INSTANCE.rfidSetupDefault_USB();
```

//修改 Wifi 频段

```
GTSP.LGTSP.L_SDK.INSTANCE.WifiFrequency_USB ("5G");
```

//BLOCK 打印

```
WString str = new WString("-- 默认简体中文测试：海天米醋白米醋 1 瓶 450ml --");
GTSP.LGTSP.L_SDK.INSTANCE.printerfontblockUnicode_USB("15", "15", "790", "90", "TSS24.BF2", "0",
"1", "1", "0", "1", str);
GTSP.LGTSP.L_SDK.INSTANCE.printlabel_USB("1", "1");
```

//UHF GEN2

```
String status = GTSP.LGTSP.L_SDK.INSTANCE.printerstatus_USB();
GTSP.LGTSP.L_SDK.INSTANCE.writeUHF_USB("H", 2, 12, "E", "11223344556677889900AABB");
GTSP.LGTSP.L_SDK.INSTANCE.printlabel_USB("1", "1");
String status = GTSP.LGTSP.L_SDK.INSTANCE.readUHF_USB("A", 0, 12, "E");
GTSP.LGTSP.L_SDK.INSTANCE.EPCPWD_Action_USB("L", "12345678");
GTSP.LGTSP.L_SDK.INSTANCE.TIDPWD_Action_USB("L", "12345678");
GTSP.LGTSP.L_SDK.INSTANCE.USERPWD_Action_USB("L", "12345678");
GTSP.LGTSP.L_SDK.INSTANCE.AccessPWD_Action_USB("U", "12345678");
GTSP.LGTSP.L_SDK.INSTANCE.KillPWD_Action_USB("U", "12345678");
GTSP.LGTSP.L_SDK.INSTANCE.Set_RFIDPorcedure_USB(8, 8, 32, "3", "N", 2, 2);
GTSP.LGTSP.L_SDK.INSTANCE.Set_RFIDPorcedure_mm_USB(5, 40, 32, 5, "N", 5, 5, "203");
```

```

GTSPL.GTSPL_SDK.INSTANCE.writeHF_USB("H", 2, 12, "414142424343444445454646")

GTSPL.GTSPL_SDK.INSTANCE.printlabel_USB("1", "1");

//UHF GJB 写入资料

GTSPL.GTSPL_SDK.INSTANCE.writeGJB_UHF_USB("H", 1, 12, "E", "414142424343444445454646",
"12345678");

*带入写入密码，写入资料(startBlockNo: GJB 默认为 1)

//UHF GJB 设定各密码区域新密码

GTSPL.GTSPL_SDK.INSTANCE.setPWD_Action_USB ("S", "S", "11112222", "12345678");

//带入写入密码，设定新状态密码

GTSPL.GTSPL_SDK.INSTANCE.setPWD_Action("R", "S", "33334444", "12345678");

//带入写入密码，设定新读取密码

GTSPL.GTSPL_SDK.INSTANCE.setPWD_Action("K", "S", "55556666", "12345678");

//带入写入密码，设定新删除密码

GTSPL.GTSPL_SDK.INSTANCE.setPWD_Action("W", "S", "87654321", "12345678");

//带入旧写入密码，设定新写入密码

//UHF GJB 设定资料区块状态

GTSPL.GTSPL_SDK.INSTANCE.statusGJB_UHF_USB ("E", "B", "11112222");

//带入状态密码，设定状态

GTSPL.GTSPL_SDK.INSTANCE.printlabel_USB( ("1", "1");

//UHF GJB 读取资料

String data = GTSPL.GTSPL_SDK.INSTANCE. readGJB_UHF_USB("H", 0, 12, "E", "33334444");

//带入读取密码，读取标签资料

//UHF GJB 删除标签

GTSPL.GTSPL_SDK.INSTANCE.killGJB_UHF ("55556666"); //带入删除密码，删除标签

GTSPL.GTSPL_SDK.INSTANCE.printlabel_USB( ("1", "1");

```

```

GTSP.LGTSP.L_SDK.INSTANCE.closeport_USB();

GTSP.LGTSP.L_SDK.INSTANCE.getDLLVersion_USB(1);

//简中打印
WString str=new WString("默认简体中文测试");//需 jni 的 WString 才能正确传送中文

GTSP.LGTSP.L_SDK.INSTANCE.clearbuffer_USB();

GTSP.LGTSP.L_SDK.INSTANCE. printerfontUnicode_USB ("100", "10", "TSS24.BF2", "0", "1", "1", str);

GTSP.LGTSP.L_SDK.INSTANCE.printlabel_USB (1, 1);

//繁中打印
WString str=new Wstring("默认繁体中文测试");//需 jni 的 WString 才能正确传送中文

GTSP.LGTSP.L_SDK.INSTANCE.clearbuffer_USB();

GTSP.LGTSP.L_SDK.INSTANCE. printerfontUnicode_USB ("100", "10", " TST24.BF2", "0", "1", "1", str);

GTSP.LGTSP.L_SDK.INSTANCE.printlabel_USB(1, 1);

GTSP.LGTSP.L_SDK.INSTANCE.closeport_USB();

GTSP.LGTSP.L_SDK.INSTANCE.getDLLVersion_USB(0);

//RFID 自动校准

GTSP.LGTSP.L_SDK.INSTANCE.clearbufferr_USB();

GTSP.LGTSP.L_SDK.INSTANCE.RFIDAutoCalibrationr_USB();

GTSP.LGTSP.L_SDK.INSTANCE.closeportr_USB();

//setResponse 自动响应功能
Timer receiveResponseTimer;           //接收响应指令计时器
private void set_response_usb_btnMouseClicked(java.awt.event.MouseEvent evt)
{
    GTSP.LGTSP.L_SDK.INSTANCE.setResponse_USB("Test","ON");
    GTSP.LGTSP.L_SDK.INSTANCE.sendcommand_USB("SIZE 4,2");
    GTSP.LGTSP.L_SDK.INSTANCE.sendcommand_USB("GAP 0,0");
    GTSP.LGTSP.L_SDK.INSTANCE.printlabel_USB("1","5");
    //打印标签必须使用 printlabel 才可正常接收
    ExecutorService executor = Executors.newSingleThreadExecutor();
    executor.submit(() -> {

```

```

        startReceiveResponseTimer();           //开启自动响应 Timer
    });
    executor.shutdown();
}

```

//开启自动响应 Timer

```

public void startReceiveResponseTimer() {
    receiveResponseTimer = new Timer();
    receiveResponseTimer.scheduleAtFixedRate(new TimerTask() {
        @Override
        public void run() {
            timerElapsed();
        }
    }, 800, 800);           // 延迟 800 毫秒后开始，每 800 毫秒重复执行
}

```

//Timer 的处理

```

public void timerElapsed() {
    String response = GTSPL.GTSPL_SDK.INSTANCE.setResponseReadFromPrinter_USB();
    SwingUtilities.invokeLater(() -> {
        // 在 UI 执行续执行
        set_response_usb_txt.setText(response + "\r\n");
    });

    if (GTSPL.GTSPL_SDK.INSTANCE.setResponseReceiveSTOP_USB() == 1) {
        stopReceiveResponseTimer();
    }
}

```

//关闭自动响应 Timer

```

public void stopReceiveResponseTimer() {
    if (receiveResponseTimer != null) {
        receiveResponseTimer.cancel(); // 停止计时器任务
        receiveResponseTimer.purge();  // 移除所有已取消的任务
        receiveResponseTimer = null;   // 释放资源
    }
}

```

//指定 Ethernet 传输接口

```

GTSPL.GTSPL_SDK.INSTANCE.openport_Ethernet(IP,Port);

GTSPL.GTSPL_SDK.INSTANCE.setup_Ethernet("54", "30", "2", "3", "0", "3", "0");

GTSPL.GTSPL_SDK.INSTANCE.sendcommand_Ethernet("DIRECTION 1");

GTSPL.GTSPL_SDK.INSTANCE.setOffset_Ethernet(20);

GTSPL.GTSPL_SDK.INSTANCE.setCutMode_Ethernet(0,2);

GTSPL.GTSPL_SDK.INSTANCE.setDirectionAndMirror_Ethernet(1, 1);

GTSPL.GTSPL_SDK.INSTANCE.setShift_Ethernet(50);

GTSPL.GTSPL_SDK.INSTANCE.setAfterPrintAction_Ethernet(2);

GTSPL.GTSPL_SDK.INSTANCE.clearbuffer_Ethernet();

GTSPL.GTSPL_SDK.INSTANCE.sendcommand_Ethernet("TEXT 100,100,\"3\",0,1,1,\"REVERSE\"");

GTSPL.GTSPL_SDK.INSTANCE.printReverse_Ethernet(90, 90, 128, 40);

GTSPL.GTSPL_SDK.INSTANCE.barcode_Ethernet("30", "30", "128", "100", "1", "0", "2", "2",
"barcode1234567");

GTSPL.GTSPL_SDK.INSTANCE.qrcode_Ethernet("50", "100", "H", "4", "A", "0", "QRcode1234567");

GTSPL.GTSPL_SDK.INSTANCE.printerfont_Ethernet("50", "10", "2", "0", "1", "1", "Print Font
123456");

GTSPL.GTSPL_SDK.INSTANCE.windowfont_Ethernet(5, 150, 48, 0, 0, 0, "arial", "JAVA WIN DRIVER");

GTSPL.GTSPL_SDK.INSTANCE.downloadpcx_Ethernet(System.getProperty("user.dir")+ "\\UL.PCX",
"UL.PCX");

    GTSPL.GTSPL_SDK.INSTANCE.sendcommand_Ethernet("PUTPCX 50,10,\"UL.PCX\"");

GTSPL.GTSPL_SDK.INSTANCE.printerfontblock_Ethernet("15", "15", "790", "90", "0", "0", "8", "8",
"20", "2", "We stand behind our products with one of the most comprehensive support programs in
the Auto-ID industry.");

```

//初始化

```
GTSP.LGTSP.L_SDK.INSTANCE.genericDefault_Ethernet();
GTSP.LGTSP.L_SDK.INSTANCE.sensorDefault_Ethernet();
GTSP.LGTSP.L_SDK.INSTANCE.rfidSetupDefault_Ethernet();
```

//修改 Wifi 频段

```
GTSP.LGTSP.L_SDK.INSTANCE.WifiFrequency("5G");
```

//BLOCK 打印

```
WString str = new WString("-- 默认简体中文测试：海天米醋白米醋 1 瓶 450ml --");
GTSP.LGTSP.L_SDK.INSTANCE.printerfontblockUnicode_Ethernet("15", "15", "790", "90", "TSS24.BF2",
"0", "1", "1", "0", "1", str);
GTSP.LGTSP.L_SDK.INSTANCE.printlabel_Ethernet("1", "1");
String status = GTSP.LGTSP.L_SDK.INSTANCE.printerstatus_Ethernet();
```

//UHF GEN2

```
GTSP.LGTSP.L_SDK.INSTANCE.writeUHF_Ethernet("H", 0, 12, "E", "11223344556677889900AABB");
GTSP.LGTSP.L_SDK.INSTANCE.printlabel_Ethernet("1", "1");
String status = GTSP.LGTSP.L_SDK.INSTANCE.readUHF_Ethernet("A", 0, 12, "E");
GTSP.LGTSP.L_SDK.INSTANCE.EPCPWD_Action_Ethernet("L", "12345678");
GTSP.LGTSP.L_SDK.INSTANCE.TIDPWD_Action_Ethernet("L", "12345678");
GTSP.LGTSP.L_SDK.INSTANCE.USERPWD_Action_Ethernet("L", "12345678");
GTSP.LGTSP.L_SDK.INSTANCE.AccessPWD_Action_Ethernet("U", "12345678");
GTSP.LGTSP.L_SDK.INSTANCE.KillPWD_Action_Ethernet("U", "12345678");
GTSP.LGTSP.L_SDK.INSTANCE.Set_RFIDPorcedure_Ethernet(8, 8, 32, "3", "N", 2, 2);
GTSP.LGTSP.L_SDK.INSTANCE.Set_RFIDPorcedure_mm_Ethernet(5, 40, 32, 5, "N", 5, 5, "203");
```



```
GTSP.LGTSP.L_SDK.INSTANCE.writeHF_Ethernet("H", 0, 12, "414142424343444445454646")
```

```
GTSP.LGTSP.L_SDK.INSTANCE.printlabel_Ethernet("1", "1");
```

```
//UHF GJB 写入资料
```

```
GTSP.LGTSP.L_SDK.INSTANCE.writeGJB_UHF_Ethernet("H", 1, 12, "E",
```

```
"414142424343444445454646", "12345678");
```

*帶入寫入密碼，寫入資料(startBlockNo: GJB 預設為 1)

```
// UHF GJB 设定各密码区域新密码
```

```
GTSP.LGTSP.L_SDK.INSTANCE.setPWD_Action_Ethernet("S", "S", "11112222", "12345678");
```

//帶入写入密码，设定新状态密码

```
GTSP.LGTSP.L_SDK.INSTANCE.setPWD_Action_Ethernet("R", "S", "33334444", "12345678");
```

//帶入写入密码，设定新读取密码

```
GTSP.LGTSP.L_SDK.INSTANCE.setPWD_Action_Ethernet("K", "S", "55556666", "12345678");
```

//帶入写入密码，设定新删除密码

```
GTSP.LGTSP.L_SDK.INSTANCE.setPWD_Action_Ethernet("W", "S", "87654321", "12345678");
```

//帶入旧写入密码，设定新写入密码

```
// UHF GJB 设定资料区块状态
```

```
GTSP.LGTSP.L_SDK.INSTANCE.statusGJB_UHF_Ethernet("E", "B", "11112222");
```

//帶入状态密码，设定状态

```
GTSP.LGTSP.L_SDK.INSTANCE.printlabel_Ethernet( ("1", "1");
```

// UHF GJB 读取资料

```
String data = GTSP.LGTSP.L_SDK.INSTANCE. readGJB_UHF_Ethernet("H", 0, 12, "E", "33334444");
```

//帶入读取密码，读取标签资料

// UHF GJB 删除标签

```
GTSP.LGTSP.L_SDK.INSTANCE.killGJB_UHF_Ethernet("55556666"); //帶入删除密码，删除標籤
```

```
GTSP.LGTSP.L_SDK.INSTANCE.printlabel_Ethernet( "1", "1");
```

```
GTSP.LGTSP.L_SDK.INSTANCE.closeport_Ethernet();
```

```
GTSP.LGTSP.L_SDK.INSTANCE.getDLLVersion_Ethernet(1);
```

```
//簡中打印
```

```
WString str=new WString("默认简体中文测试"); //需 jni 的 WString 才能正確傳送中文
```

```
GTSP.LGTSP.L_SDK.INSTANCE.clearbuffer_Ethernet();
```

```
GTSP.LGTSP.L_SDK.INSTANCE.printerfontUnicode_Ethernet("100", "10", "TSS24.BF2", "0", "1", "1",  
str);
```

```
GTSP.LGTSP.L_SDK.INSTANCE.printlabel_Ethernet(1, 1);
```

```
//繁中打印
```

```
WString str=new Wstring("默認繁體中文測試");//需 jni 的 WString 才能正確傳送中文
```

```
GTSP.LGTSP.L_SDK.INSTANCE.clearbuffer_Ethernet();
```

```
GTSP.LGTSP.L_SDK.INSTANCE. printerfontUnicode_Ethernet("100", "10", " TST24.BF2", "0", "1", "1",  
str);
```

```
GTSP.LGTSP.L_SDK.INSTANCE.printlabel_Ethernet(1, 1);
```

```
GTSP.LGTSP.L_SDK.INSTANCE.closeport_Ethernet();
```

```
GTSP.LGTSP.L_SDK.INSTANCE.getDLLVersion_Ethernet(0);
```

```
//RFID 自动校准
```

```
GTSP.LGTSP.L_SDK.INSTANCE.clearbufferr_Ethernet();
```

```
GTSP.LGTSP.L_SDK.INSTANCE.RFIDAutoCalibrationr_Ethernet();
```

```
GTSP.LGTSP.L_SDK.INSTANCE.closeport_Ethernet();
```

4.DLL 放置位置:

〔 利用 IDE 开发时 〕

GTSP.L_SDK.dll 放在 jdk 的 bin 文件夹下

如: C:\Program Files\Java\jdk1.8.0_221\bin

GTSP_L_SDK_C.dll 放在 java 项目文件夹下

[未安装 jdk 直接执行.jar 档时]

GTSP_L_SDK.dll 放在 jre 的 bin 文件夹下

如: C:\Program Files\Java\jre1.8.0_251\bin

GTSP_L_SDK_C.dll 放在.jar 档相同路径下

.jar 档相同路径下, 还需要有 lib 文件夹, 且里面要有 jna-5.5.0.jar

5.Driver 模式下, 需要安装打印机的驱动才能执行。

● Javascript

1.使用管理员身分开启命令提示字符

2.注册 dll

将 x86 的 dll 放到 Windows/Syswow64, x64 的 dll 放在 Windows/System32

进入 microsoft.net framework 的安装路径下

如: cd C:\Windows\Microsoft.NET\Framework64\v4.0.30319

*x64 的 dll 注册在 C:\Windows\Microsoft.NET\Framework64\v4.0.30319

*x86 的 dll 注册在 C:\Windows\Microsoft.NET\Framework\v4.0.30319

输入指令 regasm.exe /register /tlb <dll path>

输入指令 regasm.exe <dll path> /codebase

如: regasm.exe /register /tlb C:\Users\User\Desktop\Javascript_Sample_Code\GTSP_L_SDK.dll

regasm.exe C:\Users\User\source\repos\product\GTSP_L_SDK \GTSP_L_SDK.dll /codebase

3.范例程序:

```
<script language="javascript" type="text/javascript">
```

```
//单机打印
```

```
var driverObject = new ActiveXObject("GTSP_L_SDK.Driver");
```

```

driverObject.openport("Printer Model");

driverObject.setup("54", "30", "2", "3", "0", "3", "0");

driverObject.sendcommand("DIRECTION 1");

driverObject.clearbuffer();

driverObject.barcode("30", "30", "128", "100", "1", "0", "2", "2", "barcode1234567");

driverObject.printerfont("50", "10", "3", "0", "1", "1", "Print Font 123456");

var file = getFilePath();

driverObject.downloadpcx(file + "\\UL.PCX", "UL.PCX");

driverObject.sendcommand("PUTPCX 50,10,\"UL.PCX\"");

driverObject.windowfont(50, 20, 48, 0, 0, 0, "arial", "Windows Font Test123467");

driverObject.getDLLVersion(1)

driverObject.printlabel("1", "1");

driverObject.closeport();

```

//指定 USB 传输接口

```

var usbObject = new ActiveXObject("GTSPL_SDK.USB");

usbObject.openport_USB();

usbObject.setup_USB("54", "30", "2", "3", "0", "3", "0");

usbObject.sendcommand_USB("DIRECTION 1");

usbObject.clearbuffer_USB();

usbObject.barcode_USB("30", "30", "128", "100", "1", "0", "2", "2", "barcode1234567");

usbObject.printerfont_USB("50", "10", "3", "0", "1", "1", "Print Font 123456");

var file = getFilePath();

usbObject.downloadbmp_USB(file + "\\CIRCLE.BMP", "CIRCLE.BMP");

usbObject.sendcommand_USB("PUTBMP 150,30,\"CIRCLE.BMP\"");

usbObject.windowfont_USB(10, 100, 48, 0, 0, 0, "arial", "Windows Font Test");

```

```
var status = usbObject.printerstatus_USB();
```

```
usbObject.printlabel_USB("1", "1");
```

```
//简中打印
```

```
string stString="默认简体中文测试";
```

```
usbObject.clearbuffer_USB();
```

```
usbObject.printerfont_USB("100", "10", "TSS24.BF2", "0", "1", "1", stString);
```

```
usbObject.printlabel_USB(1, 1);
```

```
//繁中打印
```

```
string ttString="默认繁体中文测试";
```

```
usbObject.clearbuffer_USB();
```

```
usbObject.printerfont_USB ("100", "10", " TST24.BF2", "0", "1", "1", ttString);
```

```
usbObject.printlabel_USB (1, 1);
```

```
usbObject.getDLLVersion_USB(1);
```

```
usbObject.closeport_USB();
```

```
usbObject.getDLLVersion_USB(0)
```

```
</script>
```

● Asp.Net

1.加入 x86 的 dll 进项目

2.范例程序:

```
//单机打印
```

```
GTSPK_SDK.Driver driver = new GTSPK_SDK.Driver();
```

```
driver.openport("Printer Model");
```

```
driver.setup("54", "30", "2", "3", "0", "3", "0");
```

```

driver.sendcommand("DIRECTION 1");

driver.clearbuffer();

driver.barcode("30", "30", "128", "100", "1", "0", "2", "2", "barcode1234567");

string path = HttpContext.Current.Server.MapPath("~/");

driver.downloadpcx(path + "\\CIRCLE.BMP", "CIRCLE.BMP");

driver.sendcommand("PUTBMP 150,30,\"CIRCLE.BMP\"");

driver.windowfont(50, 20, 48, 0, 0, 0, "impact", "Windows Font ASP.NET");

driver.getDLLVersion(1);

driver.printlabel("1", "1");

driver.closeport();

```

//指定 USB 传输接口

```

GTSPL_SDK.USB usb = new GTSPL_SDK.USB();

usb.openport_USB();

usb.setup_USB("54", "30", "2", "3", "0", "3", "0");

usb.sendcommand_USB("DIRECTION 1");

usb.clearbuffer_USB();

usb.barcode_USB("30", "30", "128", "100", "1", "0", "2", "2", "barcode1234567usb");

usb.printerfont_USB("50", "10", "3", "0", "1", "1", "Print Font usb Asp");

usb.downloadpcx_USB(path + "\\UL.PCX", "UL.PCX");

usb.sendcommand_USB("PUTPCX 50,10,\"UL.PCX\"");

usb.windowfont_USB(10, 100, 48, 0, 0, 0, "impact", "Windows Font ASP USB");

usb.printlabel_USB("1", "1");

var status = usb.printerstatus_USB();

usb.getDLLVersion_USB(1);

```

//简中打印

```
string stString="默认简体中文测试";

usb.clearbuffer_USB();

usb.printerfont_USB("100", "10", "TSS24.BF2", "0", "1", "1", stString);

usb.printlabel_USB(1, 1);
```

//繁中打印

```
string ttString="默認繁體中文測試";

usb.clearbuffer_USB();

usb.printerfont_USB ("100", "10", " TST24.BF2", "0", "1", "1", ttString);

usb.printlabel_USB (1, 1);

usb.closeport_USB();
```

● VB.Net

1.加入 dll 进项目

2.范例程序:

```
Dim path = Environment.CurrentDirectory

//单机打印

Dim driver As New GTSP_SDK.Driver

driver.openport("Printer Model")

driver.setup("54", "30", "2", "3", "0", "3", "0")

driver.sendcommand("DIRECTION 1")

driver.clearbuffer()

driver.barcode("30", "30", "128", "100", "1", "0", "2", "2", "barcode1234567")

driver.printerfont("50", "10", "3", "0", "1", "1", "PrintFont in vb.net")
```

```

driver.downloadbmp(path + "\\CIRCLE.BMP", "CIRCLE.BMP")

driver.sendcommand("PUTBMP 50,10, ""CIRCLE.BMP""")

driver.windowfont(50, 20, 48, 0, 0, 0, "impact", "VB Imapct Driver")

driver.printlabel("1", "1")

driver.closeport()

```

//指定 USB 传输接口

```

Dim usb As New GTSPL_SDK.USB

usb.openport_USB()

usb.setup_USB("54", "30", "2", "3", "0", "3", "0")

usb.sendcommand_USB("DIRECTION 1")

usb.clearbuffer_USB()

usb.barcode_USB("30", "30", "128", "100", "1", "0", "2", "2", "barcode1234567")

usb.downloadbmp_USB(path + "\\CIRCLE.BMP", "CIRCLE.BMP")

usb.sendcommand_USB("PUTBMP 50,10, ""CIRCLE.BMP""")

usb.windowfont_USB(50, 20, 48, 0, 0, 0, "impact", "VB Imapct Driver")

Dim status = usb.printerstatus_USB()

Dim vcode = usb.getDLLVersion_USB("1")

usb.printlabel_USB("1", "1")

usb.closeport_USB()

```

//简中打印

```

usb.clearbuffer_USB();

usb.printerfont_USB("100", "10", "TSS24.BF2", "0", "1", "1", "默认简体中文测试");

usb.printlabel_USB(1, 1);

```


//繁中打印

```
usb.clearbuffer_USB();

usb.printerfont_USB ("100", "10", " TST24.BF2", "0", "1", "1", "默認繁體中文測試");

usb.printlabel_USB (1, 1);

usb.closeport_USB();
```

● Access VBA

1.使用管理员身分开启命令提示字符

2.注册 dll

将 x86 的 dll 放到 Windows/Syswow64

进入 microsoft.net framework 的安装路径下

如: cd C:\Windows\[Microsoft.NET](#)\Framework\v4.0.30319

输入指令 regasm.exe /register /tlb <dll path>

输入指令 regasm.exe <dll path> /codebase

如: regasm.exe /register /tlb C:\Users\User\Desktop\Javascript_Sample_Code\ GTSPL_SDK.dll

regasm.exe C:\Users\User\source\repos\product\GTSPL_SDK \GTSPL_SDK.dll /codebase

3.范例程序:

```
Dim usb As New GTSPL_SDK.usb

Dim path As String

usb.openport_USB()

Call usb.clearbuffer_USB

Call usb.setup_USB("54", "39", "4", "7", "0", "3", "0")

Call usb.clearbuffer_USB
```

Call `usb.printerfont_USB("10", "20", "2", "0", "1", "1", "Product Management")`

Call `usb.printerfont_USB("10", "40", "2", "0", "1", "1", "Product No. : ")`

Call `usb.barcode_USB("10", "60", "128", "50", "1", "0", "1", "1", ProductNo)`

Call `usb.printerfont_USB("10", "140", "2", "0", "1", "1", "Product Name : " + ProductName)`

Call `usb.printerfont_USB("10", "160", "2", "0", "1", "1", "Unit Price : " + UnitPrice)`

Call `usb.printerfont_USB("10", "180", "2", "0", "1", "1", "Amount : " + Amount)`

Call `usb.printerfont_USB("10", "200", "2", "0", "1", "1", "Total Price : " + TotalPrice)`

Call `usb.sendcommand_USB("PUTBMP 10,230, ""LOGO.BMP""")`

Call `usb.printlabel_USB("1", "1")`

Call `usb.closeport_USB`

● Excel VBA

1.使用管理员身分开启命令提示字符

2.注册 dll

将 x86 的 dll 放到 Windows/Syswow64

进入 microsoft.net framework 的安装路径下

如: cd C:\Windows\Microsoft.NET\Framework\v4.0.30319

输入指令 regasm.exe /register /tlb <dll path>

输入指令 regasm.exe <dll path> /codebase

如: regasm.exe /register /tlb C:\Users\User\Desktop\Javascript_Sample_Code\ GTSPL_SDK.dll

regasm.exe C:\Users\User\source\repos\product\GTSPL_SDK \GTSPL_SDK.dll /codebase

3.范例程序:

```
Dim driver As New GTSPL_SDK.driver
```

```
Dim path As String
```

```
openport = driver.openport("Printer Model")
```

```
Call driver.setup("54", "20", "2", "3", "0", "3", "0")
```

```
Call driver.sendcommand("DIRECTION 1")
```

```
Call driver.clearbuffer
```

```
Call driver.barcode("30", "30", "128", "100", "1", "0", "2", "2", "barcode1234567")
```

```
Call driver.printerfont("50", "10", "3", "0", "1", "1", "PrintFont in vba_xls")
```

```
Call driver.downloadpcx(path + "\\UL.PCX", "UL.PCX")
```

```
Call driver.sendcommand("PUTPCX 50,10,\"\"UL.PCX\"\"")
```

```
Call driver.windowfont(50, 20, 48, 0, 0, 0, "impact", "VBAEX Driver")
```

```
Call driver.printlabel("1", "1")
```

```
Call driver.closeport
```

● Visual C++

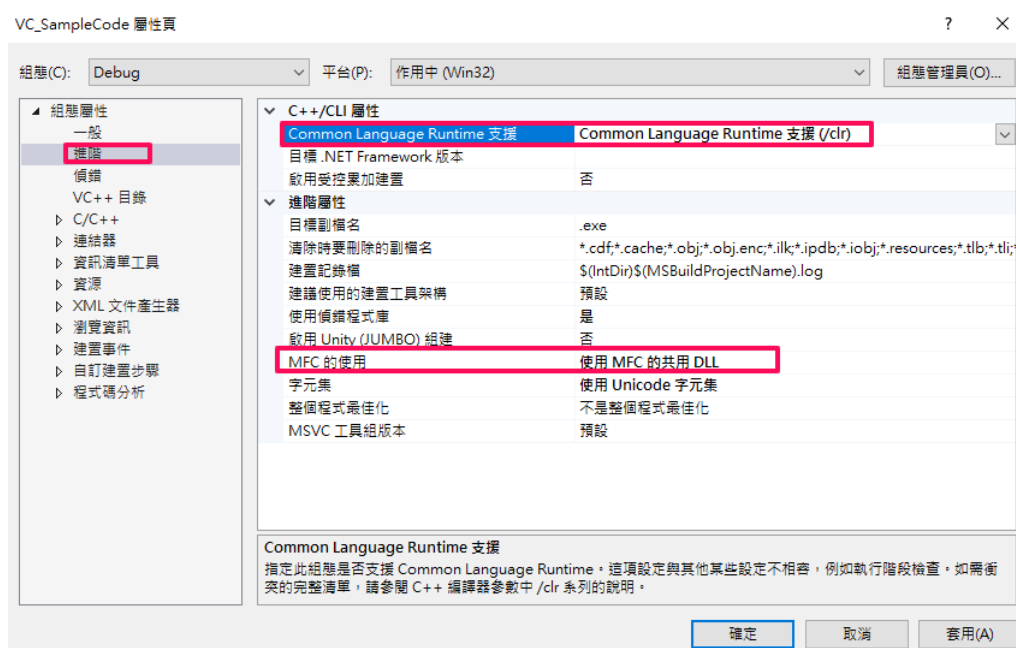
1. 设定项目属性

要支持 dll 必须将 Common Language Runtime 支持启动

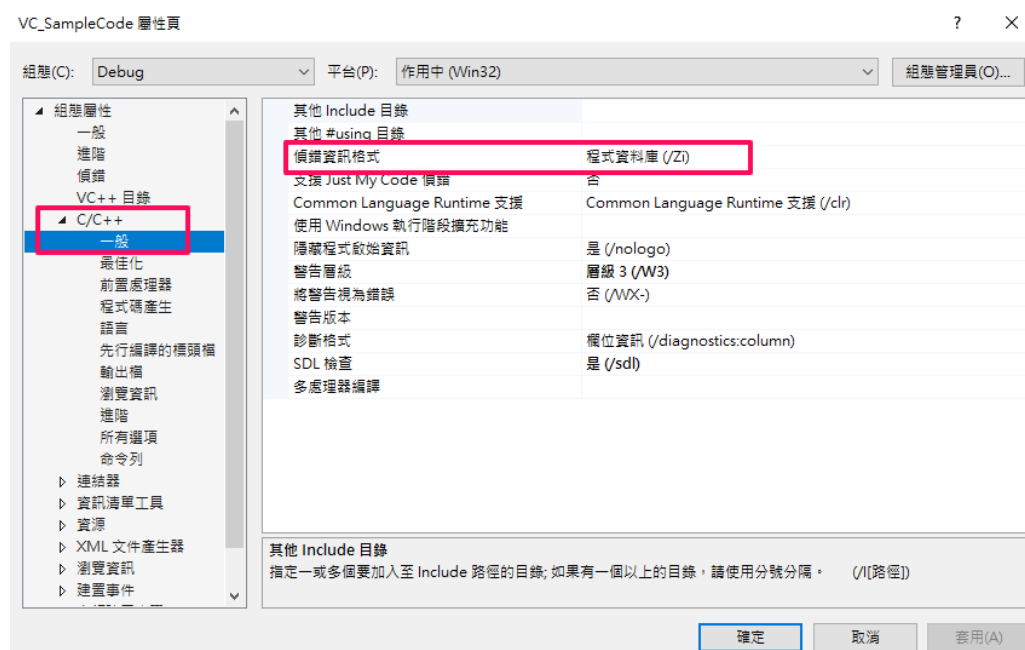
=>开启项目属性

=>组态属性=>进阶=>Common Language Runtime 支持设定为(/clr)

=>组态属性=>进阶=>MFC 的使用设定为使用 MFC 的共享 DLL

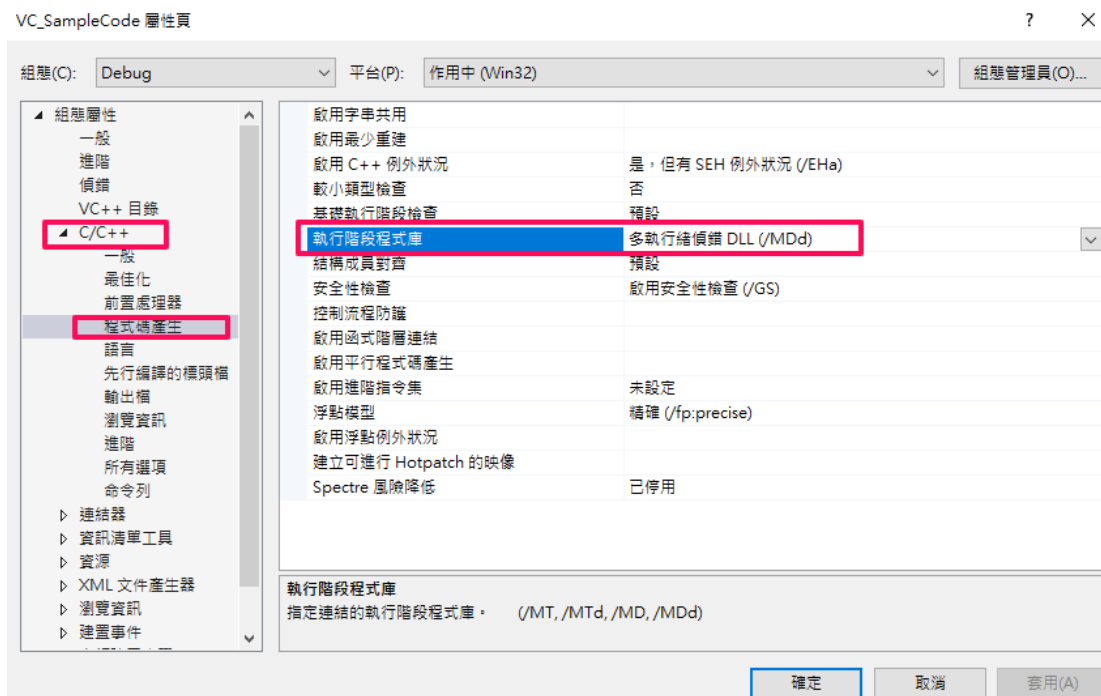


属性=>C/C++=>一般=>侦错信息格式设定为"程序数据库/zi"(不可以设定为/zi)



属性=>C/C++=>程序代码产生

=>运行时间链接库设定为"多执行续侦错 DLL(/MDd)"或"多执行续(/MD)"



3.范例程序:

```
#using "GTSPL_SDK.dll"

using namespace System;

using namespace GTSPL_SDK;

//单机打印

Driver^ driver = gcnew Driver();

String^ printerName = gcnew String(str);

driver->openport(printerName);

driver->setup("30", "30", "2", "3", "0", "3", "0");

driver->clearbuffer();

driver->barcode("30", "30", "128", "100", "1", "0", "2", "2", "barcode VC5678 ");

driver->printerfont("10", "150", "2", "0", "1", "1", "Print Font VC DRIVER");

driver->downloadpcx(_T("../VC_SampleCode/UL.PCX"), "UL.PCX");

driver->sendcommand("PUTPCX 50,10,\"UL.PCX\"");
```

```

driver ->downloadbmp (_T("../VC_SampleCode/CIRCLE.BMP"), "CIRCLE.BMP");

driver ->sendcommand ("PUTBMP 150,30,\"CIRCLE.BMP\"");

driver->windowsfont(10, 120, 48, 0, 0, 0, "impact", "VC Driver test");

driver->printlabel("1", "1");

driver->sendcommand("DIRECTION 1");

driver->qrcode("220", "260", "M", "3", "A", "0", "AABCB03abcN123");

driver->printerfontblock ("35","15","790","90","1","0","8","8","0","1", ttString);

driver ->setDirectionAndMirror(1,1);

driver ->setShift(50);

driver ->setAfterPrintAction(2);

driver ->setOffset(20);

driver ->setCutMode(0,2);

driver ->printReverse (90,90,128,40);

driver->printlabel("1", "1");

driver->sendcommand("DIRECTION 1");

driver->genericDefault();

driver->sensorDefault();

int BitmapResult = driver->Bitmap ("0", "0", 640, 484, 1, "octopus.png");

BitmapResult = driver->compressBitmap ("0", "0", 640, 484, 1, "octopus.jpg");

driver->printlabel ("1", "1");

// RFID

driver-> rfidSetupDefault();

driver->RFIDAutoCalibration();

//UHF GEN2

driver->writeUHF("H", 2, 12, "E", "414142424343444445454646"); // startBlockNo: Gen2 預設
為 2

```

```

driver->printlabel("1", "1");

driver->EPCPWD_Action("U", "12345678");

driver->TIDPWD_Action("L", "12345678");

driver->USERPWD_Action("L", "12345678");

driver->AccessPWD_Action("S", "12345678");

driver->KillPWD_Action("L", "12345678");

driver->Set_RFIDPorcedure(8, 8, 32, 3, "N", 2, 2);

driver->Set_RFIDPorcedure_mm(5, 40, 32, 5, "N", 5, 5, "203");

driver->writeHF("H", 0, 12, "414142424343444445454646");

driver->printlabel("1", "1");

//UHF GJB 写入资料

driver->writeGJB_UHF("H", 1, 12, "E", "414142424343444445454646", "12345678");

*帶入寫入密碼，寫入資料(startBlockNo: GJB 預設為 1)

//UHF GJB 设定各密码区域新密码

driver->setPWD_Action("S", "S", "11112222", "12345678"); //帶入写入密码，设定新状态密码

driver->setPWD_Action("R", "S", "33334444", "12345678"); //帶入写入密码，设定新读取密码

driver->setPWD_Action("K", "S", "55556666", "12345678"); //帶入写入密码，设定新删除密码

//帶入旧写入密码，设定新写入密码

driver->setPWD_Action("W", "S", "87654321", "12345678");

//UHF GJB 设定资料区块状态

driver->statusGJB_UHF("E", "B", "11112222"); //帶入状态密码，设定状态

driver->printlabel("1", "1");

//UHF GJB 删除标签

driver->killGJB_UHF("55556666"); //帶入删除密码，删除标签

driver->printlabel("1", "1");

driver->closeport();

```

//指定 USB 传输接口

```

USB^ usb = gcnew USB();

usb->openport_USB();

usb->setup_USB("54", "30", "2", "3", "0", "3", "0");

usb->sendcommand_USB("DIRECTION 1");

usb->clearbuffer_USB();

usb->barcode_USB("10", "80", "128", "100", "1", "0", "2", "2", "barcode vc");

usb->printerfont_USB("50", "100", "2", "0", "1", "1", "Print Font VC123");

usb->downloadbmp_USB(_T("../VC_SampleCode/CIRCLE.BMP"), "CIRCLE.BMP");

usb->sendcommand_USB("PUTBMP 150,30,\"CIRCLE.BMP\"");

usb ->downloadpcx_ USB (_T("../VC_SampleCode/UL.PCX "), "UL.PCX");

usb ->sendcommand_ USB ("PUTPCX 50,10,\"UL.PCX\"");

usb->windowsfont_USB(10, 20, 48, 0, 0, 0, "impact", "VC WIN TEST");

usb ->qrcode_USB("220", "260", "M", "3", "A", "0", "AABCB03abcN123");

usb ->printerfontblock_USB("35", "15", "790", "90", "1", "0", "8", "8", "0", "1", ttString);

usb->getDLLVersion_USB(1);

CString status;

status=usb->printerstatus_USB();

usb->printlabel_USB("1", "1");

```

//简中打印

```

usb->clearbuffer_USB();

usb->printerfont_USB("10", "110", "TSS24.BF2", "0", "1", "1", L"默认简体中文测试: 海天米醋白
米醋 1 瓶 450ml");

usb->printlabel_USB("1", "1");

```


//繁中打印

```

usb->clearbuffer_USB();

usb->printerfont_USB("10", "110", "TST24.BF2", "0", "1", "1", L"默認繁體中文測試: 海天米醋白
米醋 1 瓶 450ml");

usb ->setDirectionAndMirror_USB(1,1);

usb ->setShift_USB(50);

usb ->setAfterPrintAction_USB(2);

usb ->setOffset_USB(20);

usb ->setCutMode_USB(0,2);

usb ->printReverse_USB(90,90,128,40);

usb ->printlabel_USB("1", "1");

usb ->sendcommand_USB("DIRECTION 1");

usb ->genericDefault_USB();

usb ->sensorDefault_USB();

int BitmapResult = usb ->Bitmap_USB ("0", "0", 640, 484, 1, "octopus.png");

BitmapResult = usb ->compressBitmap_USB ("0", "0", 640, 484, 1, "octopus.jpg");

usb ->printlabel_USB ("1", "1");

// RFID

usb -> rfidSetupDefault_USB();

usb ->RFIDAutoCalibration_USB();

//UHF GEN2

usb ->writeUHF_USB("H", 2, 12, "E", "4141424243434444445454646"); // startBlockNo: Gen2
預設為 2

usb ->printlabel_USB("1", "1");

string data = usb->readUHF_USB("H", 0, 12, "E");

usb->EPCPWD_Action_USB("L", "12345678");

```

```

usb->TIDPWD_Action_USB("L", "12345678");

usb->USERPWD_Action_USB("L", "12345678");

usb->AccessPWD_Action_USB("U", "12345678");

usb->KillPWD_Action_USB("U", "12345678");

usb->Set_RFIDPorcedure_USB(8,8,32,3,"N",2,2);

usb->Set_RFIDPorcedure_mm_USB(5, 40, 32, 5, "N", 5, 5, "203");

usb->writeHF_USB("H", 0, 12, "414142424343444445454646");

usb->printlabel_USB("1", "1");

//UHF GJB 写入资料

usb ->writeGJB_UHF_USB("H", 1, 12, "E", "414142424343444445454646", "12345678");

*帶入寫入密碼，寫入資料(startBlockNo: GJB 預設為 1)

//UHF GJB 设定各密码区域新密码

//帶入写入密码，设定新状态密码

usb ->setPWD_Action_USB ("S", "S", "11112222", "12345678");

//帶入写入密码，设定新读取密码

usb ->setPWD_Action_USB ("R", "S", "33334444", "12345678");

//帶入写入密码，设定新删除密码

usb ->setPWD_Action_USB ("K", "S", "55556666", "12345678");

//帶入旧写入密码，设定新写入密码*

usb ->setPWD_Action_USB ("W", "S", "87654321", "12345678");

//UHF GJB 设定资料区块状态

usb ->statusGJB_UHF_USB ("E", "B", "11112222"); //帶入状态密码，设定状态

usb ->printlabel_USB ("1", "1");

//UHF GJB 读取资料

//帶入读取密码，读取标签资料

string data =usb->readGJB_UHF_USB("H", 0, 12, "E", "33334444");

```

//UHF GJB 删除标签

```
usb ->killGJB_UHF_USB ("55556666"); //帶入删除密码，删除标签
```

```
usb ->printlabel_USB ("1", "1");
```

```
usb ->closeport_USB ();
```

```
usb->getDLLVersion_USB(1);
```

//指定网口传输介面

```
SocketConnect socketconnect = gcnew SocketConnect();
```

```
socketconnect ->openport_Ethernet("xxx.xxx.xx.xxx", 9100);
```

```
socketconnect->setup_Ethernet ("54", "30", "2", "3", "0", "3", "0");
```

```
socketconnect->sendcommand_Ethernet ("DIRECTION 1");
```

```
socketconnect->clearbuffer_Ethernet ();
```

```
socketconnect->barcode_Ethernet ("30", "30", "128", "100", "1", "0", "2", "2",
```

```
"barcode1234567");
```

```
socketconnect->qrcode_Ethernet ("220", "260", "M", "3", "A", "0", "AABCB03abcN123");
```

```
socketconnect ->downloadpcx_Ethernet (_T("../VC_SampleCode/UL.PCX "), "UL.PCX");
```

```
socketconnect ->sendcommand_Ethernet ("PUTPCX 50,10,\"UL.PCX\");
```

```
socketconnect ->downloadbmp_Ethernet (_T("../VC_SampleCode/ CIRCLE.BMP "),
```

```
"CIRCLE.BMP");
```

```
socketconnect ->sendcommand_Ethernet ("PUTBMP 150,30,\"CIRCLE.BMP\");
```

```
socketconnect->windowsfont_Ethernet (10, 100, 48, 0, 0, 0, "arial", "VC WIN TEST");
```

```
socketconnect ->qrcode_Ethernet("220", "260", "M", "3", "A", "0", "AABCB03abcN123");
```

```
socketconnect ->printerfontblock_Ethernet ("35", "15", "790", "90", "1", "0", "8", "8", "0", "1",
```

```
ttString);
```

```
socketconnect->printlabel_Ethernet ("1", "1");
```

```
string status = socketconnect->printerstatus_Ethernet ();
```

//简中打印

```
string stString="默认简体中文测试";
```

```
socketconnect->clearbuffer_Ethernet ();
```

```
socketconnect->printerfont_Ethernet ("100", "10", "TSS24.BF2", "0", "1", "1", stString);
```

```
socketconnect->printlabel_Ethernet (1, 1, this);
```

```
//繁中打印
```

```
string ttString="默认繁体中文测试";
```

```
socketconnect->clearbuffer_Ethernet ();
```

```
socketconnect->printerfont_Ethernet ("100", "10", "TST24.BF2", "0", "1", "1", ttString);
```

```
socketconnect->printlabel_Ethernet (1, 1, this);
```

```
//BLOCK 打印
```

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

```
socketconnect->clearbuffer_Ethernet ();
```

```
socketconnect->printerfontblock_Ethernet ("35","15","790","90","1","0","8","8","0","1",  
ttString);
```

```
socketconnect ->setShift_Ethernet (50);
```

```
socketconnect ->setAfterPrintAction_Ethernet (2);
```

```
socketconnect ->setOffset_Ethernet (20);
```

```
socketconnect ->setCutMode_Ethernet (0,2);
```

```
socketconnect ->printReverse_Ethernet (90,90,128,40);
```

```
socketconnect ->printlabel_Ethernet ("1", "1");
```

```
socketconnect ->sendcommand_Ethernet ("DIRECTION 1");
```

```
socketconnect ->genericDefault_Ethernet ();
```

```
socketconnect ->sensorDefault_Ethernet ();
```

```
int BitmapResult = socketconnect ->Bitmap_Ethernet ("0", "0", 640, 484, 1, "octopus.png");
```

```
BitmapResult = socketconnect ->compressBitmap_Ethernet ("0", "0", 640, 484, 1,
```

```

"octopus.jpg");

socketconnect->printlabel_Ethernet ("1", "1");

// RFID

socketconnect->rfidSetupDefault_Ethernet ();

socketconnect->RFIDAutoCalibration_Ethernet ();

//UHF GEN2

socketconnect->writeUHF_Ethernet("H", 2, 12,"E","414142424343444445454646");

//startBlockNo : Gen2 预设为 2

socketconnect->printlabel_Ethernet ("1", "1");

string data = socketconnect->readUHF_Ethernet ("H", 0, 12, "E");

string data_Q = socketconnect->query_UHF_Ethernet ( "A", 0, 0); //以 Q 指令读取 EPC 资料

socketconnect->EPCPWD_Action_Ethernet ("L","12345678");

socketconnect->TIDPWD_Action_Ethernet ("L", "12345678");

socketconnect->USERPWD_Action_Ethernet ("L", "12345678");

socketconnect->AccessPWD_Action_Ethernet ("U", "12345678");

socketconnect->KillPWD_Action_Ethernet ("U", "12345678");

socketconnect->Set_RFIDPorcedure_Ethernet (8,8,32,3,"N",2,2);

socketconnect->Set_RFIDPorcedure_mm_Ethernet(5, 40, 32, 5, "N", 5, 5, "203");

socketconnect->writeHF_Ethernet ("H", 0, 12, "414142424343444445454646");

socketconnect->printlabel_Ethernet ("1", "1");

//UHF GJB 写入资料

socketconnect->writeGJB_UHF_Ethernet ("H", 1, 12, "E", "414142424343444445454646",

"12345678");

*带入写入密码， 写入资料(startBlockNo: GJB 预设为 1)

//UHF GJB 设定各密码区域新密码

socketconnect->setPWD_Action_Ethernet("S", "S", "11112222", "12345678"); //带入写入密码，

```

设定新状态密码

```
socketconnect->setPWD_Action_Ethernet ("R", "S", "33334444", "12345678");//带入写入密码,
```

设定新读取密码

```
socketconnect->setPWD_Action_Ethernet ("K", "S", "55556666", "12345678"); //带入写入
```

密码, 设定新删除密码

```
socketconnect->setPWD_Action_Ethernet("W", "S", "87654321", "12345678"); //带入旧写入
```

密码, 设定新写入密码

```
//UHF GJB 设定资料区块状态
```

```
socketconnect->statusGJB_UHF_Ethernet ("E", "B", "11112222"); //带入状态密码, 设定状态
```

```
socketconnect->printlabel_Ethernet ("1", "1");
```

```
//UHF GJB 读取资料
```

```
string data = socketconnect->readGJB_UHF_Ethernet ("H", 0, 12, "E", "33334444"); //带入
```

读取密码, 读取标签资料

```
//UHF GJB 删除标签
```

```
socketconnect->killGJB_UHF_Ethernet ("55556666"); //带入删除密码, 删除标签
```

```
socketconnect->printlabel_Ethernet ("1", "1");
```

```
socketconnect->closeport_Ethernet ();
```

```
socketconnect->getDLLVersion_Ethernet (1);
```

1.使用管理员身分开启命令提示字符

2.注册 dll

将 x86 的 dll 放到 Windows/Syswow64, x64 的 dll 放在 Windows/System32

进入 microsoft.net framework 的安装路径下

如: cd C:\Windows\Microsoft.NET\Framework64\v4.0.30319

*x64 的 dll 注册在 C:\Windows\Microsoft.NET\Framework64\v4.0.30319

*x86 的 dll 注册在 C:\Windows\Microsoft.NET\Framework\v4.0.30319

输入指令 regasm.exe /register /tlb <dll path>

输入指令 regasm.exe <dll path> /codebase

如: regasm.exe /register /tlb C:\Users\User\Desktop\Javascript_Sample_Code\GTSPL_SDK.dll

regasm.exe C:\Users\User\source\repos\product\GTSPL_SDK\GTSPL_SDK.dll /codebase

3.范例程序:

LOCAL driver as GTSPL_SDK.Driver

LOCAL filePath

LOCAL printerStatus

filePath= JUSTPATH(SYS(16,0))

driver =CREATEOBJECT("GTSPL_SDK.Driver")

driver.openport("Printer Model ")

driver.setup("54", "40", "2", "3", "0", "3", "0")

driver.sendcommand("DIRECTION 1")

driver.clearbuffer()

driver.barcode("30", "2", "128", "50", "1", "0", "2", "2", "barcode Foxpro Driver")

driver.printerfont("30", "75", "3", "0", "1", "1", "Print Font Foxpro Driver")

driver.downloadbmp(filePath+"\CIRCLE.BMP", "CIRCLE.BMP")

driver.sendcommand("PUTBMP 200,150,"+CHR(34)+"CIRCLE.BMP"+CHR(34))

```
driver.downloadbmp(filePath+"\\impact.ttf", "impact.ttf")  
driver.windowfont(10, 100, 36, 0, 0, 0, "impact", "Foxpro Driver")  
driver.printlabel("1", "1")  
driver.getDLLVersion(0)  
driver.closeport()
```


● Python

1. 为调用 dll，请先安装 pythonnet

安装命令 `pip install pythonnet` 或 `python -m pip install pythonnet`

2. 在程序中导入 clr

```
import clr
```

3. 加入 GTSPL_SDK:

```
clr.AddReference("GTSPL_SDK") # C# DLL
```

```
from GTSPL_SDK import USB # 从 DLL 中导入 USB Class
```

```
from GTSPL_SDK import Driver # 从 DLL 中导入 Driver Class
```

```
from GTSPL_SDK import SocketConnect # 从 DLL 中导入 Socket Class
```

4. 范例程序:

```
//單機列印
```

```
driver=Driver()
```

```
dresult=driver.openport("Printrer Model")
```

```
print(dresult)
```

```
driver.clearbuffer()
```

```
driver.setup("54", "30", "2", "3", "0", "3", "0")
```

```
driver.setDirectionAndMirror(1,1)
```

```
driver.setShift(50)
```

```
driver.setAfterPrintAction(2)
```

```
driver.setOffset(20)
```

```
driver.setCutMode(0,2)
```

```
driver.printReverse(90,90,128,40)
```

```
driver.barcode("30","30", "128", "100", "1", "0", "2","2", "barcode1234567")
```

```
driver.qrcode("220", "260", "M", "3", "A", "0", "AABCB03abcN123")
```

```
driver.printerfont("50", "10", "3", "0", "1", "1", "Print Font 123456")
```

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

```
driver.printerfontblock("35","15","500","90","1","0","2","2","0","1", ttString)
```

```
driver.downloadpcx("UL.PCX", "UL.PCX")
```

```
driver.sendcommand("PUTPCX 150,30,\"UL.PCX\"")
```

```
driver.downloadbmp("CIRCLE.BMP", "CIRCLE.BMP")ex:D:\\VS_Project\\python_sample_code\\UL.PCX
```

```
driver.sendcommand("PUTBMP 150,30,\"CIRCLE.BMP\"")
```

```
driver.windowfont(10, 100, 48, 0, 0, 0, "arial", "C# WIN TEST")
```

```
BitmapResult = driver.Bitmap("500", "70", 300, 250, 1, "test.png")
```

```
BitmapResult=driver.compressBitmap("300", "70", 400, 350, "test.jpg")
```

```
stString="默认简体中文测试"
```

```
driver.printerfont("100", "10", "TSS24.BF2", "0", "1", "1", stString)
```

```
driver.formfeed()
```

```
driver.printlabel("1", "1")
```

```
driver.closeport()
```

//USB 列印

```
usb = USB()
```

```
result=usb.openport_USB()
```

```
def check_connection(result):
```

```
    if result!=1:
```

```
        return "fail to connect"
```

```
    return "connect successfully"
```

```
connection =check_connection(result)
```

```
print(connection)
```

```
usb.clearbuffer_USB()
```

```

usb.setup_USB("54", "30", "2", "3", "0", "3", "0")

usb.setDirectionAndMirror_USB(0,0)

usb.setShift_USB(50)

usb.setAfterPrintAction_USB(2)

usb.setOffset_USB(20)

usb.setCutMode_USB(0,2)

usb.printReverse_USB(90,90,128,40)

usb.barcode_USB("30","30", "128", "100", "1", "0", "2","2", "barcode1234567")

usb.qrcode_USB("220", "260", "M", "3", "A", "0", "AABCB03abcN123")

usb.printerfont_USB("50", "10", "3", "0", "1", "1", "Print Font 123456")

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

usb.printerfontblock_USB("35","15","500","90","1","0","2","2","0","1", ttString)

usb.downloadpcx_USB("UL.PCX", "UL.PCX")

usb.sendcommand_USB("PUTPCX 150,30,\"UL.PCX\"")

usb.downloadbmp_USB("CIRCLE.BMP", "CIRCLE.BMP")

ex:D:\\VS_Project\\python_sample_code\\UL.PCX

usb.sendcommand_USB("PUTBMP 150,30,\"CIRCLE.BMP\"")

usb.windowfont_USB(10, 100, 48, 0, 0, 0, "arial", "C# WIN TEST")

BitmapResult = usb.Bitmap_USB("500", "70", 300, 250, 1, "test.png")

BitmapResult=usb.compressBitmap_USB("300", "70", 400, 350, "test.jpg")

stString="默认简体中文测试"

usb.printerfont_USB ("100", "10", "TSS24.BF2", "0", "1", "1", stString)

usb.printlabel_USB("1","1")

usb.formfeed_USB()

status=usb.printerstatus_USB()

```

```

print(status)

usb.closeport_USB( )

//乙太網路列印

socket = SocketConnect()

result=socket.openport_Ethernet("192.168.66.137", 9100)

socket.clearbuffer_Ethernet()

socket.setup_Ethernet("54", "30", "2", "3", "0", "3", "0")

socket.setDirectionAndMirror_Ethernet(1,1)

socket.setShift_Ethernet(50)

socket.setAfterPrintAction_Ethernet(2)

socket.setOffset_Ethernet(20)

socket.setCutMode_Ethernet(0,2)

socket.printReverse_Ethernet(90,90,128,40)

socket.barcode_Ethernet("30","30", "128", "100", "1", "0", "2","2", "barcode1234567")

socket.qrcode_Ethernet("220", "260", "M", "3", "A", "0", "AABCB03abcN123")

socket.printerfont_Ethernet("50", "10", "3", "0", "1", "1", "Print Font 123456")

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

socket.printerfontblock_Ethernet("35","15","500","90","1","0","2","2","0","1", ttString)

socket.downloadpcx_Ethernet("UL.PCX", "UL.PCX")

socket.sendcommand_Ethernet("PUTPCX 150,30,\"UL.PCX\"")

socket.downloadbmp_Ethernet("CIRCLE.BMP", "CIRCLE.BMP")

ex:D:\\VS_Project\\python_sample_code\\UL.PCX

socket.sendcommand_Ethernet("PUTBMP 150,30,\"CIRCLE.BMP\"")

socket.windowfont_Ethernet(10, 100, 48, 0, 0, 0, "arial", "C# WIN TEST")

```

```
BitmapResult = socket.Bitmap_Ethernet("500", "70", 300, 250, 1, "test.png")  
BitmapResult=socket.compressBitmap_Ethernet("300", "70", 400, 350, "test.jpg")  
stString="默认简体中文测试"  
socket.printerfont_Ethernet("100", "10", "TSS24.BF2", "0", "1", "1", stString)  
socket.formfeed_Ethernet()  
status=socket.printerstatus_Ethernet()  
print(status )  
socket.printlabel_Ethernet("1", "1")  
socket.closeport_Ethernet()
```

1. 使用系统管理员开启命令提示字元
2. 进入 Windows 的 Microsoft.NET Framework 目录 (系统预设为下列位置, 若有变更路径请自行调整)

`cd C:\Windows\Microsoft.NET\Framework\v4.0.30319`

3. 注册 dll

`RegAsm.exe` dll 的绝对路径 `/tlb:dll 档名.tlb /codebase`

命令范例：

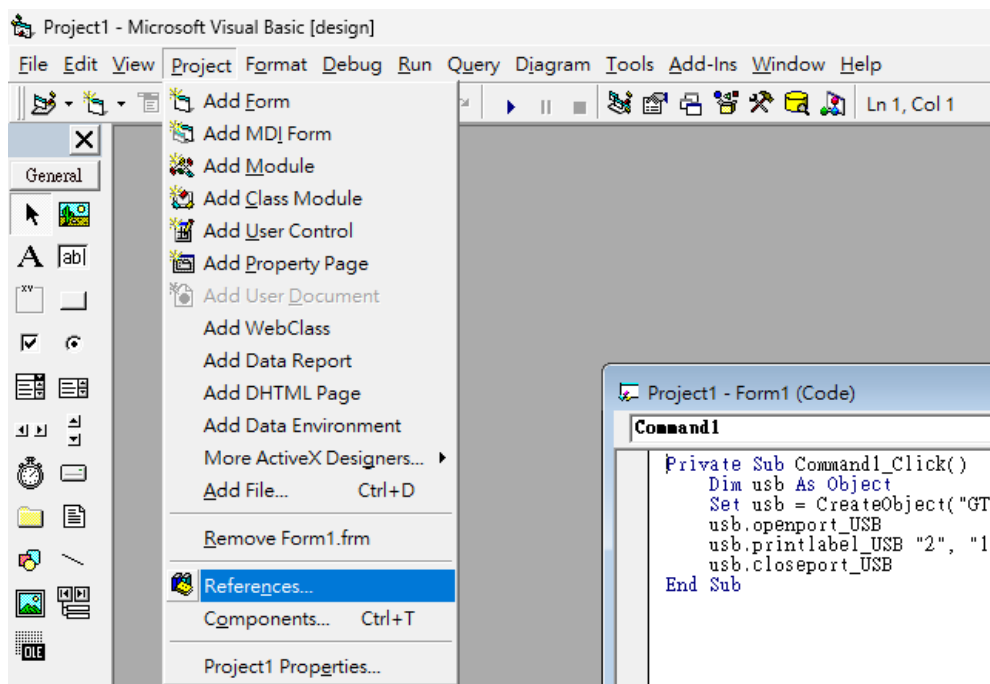
`RegAsm.exe D:\Program\gtspl_sdk\GTSPL_SDK\bin\x86\Release\GTSPL_SDK.dll /tlb: GTSPL_SDK.tlb /codebase`

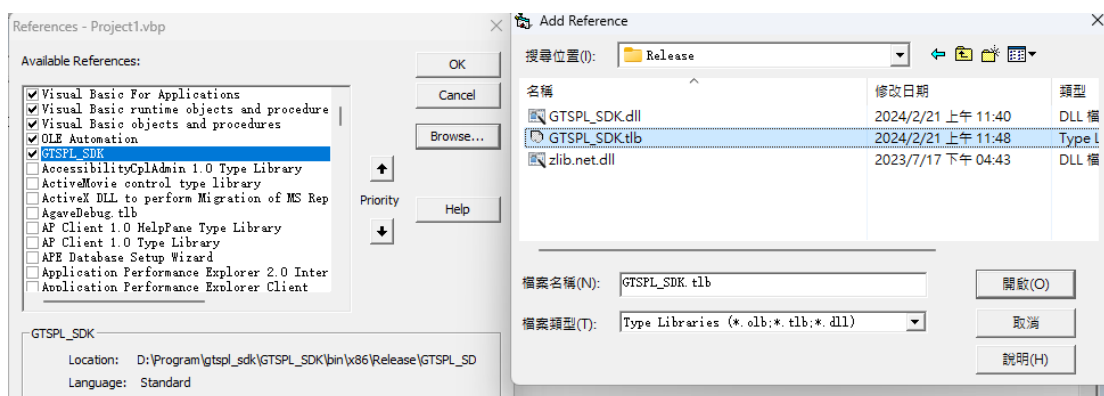
```
C:\Windows\System32>cd C:\Windows\Microsoft.NET\Framework\v4.0.30319
C:\Windows\Microsoft.NET\Framework\v4.0.30319>RegAsm.exe D:\Program\gtspl_sdk\GTSPL_SDK\bin\x86\Release\GTSPL_SDK.dll /tlb:GTSPL_SDK.tlb /codebase
Microsoft .NET Framework Assembly Registration Utility version 4.8.9032.0
for Microsoft .NET Framework version 4.8.9032.0
Copyright (C) Microsoft Corporation. All rights reserved.

RegAsm : warning RA0000 : Registering an unsigned assembly with /codebase can cause your assembly to interfere with other applications that may be installed on the same computer. The /codebase switch is intended to be used only with signed assemblies. Please give your assembly a strong name and re-register it.
Types registered successfully
Type library exporter warning processing 'GTSPL_SDK.Driver+DOCINFOA, GTSPL_SDK'. Warning: The reference type had sequential or explicit layout, and so was exported as a struct.
Assembly exported to 'D:\Program\gtspl_sdk\GTSPL_SDK\bin\x86\Release\GTSPL_SDK.tlb', and the type library was registered successfully
```

4. 加入参考：

VB 开启后，Project>References...里面加入 SDK(选择 tlb)





5. 范例程式：

//单机列印

Dim driver. As Object

Dim dresult As Integer

Dim ttString As String

Dim BitmapResult As Integer

Dim stString As String

Dim VerString As String

Set driver = CreateObject("GTSPL_SDK.Driver")

dresult =driver.openport("Printrer Model")

Print dresult

driver.clearbuffer

driver.setup "54", "30", "2", "3", "0", "3", "0"

driver.setDirectionAndMirror 1,1

driver.setShift 50

driver.setAfterPrintAction 2

driver.setOffset 20

driver.setCutMode 0,2

driver.printReverse 90,90,128,40

driver.barcode "30","30", "128", "100", "1", "0", "2","2", "barcode1234567"

```

driver.qrcode "220", "260", "M", "3", "A", "0", "AABCB03abcN123"

driver.printerfont "50", "10", "3", "0", "1", "1", "Print Font 123456"

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

driver.printerfontblock "35", "15", "500", "90", "1", "0", "2", "2", "0", "1", ttString

driver.downloadpcx "[UL.PCX 的绝对路径]", "UL.PCX"

driver.sendcommand "PUTPCX 150,30,\"UL.PCX\""

driver.downloadbmp "[CIRCLE.BMP 的绝对路径]", "CIRCLE.BMP"

ex: D:\Microsoft Visual Studio\VB98\Project\UL.PCX

driver.sendcommand "PUTBMP 150,30,\"CIRCLE.BMP\""

driver.windowsfont 10, 100, 48, 0, 0, 0, "arial", "C# WIN TEST"

BitmapResult = driver.Bitmap("500", "70", 300, 250, 1, "[test.png 的绝对路径]")

BitmapResult=driver.compressBitmap("300", "70", 400, 350, "[test.jpg 的绝对路径]")

stString="默认简体中文测试"

driver.printerfont "100", "10", "TSS24.BF2", "0", "1", "1", stString

driver.formfeed

driver.printlabel "1", "1"

VerString =driver.getDLLVersion(0)

driver.closeport

```

//USB 列印

```

Dim usb As Object

Dim ttString As String

Dim BitmapResult As Integer

Dim stString As String

Dim Status As String

```


Dim VerString As String

Set usb = CreateObject("GTSPL_SDK.USB")

usb.openport_USB

usb.clearbuffer_USB

usb.setup_USB "54", "30", "2", "3", "0", "3", "0"

usb.setDirectionAndMirror_USB 0,0

usb.setShift_USB 50

usb.setAfterPrintAction_USB 2

usb.setOffset_USB 20

usb.setCutMode_USB 0,2

usb.printReverse_USB 90,90,128,40

usb.barcode_USB "30","30", "128", "100", "1", "0", "2","2", "barcode1234567"

usb.qrcode_USB "220", "260", "M", "3", "A", "0", "AABCB03abcN123"

usb.printerfont_USB "50", "10", "3", "0", "1", "1", "Print Font 123456"

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

usb.printerfontblock_USB "35","15", "500", "90", "1", "0", "2", "2", "0", "1", ttString

usb.downloadpcx_USB "[UL.PCX 的绝对路径]", "UL.PCX"

usb.sendcommand_USB "PUTPCX 150,30,\"UL.PCX\""

usb.downloadbmp_USB "[CIRCLE.BMP 的绝对路径]", "CIRCLE.BMP"

ex: D:\Microsoft Visual Studio\VB98\Project\UL.PCX

usb.sendcommand_USB "PUTBMP 150,30,\"CIRCLE.BMP\""

usb.windowfont_USB 10, 100, 48, 0, 0, 0, "arial", "C# WIN TEST"

BitmapResult = usb.Bitmap_USB("500", "70", 300, 250, 1, "[test.png 的绝对路径]")

BitmapResult=usb.compressBitmap_USB("300", "70", 400, 350, "[test.jpg 的绝对路径]")

stString="默认简体中文测试"

```

usb.printerfont_USB "100", "10", "TSS24.BF2", "0", "1", "1", stString
usb.printlabel_USB "1", "1"
usb.formfeed_USB
status=usb.printerstatus_USB()
Print status
VerString =usb.getDLLVersion_USB(0)
usb.closeport_USB

```

//网口列印

```

Dim socket As Object
Dim ttString As String
Dim BitmapResult As Integer
Dim stString As String
Dim Status As String
Dim VerString As String
Set socket = CreateObject("GTSPL_SDK.SocketConnect")
socket.openport_Ethernet("192.168.66.137", 9100)
socket.clearbuffer_Ethernet
socket.setup_Ethernet "54", "30", "2", "3", "0", "3", "0"
socket.setDirectionAndMirror_Ethernet 1,1
socket.setShift_Ethernet 50
socket.setAfterPrintAction_Ethernet 2
socket.setOffset_Ethernet 20
socket.setCutMode_Ethernet 0,2
socket.printReverse_Ethernet 90,90,128,40
socket.barcode_Ethernet "30","30","128","100","1","0","2","2","barcode1234567"

```

socket.qrcode_Ethernet "220", "260", "M", "3", "A", "0", "AABCB03abcN123"

socket.printerfont_Ethernet "50", "10", "3", "0", "1", "1", "Print Font 123456"

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

socket.printerfontblock_Ethernet "35", "15", "500", "90", "1", "0", "2", "2", "0", "1", ttString

socket.downloadpcx_Ethernet "[UL.PCX 的绝对路径]", "UL.PCX"

socket.sendcommand_Ethernet "PUTPCX 150,30,\"UL.PCX\""

socket.downloadbmp_Ethernet "[CIRCLE.BMP 的绝对路径]", "CIRCLE.BMP"

ex: D:\Microsoft Visual Studio\VB98\Project\UL.PCX

socket.sendcommand_Ethernet "PUTBMP 150,30,\"CIRCLE.BMP\""

socket.windowfont_Ethernet 10, 100, 48, 0, 0, 0, "arial", "C# WIN TEST"

BitmapResult = socket.Bitmap_Ethernet "500", "70", 300, 250, 1, "[test.png 的绝对路径]"

BitmapResult=socket.compressBitmap_Ethernet "300", "70", 400, 350, "[test.jpg 的绝对路径]"

stString="默认简体中文测试"

socket.printerfont_Ethernet "100", "10", "TSS24.BF2", "0", "1", "1", stString

socket.formfeed_Ethernet

status=socket.printerstatus_Ethernet()

Print status

socket.printlabel_Ethernet "1", "1"

VerString=socket.getDLLVersion_Ethernet(0)

socket.closeport_Ethernet

● PHP

1. 使用系统管理员开启命令提示字元
2. 进入 Windows 的 Microsoft.NET Framework 目录 (系统预设为下列位置，若有变更路径请自行调整)

64 位元：

`cd C:\Windows\Microsoft.NET\Framework64\v4.0.30319`

32 位元：

`cd C:\Windows\Microsoft.NET\Framework64\v4.0.30319`

3. 注册 dll

`regasm /codebase dll` 的绝对路径

命令范例：

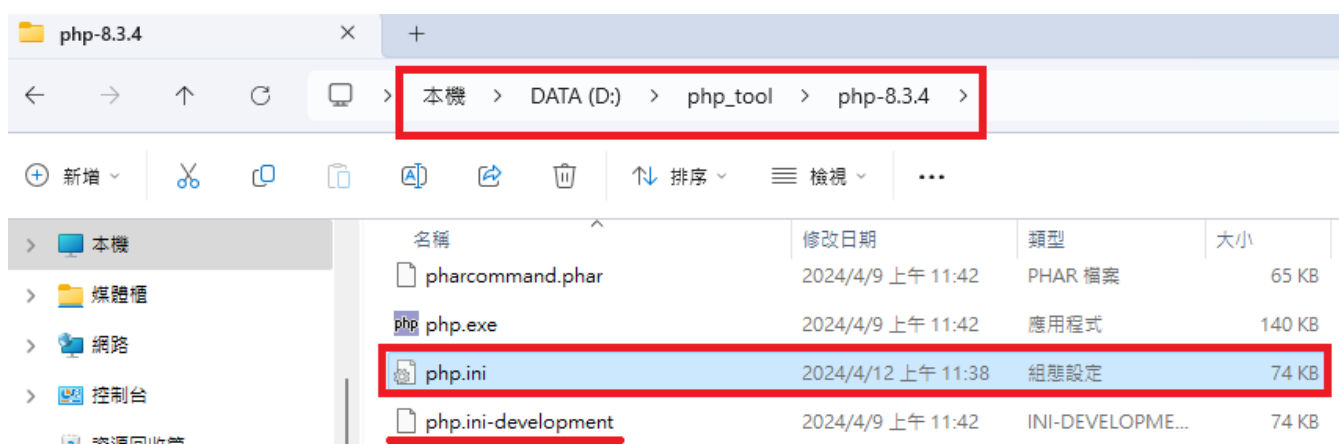
`regasm /codebase D:\Program\gtspl_sdk\GTSPL_SDK\bin\x64\Release\GTSPL_SDK.dll`

```
C:\Windows\System32>cd C:\Windows\Microsoft.NET\Framework64\v4.0.30319
C:\Windows\Microsoft.NET\Framework64\v4.0.30319>regasm /codebase D:\Program\gtspl_sdk\GTSPL_SDK\bin\x64\Release\GTSPL_SDK.dll
Microsoft .NET Framework Assembly Registration Utility 版本 4.8.9032.0
for Microsoft .NET Framework 版本 4.8.9032.0
Copyright (C) Microsoft Corporation. 著作權所有，並保留一切權利。
RegAsm : warning RA0000 : 使用 /codebase 選項註冊一個 unsigned 組件，您的組件可能會和安裝在同一部電腦上的其他應用程式相衝突。/codebase 選項通常只在已簽署的組件。請以強式名稱命名您的組件並重新註冊。
已成功註冊類型
C:\Windows\Microsoft.NET\Framework64\v4.0.30319>
```

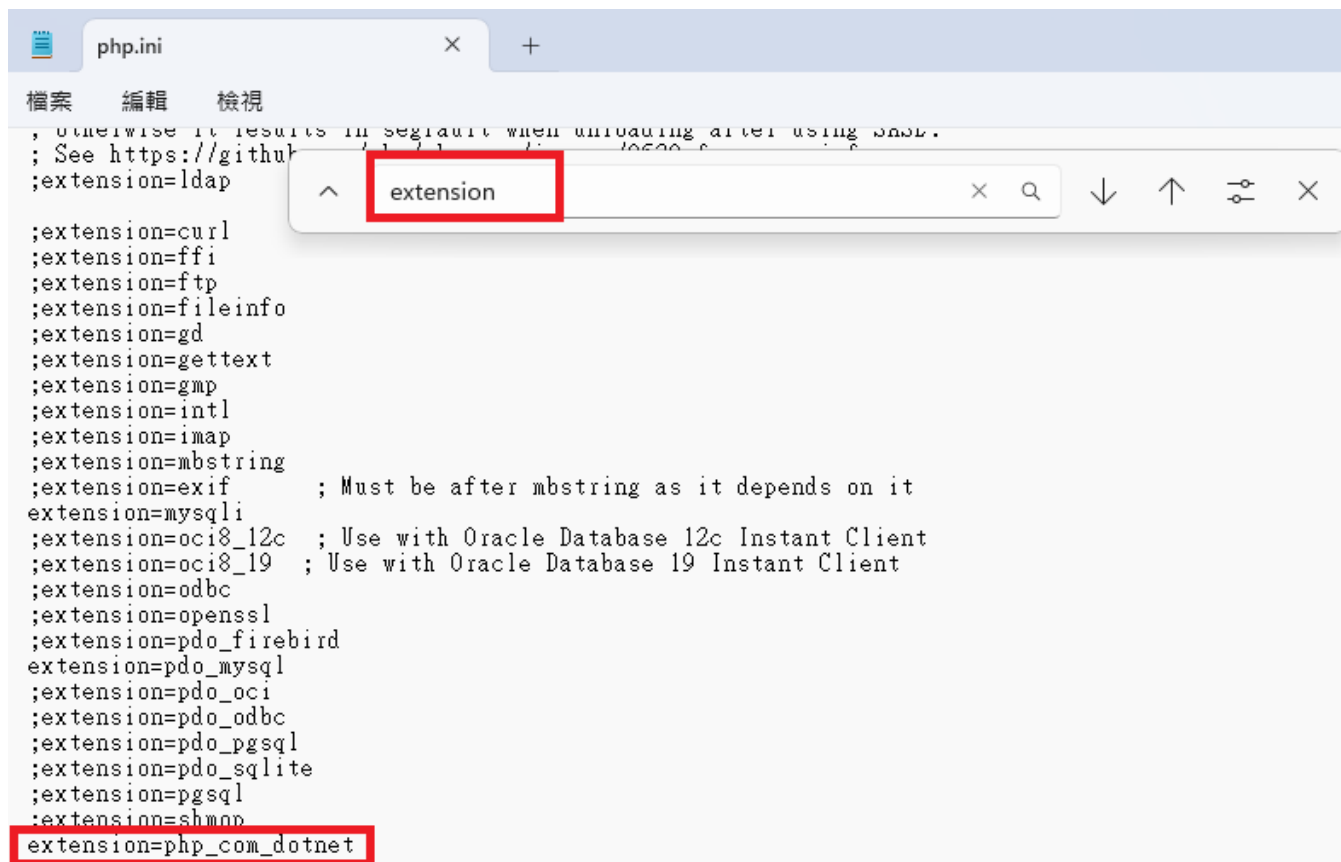
4. 启用 com 类别

到 PHP 的资料夹寻找 php.ini(若没有，将 php.ini-development 备份，再将其改为 php.ini 即可)

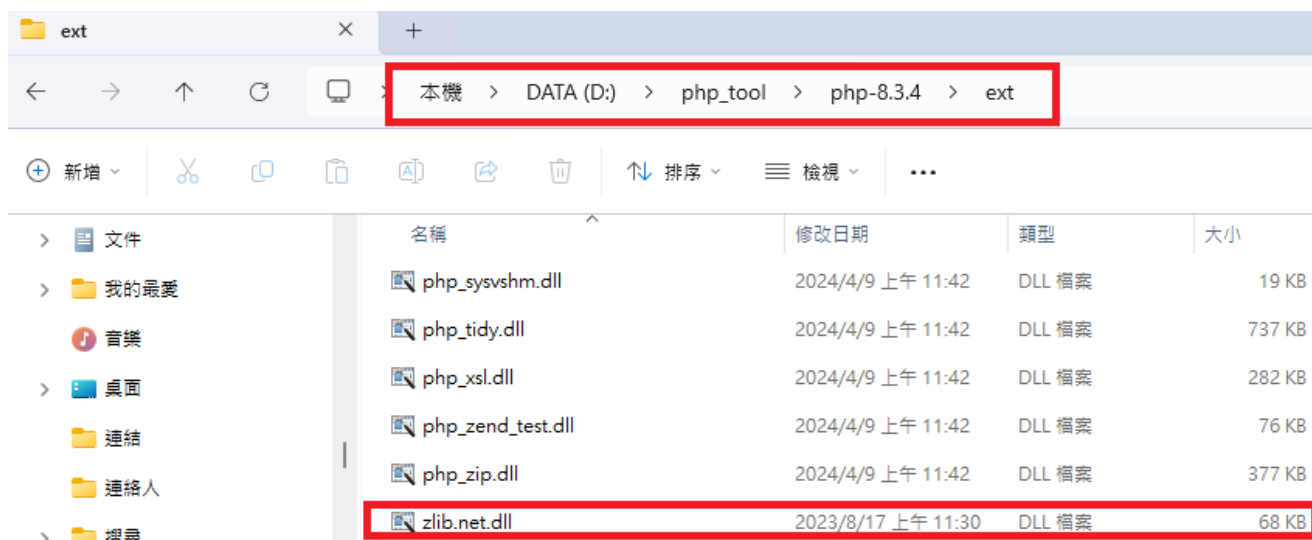
范例如下：



开启 php.ini 后，搜寻"extension="，在一堆"extension="的最后一行新增
"extension=php_com_dotnet"，若文件本身已经有，则不用再新增，若前面有分号则删掉分号储存即可。



※若需使用 compressBitmap，则需要到以下路径，放进已提供的 zlib.net.dll，才可正常使用。



5. 范例程式：

//单机列印

```
$dll = new Com("GTSPL_SDK.Driver");
```

```

$dll->openport("Printrer Model");

$dll->clearbuffer();

$dll ->setup("54", "30", "2", "3", "0", "3", "0");

$dll->setDirectionAndMirror(1,1);

$dll->setShift(50);

$dll->setAfterPrintAction(2);

$dll->setOffset(20);

$dll->setCutMode(0,2);

$dll->printReverse(90,90,128,40);

$dll->barcode("30","30", "128", "100", "1", "0", "2","2", "barcode1234567");

$dll->qrcode("220", "260", "M", "3", "A", "0", "AABCB03abcN123");

$dll->printerfont("50", "10", "3", "0", "1", "1", "Print Font 123456");

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

$dll->printerfontblock("35","15","500","90","1","0","2","2","0","1", $ttString);

$path="C:\Users\sw_user1\Desktop\TEMP\TEST_PIC\UL.PCX ";

$dll->downloadpcx($path, "UL.PCX");

$dll->sendcommand("PUTPCX 150,30,\"UL.PCX\");

$path1="C:\Users\sw_user1\Desktop\TEMP\TEST_PIC\ CIRCLE.BMP ";

$dll->downloadbmp($path1, "CIRCLE.BMP") ;

$dll->$dll->sendcommand("PUTBMP 150,30,\"CIRCLE.BMP\");

$dll->windowsfont(10, 100, 48, 0, 0, 0, "arial", "C# WIN TEST");

$path2="C:\Users\sw_user1\Desktop\TEMP\TEST_PIC\ test.png ";

$BitmapResult = $dll->Bitmap("500", "70", 300, 250, 1, $path2);

$path3="C:\Users\sw_user1\Desktop\TEMP\TEST_PIC\ test.jpg ";

$BitmapResult1=$dll->compressBitmap("300", "70", 400, 350, $path3) ;

```

```

$stString="默认简体中文测试";

$dll->printerfont("100", "10", "TSS24.BF2", "0", "1", "1", $stString);

$dll->formfeed();

$dll->printlabel("1", "1");

$verString=$dll-> getDLLVersion(1);

$dll->closeport();

```

//USB 列印

```

$dll = new Com("GTSPL_SDK.USB");

$dll->usb.openport_USB();

$dll->clearbuffer_USB();

$dll->setup_USB("54", "30", "2", "3", "0", "3", "0");

$dll->setDirectionAndMirror_USB(0,0);

$dll->setShift_USB(50);

$dll->setAfterPrintAction_USB(2);

$dll->setOffset_USB(20);

$dll->setCutMode_USB(0,2);

$dll->printReverse_USB(90,90,128,40);

$dll->barcode_USB("30","30", "128", "100", "1", "0", "2","2", "barcode1234567");

$dll->qrcode_USB("220", "260", "M", "3", "A", "0", "AABCB03abcN123");

$dll->printerfont_USB("50", "10", "3", "0", "1", "1", "Print Font 123456");

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

$dll->printerfontblock_USB("35","15","500","90","1","0","2","2","0","1", $ttString);

$path="C:\Users\sw_user1\Desktop\TEMP\TEST_PIC\UL.PCX ";

$dll->downloadpcx_USB($path, "UL.PCX");

```

```

$dll->sendcommand_USB("PUTPCX 150,30,\"UL.PCX\");
$path1="C:\Users\sw_user1\Desktop\TEMP\TEST_PIC\CIRCLE.BMP ";
$dll->downloadbmp_USB($path1, "CIRCLE.BMP") ;
$dll->sendcommand_USB("PUTBMP 150,30,\"CIRCLE.BMP\");
$dll->windowsfont_USB(10, 100, 48, 0, 0, 0, "arial", "C# WIN TEST");
$path2="C:\Users\sw_user1\Desktop\TEMP\TEST_PIC\ test.png ";
$BitmapResult = $dll->Bitmap_USB("500", "70", 300, 250, 1, $path2);
$path3="C:\Users\sw_user1\Desktop\TEMP\TEST_PIC\ test.jpg ";
$BitmapResult1=$dll->compressBitmap_USB("300", "70", 400, 350, $path3);
$stString="默认简体中文测试";
$dll->printerfont_USB ("100", "10", "TSS24.BF2", "0", "1", "1", $stString);
$dll->printlabel_USB("1", "1");
$dll->formfeed_USB();
$status=$dll->usb.printerstatus_USB();
echo $status;
$VerString=$dll-> getDLLVersion_USB(1);
$dll->closeport_USB( );

```

//以太网列印

```

socket = SocketConnect();
$dll->openport_Ethernet("192.168.66.137", 9100);
$dll->clearbuffer_Ethernet();
$dll->setup_Ethernet("54", "30", "2", "3", "0", "3", "0");
$dll->setDirectionAndMirror_Ethernet(1,1);
$dll->setShift_Ethernet(50);
$dll->setAfterPrintAction_Ethernet(2);

```



```

$dll->setOffset_Ethernet(20);

$dll->setCutMode_Ethernet(0,2);

$dll->$dll->printReverse_Ethernet(90,90,128,40);

$dll->barcode_Ethernet("30","30","128","100","1","0","2","2","barcode1234567");

$dll->qrcode_Ethernet("220","260","M","3","A","0","AABCB03abcN123");

$dll->printerfont_Ethernet("50","10","3","0","1","1","Print Font 123456");

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

$dll->printerfontblock_Ethernet("35","15","500","90","1","0","2","2","0","1",$ttString);

$path="C:\Users\sw_user1\Desktop\TEMP\TEST_PIC\UL.PCX ";

$dll->downloadpcx_Ethernet($path, "UL.PCX");

$dll->sendcommand_Ethernet("PUTPCX 150,30,\"UL.PCX\"");

$path1="C:\Users\sw_user1\Desktop\TEMP\TEST_PIC\ CIRCLE.BMP ";

$dll->downloadbmp_Ethernet($path1, "CIRCLE.BMP") ;

$dll->sendcommand_Ethernet("PUTBMP 150,30,\"CIRCLE.BMP\"");

$dll->windowsfont_Ethernet(10, 100, 48, 0, 0, 0, "arial", "C# WIN TEST");

$path2="C:\Users\sw_user1\Desktop\TEMP\TEST_PIC\ test.png ";

$BitmapResult = $dll->Bitmap_Ethernet("500", "70", 300, 250, 1, $path2);

$path3="C:\Users\sw_user1\Desktop\TEMP\TEST_PIC\ test.jpg ";

$BitmapResult1=$dll->compressBitmap_Ethernet("300", "70", 400, 350, $path3);

$stString="默认简体中文测试";

$dll->printerfont_Ethernet("100", "10", "TSS24.BF2", "0", "1", "1", $stString);

$dll->formfeed_Ethernet();

$status=$dll->printerstatus_Ethernet();

echo $status;

$dll->printlabel_Ethernet("1", "1");

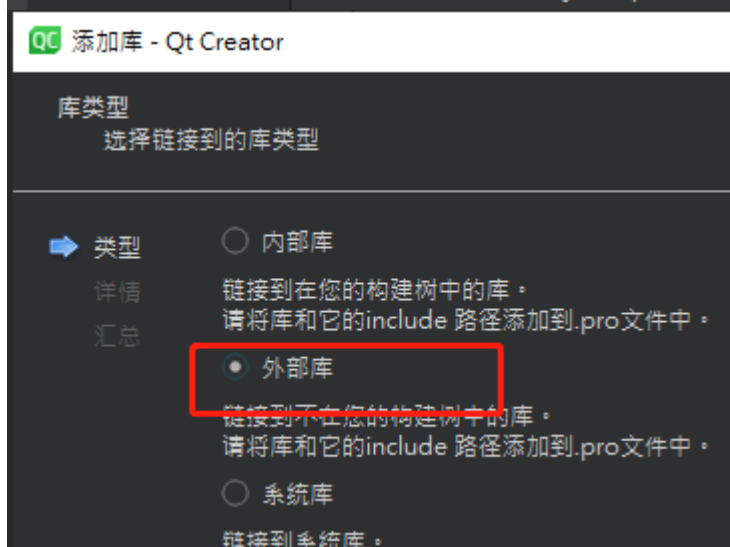
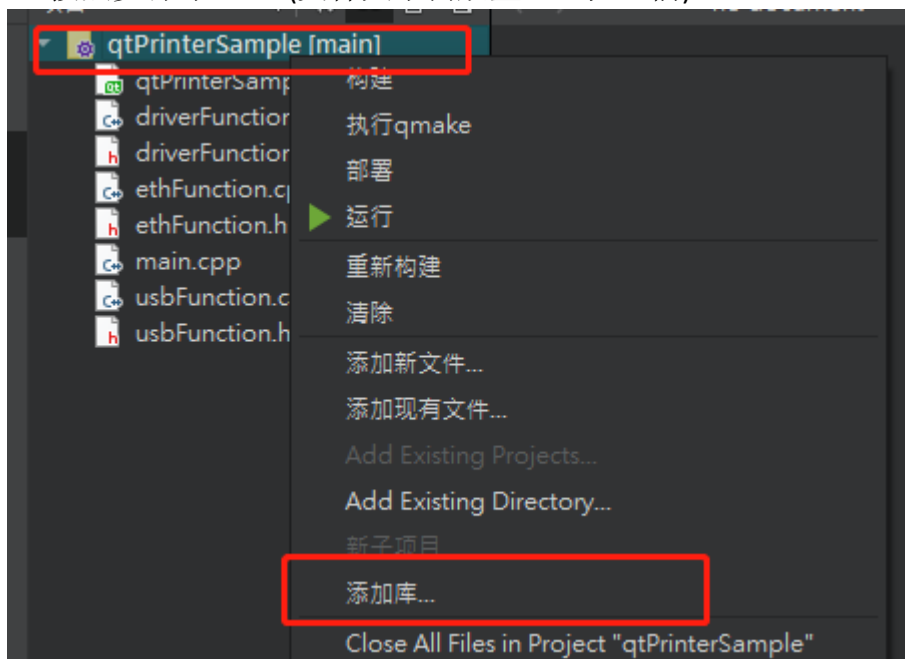
```

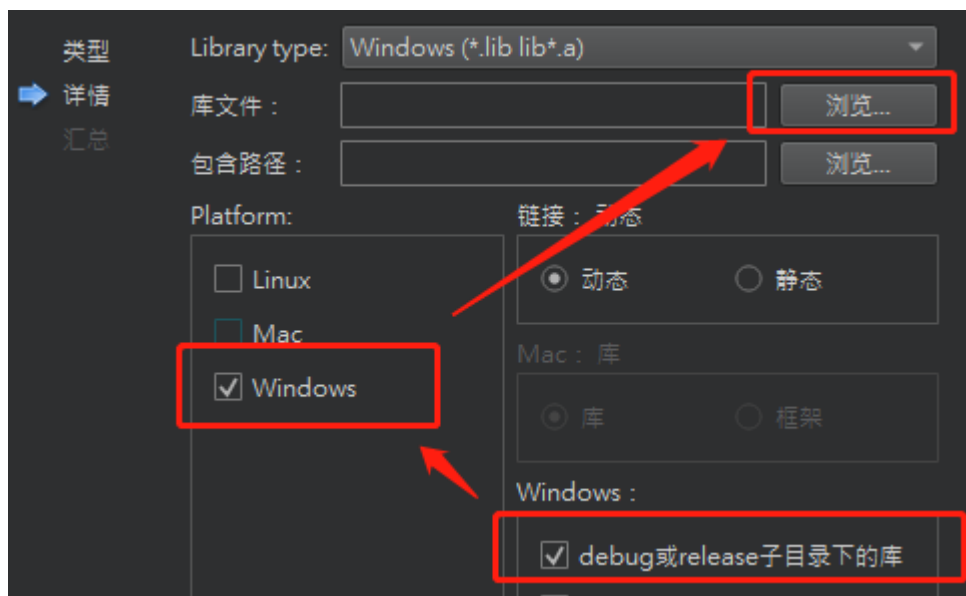
```
$VerString =$dll-> getDLLVersion_Ethernet (1);
```

```
$dll->closeport_Ethernet();
```

● Qt C++

1. 请确保 项目使用 Desktop Qt MSVC2019 64bit 运行
2. 按照步骤导入 lib (文件夹下需放置.lib 与.dll 檔)





2. 先透过 typedef 定义函式指标，再透过 HMODULE (LoadLibrary) 与 GetProcAddress 来取得函式指针并调用。

//引入 dll

```
HMODULE hModule = LoadLibrary(L"GTSP_L_SDK_C.dll");
```

```
if (!hModule)
```

```
{
```

```
    qDebug() << "Failed to load CLR DLL";
```

```
    return 1;
```

```
}
```

单机打印范例

////===== 定义函式指标=====

```
typedef void (*sendcommand)(char* command);
```

```
typedef int (*openport)(char* PrinterName);
```

```
typedef void (*closeport)();
```

```
typedef void (*setup)(char*, char*, char*, char*, char*, char*, char*);
```

```
typedef void (*barcode)(char*, char*, char*, char*, char*, char*, char*, char*, char* );
```

```
typedef void (*printLabel)(char*,char*);
```

```
typedef void (*printerfont)(char*, char*, char* , char* , char* , char* , char* );
```

```
typedef void (*printerfontUnicode)(char* , char* , char* , char* , char* , char* , wchar_t* );
```

```
typedef void (*clearbuffer)();
```

```
typedef void (*downloadpcx)(char*,char*);
```

```
typedef void (*downloadbmp)(char* , char* );
```

```
typedef void (*windowsfont)(int , int , int , int , int , int , char* , char* );
```

```

typedef void (*printerfontblock)(char* , char* , char* , char* , char* , char* , char* , char* , char* , char* ,
char* );
typedef void (*qrcode)(char* , char* , char* , char* , char* , char* , char* );
typedef void (*setDirectionAndMirror)(int , int );
typedef void (*setShift)(int);
typedef void (*printReverse)(int , int , int , int );
typedef void (*setAfterPrintAction)(int);
typedef void (*setOffset)(double);
typedef void (*setCutMode)(int , int );
typedef void (*genericDefault)();
typedef void (*sensorDefault)();
typedef void (*WifiFrequency)(char*);
typedef int (*Bitmap)(char* , char* , int , int , int , char* );
typedef int (*compressBitmap)(char* , char* , int , int , char* );

```

////===== 动态函数调用撰写 =====

// 直接发送命令

```

void sendcommandDriver(const std::string& command)
{
    sendcommand sendCommand = (sendcommand)GetProcAddress(g_hModu_Driver, "sendcommand");
    if (!sendCommand) {
        qDebug() << "Failed to resolve openport_Ethernet function from CLR DLL";
        return;
    }

    sendCommand(const_cast<char*>(command.c_str()));
}

```

// 开启 Driver 通道

```

int openPortDriver(const std::string& printerName)
{
    openport openPort = (openport)GetProcAddress(g_hModu_Driver, "openport");
    if (!openPort) {
        qDebug() << "Failed to resolve openport_USB function from CLR DLL";
        return 0;
    }
}

```

```

int result = openPort(const_cast<char*>(printerName.c_str()));

return result;
}

// 关闭 Driver 通道
void closeportDriver(void)
{
    closeport closePort = (closeport)GetProcAddress(g_hModu_Driver, "closeport");
    if (!closePort) {
        qDebug() << "Failed to resolve closeport_USB function from CLR DLL";
        return ;
    }
    closePort();
}

// 发送设定值
void setupDriver(const std::string& width, const std::string& height, const std::string& speed,
                const std::string& density, const std::string& sensor, const std::string& vertical,
                const std::string& offset)
{
    setup setPrinter = (setup)GetProcAddress(g_hModu_Driver, "setup");
    if (!setPrinter) {
        qDebug() << "Failed to resolve openport function from CLR DLL";
        return;
    }

    setPrinter(const_cast<char*>(width.c_str()), const_cast<char*>(height.c_str()),
               const_cast<char*>(speed.c_str()), const_cast<char*>(density.c_str()),
               const_cast<char*>(sensor.c_str()), const_cast<char*>(vertical.c_str()),
               const_cast<char*>(offset.c_str()));
}

// 打印条形码指令
void barcode_Driver(const std::string& x, const std::string& y, const std::string& type,
                  const std::string& height, const std::string& readable, const std::string& rotation,
                  const std::string& narrow, const std::string& wide, const std::string& code)
{

```

```

barcode barcodeDriver = (barcode)GetProcAddress(g_hModu_Driver, "barcode");
if (!barcodeDriver) {
    qDebug() << "Failed to resolve barcode function from CLR DLL";
    return;
}

barcodeDriver(const_cast<char*>(x.c_str()), const_cast<char*>(y.c_str()),
const_cast<char*>(type.c_str()),
               const_cast<char*>(height.c_str()), const_cast<char*>(readable.c_str()),
const_cast<char*>(rotation.c_str()),
               const_cast<char*>(narrow.c_str()), const_cast<char*>(wide.c_str()),
const_cast<char*>(code.c_str()));
}

// 发送打印指令
void printLabelDriver(const std::string& setting, const std::string& copy) {

    printLabel printLabelDriver = (printLabel)GetProcAddress(g_hModu_Driver, "printlabel");
    if (!printLabelDriver) {
        qDebug() << "Failed to resolve printLabelDriver function from CLR DLL";
        return;
    }

    printLabelDriver(const_cast<char*>(setting.c_str()), const_cast<char*>(copy.c_str()));
}

// 打印文字
void printFontDriver(const std::string& x, const std::string& y, const std::string& fontSize,
                    const std::string& rotation, const std::string& xMul, const std::string& yMul,
                    const std::string& text)
{

    printerfont printerFontDriver = (printerfont)GetProcAddress(g_hModu_Driver, "printerfont");
    if (!printerFontDriver) {
        qDebug() << "Failed to resolve printerfont function from CLR DLL";
        return;
    }

```

```

const char* xChar = x.c_str();
const char* yChar = y.c_str();
const char* fontSizeChar = fontSize.c_str();
const char* rotationChar = rotation.c_str();
const char* xMulChar = xMul.c_str();
const char* yMulChar = yMul.c_str();

const char* textChar = text.c_str();

printerFontDriver(const_cast<char*>(xChar), const_cast<char*>(yChar),
const_cast<char*>(fontSizeChar),
                    const_cast<char*>(rotationChar), const_cast<char*>(xMulChar),
const_cast<char*>(yMulChar),
                    const_cast<char*>(textChar));
}

// 打印 Unicode 格式文字
void printFontUnicodeDriver(const std::string& x, const std::string& y, const std::string& fontSize,
                           const std::string& rotation, const std::string& xMul, const std::string& yMul,
                           const QString& text)
{

    printerfontUnicode printerFontUnicode = (printerfontUnicode)GetProcAddress(g_hModu_Driver,
"printerfontUnicode");
    if (!printerFontUnicode) {
        qDebug() << "Failed to resolve printerfont_USB function from CLR DLL";
        return;
    }

    const char* xChar = x.c_str();
    const char* yChar = y.c_str();
    const char* fontSizeChar = fontSize.c_str();
    const char* rotationChar = rotation.c_str();
    const char* xMulChar = xMul.c_str();
    const char* yMulChar = yMul.c_str();

    QVector<wchar_t> wcharArray(text.length() + 1); // +1 用于终止空字符

```



```

text.toWCharArray(wcharArray.data());

printerFontUnicode(const_cast<char*>(xChar), const_cast<char*>(yChar),
const_cast<char*>(fontSizeChar),
                    const_cast<char*>(rotationChar), const_cast<char*>(xMulChar),
const_cast<char*>(yMulChar),
                    wcharArray.data());
}

// 清除 label 版面
void clearBufferDriver(void)
{

    clearbuffer clearBufferDriver = (clearbuffer)GetProcAddress(g_hModu_Driver, "clearbuffer");
    if (!clearBufferDriver) {
        qDebug() << "Failed to resolve clearbuffer function from CLR DLL";
        return;
    }

    clearBufferDriver();
}

// 下载打印 PCX 图片
void downloadPcxDriver(const std::string& filename, const std::string& imagename) {

    downloadpcx downloadPcxFunc = (downloadpcx)GetProcAddress(g_hModu_Driver, "downloadpcx");
    if (!downloadPcxFunc) {
        qDebug() << "Failed to resolve downloadpcx function from CLR DLL";
        return;
    }

    downloadPcxFunc(const_cast<char*>(filename.c_str()), const_cast<char*>(imagename.c_str())); // 将
std::string 转换为 const char*
}

// 下载打印 BMP 图片
void downloadBmpDriver(const std::string& filename, const std::string& imagename) {

```

```

downloadbmp downloadBmpFunc = (downloadbmp)GetProcAddress(g_hModu_Driver,
"downloadbmp");
    if (!downloadBmpFunc) {
        qDebug() << "Failed to resolve downloadpcx function from CLR DLL";
        return;
    }

    downloadBmpFunc(const_cast<char*>(filename.c_str()), const_cast<char*>(imagename.c_str())); //
    将 std::string 转换为 const char*
}

// 使用 font 打印文字
void windowsFontDriver(int x, int y, int fontheight, int rotation, int fontstyle, int fontunderline, const
std::string& szFaceName, const std::string& content)
{

    windowsfont windowsfontDriver = (windowsfont)GetProcAddress(g_hModu_Driver, "windowsfont");
    if (!windowsfontDriver) {
        qDebug() << "Failed to resolve downloadpcx function from CLR DLL";
        return;
    }

    const char* szFaceNameChar = szFaceName.c_str();
    const char* contentChar = content.c_str();
    windowsfontDriver(x, y, fontheight, rotation, fontstyle, fontunderline,
const_cast<char*>(szFaceNameChar), const_cast<char*>(contentChar));
}

// 打印文字块 (可换行)
void printFontblockDriver(const std::string& x, const std::string& y, const std::string& width, const
std::string& height,

                                const std::string& fontname, const std::string& rotation, const std::string&
xmul,

                                const std::string& ymul, const std::string& space, const std::string& align,
                                const std::string& text)
{

```

```

printerfontblock printerFontBlockDriver = (printerfontblock)GetProcAddress(g_hModu_Driver,
"printerfontblock");
if (!printerFontBlockDriver) {
    qDebug() << "Failed to resolve downloadpcx function from CLR DLL";
    return;
}

```

```

const char* xChar = x.c_str();
const char* yChar = y.c_str();
const char* widthChar = width.c_str();
const char* heightChar = height.c_str();
const char* fontnameChar = fontname.c_str();
const char* rotationChar = rotation.c_str();
const char* xmulChar = xmul.c_str();
const char* ymulChar = ymul.c_str();
const char* spaceChar = space.c_str();
const char* alignChar = align.c_str();
const char* textChar = text.c_str();
printerFontBlockDriver(const_cast<char*>(xChar), const_cast<char*>(yChar),
const_cast<char*>(widthChar),
                        const_cast<char*>(heightChar), const_cast<char*>(fontnameChar),
const_cast<char*>(rotationChar),
                        const_cast<char*>(xmulChar), const_cast<char*>(ymulChar),
const_cast<char*>(spaceChar),
                        const_cast<char*>(alignChar), const_cast<char*>(textChar));
}

```

// 打印 qrcode

```

void qrcodeDriver(const std::string& x, const std::string& y, const std::string& ECClevel, const std::string&
cellwidth,
                const std::string& mode, const std::string& rotation, const std::string& content)
{

    qrcode qrcodeDriver = (qrcode)GetProcAddress(g_hModu_Driver, "qrcode");
    if (!qrcodeDriver) {
        qDebug() << "Failed to resolve qrcodeDriver function from CLR DLL";
        return;
    }

```

```

const char* xChar = x.c_str();
const char* yChar = y.c_str();
const char* ECClevelChar = ECClevel.c_str();
const char* cellwidthChar = cellwidth.c_str();
const char* modeChar = mode.c_str();
const char* rotationChar = rotation.c_str();
const char* contentChar = content.c_str();
qrCodeDriver(const_cast<char*>(xChar), const_cast<char*>(yChar), const_cast<char*>(ECClevelChar),
              const_cast<char*>(cellwidthChar), const_cast<char*>(modeChar),
const_cast<char*>(rotationChar),
              const_cast<char*>(contentChar));
}

// 设定打印方向与镜像
void setDirectionAndMirrorDriver(int direction,int mirror)
{
    setDirectionAndMirror setDAM = (setDirectionAndMirror)GetProcAddress(g_hModu_Driver,
"setDirectionAndMirror");

    if (!setDAM) {
        qDebug() << "Failed to resolve qrCodeDriver function from CLR DLL";
        return;
    }

    setDAM(direction,mirror);
}

// 设定 shift
void setShiftDriver(int shift)
{
    setShift setShiftDriver = (setShift)GetProcAddress(g_hModu_Driver, "setShift");

    if (!setShiftDriver) {
        qDebug() << "Failed to resolve qrCodeDriver function from CLR DLL";
        return;
    }
}

```

```

    setShiftDriver(shift);
}

// 打印反色区块
void printReverseDriver(int x_start, int y_start, int x_width, int y_height)
{
    printReverse setReverseDriver = (printReverse)GetProcAddress(g_hModu_Driver,"printReverse");

    if (!setReverseDriver) {
        qDebug() << "Failed to resolve qrcodeDriver function from CLR DLL";
        return;
    }

    setReverseDriver(x_start,y_start,x_width,y_height);
}

// 设定打印后动作
void setAfterPrintModeDriver(int mode)
{
    setAfterPrintAction setAPM
    =(setAfterPrintAction)GetProcAddress(g_hModu_Driver,"setAfterPrintAction");

    if (!setAPM) {
        qDebug() << "Failed to resolve qrcodeDriver function from CLR DLL";
        return;
    }

    setAPM(mode);
}

// 设定 offset
void setOffsetDriver(double offset)
{
    setOffset setOff =(setOffset)GetProcAddress(g_hModu_Driver,"setOffset");

    if (!setOff) {
        qDebug() << "Failed to resolve qrcodeDriver function from CLR DLL";
        return;
    }
}

```

```

        setOff(offset);
    }

// 设定裁切模式
void setCutModeDriver(int mode, int piece)
{
    setCutMode setCut = (setCutMode)GetProcAddress(g_hModu_Driver,"setCutMode");

    if (!setCut) {
        qDebug() << "Failed to resolve qrcodeDriver function from CLR DLL";
        return;
    }

    setCut(mode,piece);
}

// 设定 generic 设定初始化
void genericDefaultDriver(void)
{
    genericDefault setGenericDef = (genericDefault)GetProcAddress(g_hModu_Driver,"genericDefault");

    if (!setGenericDef) {
        qDebug() << "Failed to resolve qrcodeDriver function from CLR DLL";
        return;
    }

    setGenericDef();
}

// 设定 sensor 设定初始化
void sensorDefaultDriver(void)
{
    sensorDefault setSensorDefDriver =
(sensorDefault)GetProcAddress(g_hModu_Driver,"sensorDefault");

    if (!setSensorDefDriver) {
        qDebug() << "Failed to resolve sensorDefault function from CLR DLL";
        return;
    }
}

```

```

    }

    setSensorDefDriver();
}

// 设定 wifi 频段
void setWifiFrequencyDriver(const std::string& frequency)
{
    WifiFrequency wifiFrequencyDriver = (WifiFrequency)GetProcAddress(g_hModu_Driver,
"WifiFrequency");
    if (!wifiFrequencyDriver) {
        qDebug() << "Failed to resolve WifiFrequency function from CLR DLL";
        return;
    }

    const char* frequencyChar = frequency.c_str();

    wifiFrequencyDriver(const_cast<char*>(frequencyChar));
}

// 直接打印 bitmap
int BitmapDriver(const std::string& x, const std::string& y, int width, int height, int mode, const std::string&
filename)
{
    Bitmap bitmapDriver = (Bitmap)GetProcAddress(g_hModu_Driver, "Bitmap");
    if (!bitmapDriver) {
        qDebug() << "Failed to resolve Bitmap function from CLR DLL";
        return 0;
    }

    const char* xChar = x.c_str();
    const char* yChar = y.c_str();
    const char* filenameChar = filename.c_str();

    int result = bitmapDriver(const_cast<char*>(xChar), const_cast<char*>(yChar), width, height, mode,
const_cast<char*>(filenameChar));
}

```

```

        return result;
    }

// 直接打印压缩的 bitmap
int CompressBitmapDriver(const std::string& x, const std::string& y, int width, int height, const std::string&
filename)
{
    compressBitmap bitmapDriver = (compressBitmap)GetProcAddress(g_hModu_Driver,
"compressBitmap");
    if (!bitmapDriver) {
        qDebug() << "Failed to resolve Compress Bitmap function from CLR DLL";
        return 0;
    }

    const char* xChar = x.c_str();
    const char* yChar = y.c_str();
    const char* filenameChar = filename.c_str();

    int result = bitmapDriver(const_cast<char*>(xChar), const_cast<char*>(yChar), width,
height,const_cast<char*>(filenameChar));
    return result;
}

////===== 实际调用范本 =====
#define PRINTER_NAME "Gainscha GS-2406T"

void driverSettingTest(HMODULE hModu)
{
    g_hModu_Driver = hModu;
    int result = 0;

    result = openPortDriver(PRINTER_NAME);
    if(result)
    {
        // 设定方向与镜像

        setDirectionAndMirroDriver(0,0);
    }
}

```



```
    setDirectionAndMirroDriver(1,0);

    setDirectionAndMirroDriver(0,1);

    setDirectionAndMirroDriver(1,1);
    closeportDriver();
}
result = openPortDriver(PRINTER_NAME);
if(result)
{
    // 设定 Shift

    setShiftDriver(80);

    setShiftDriver(50);
    closeportDriver();
}
result = openPortDriver(PRINTER_NAME);
if(result)
{
    // 设定 AfterPrinteAction

    setAfterPrintModeDriver(0);

    setAfterPrintModeDriver(1);

    setAfterPrintModeDriver(2);

    setAfterPrintModeDriver(3);
    closeportDriver();
}
result = openPortDriver(PRINTER_NAME);
if(result)
{
    // 设定 Offset

    setOffsetDriver(50);
```

```

        setOffsetDriver(5);
        closeportDriver();
    }
    result = openPortDriver(PRINTER_NAME);
    if(result)
    {
        // 设定切刀

        setCutModeDriver(0, 2);

        setCutModeDriver(1, 5);
        closeportDriver();
    }
    result = openPortDriver(PRINTER_NAME);
    if(result)
    {
        // 设定 genetic 初始化

        genericDefaultDriver();
        closeportDriver();
    }
    result = openPortDriver(PRINTER_NAME);
    if(result)
    {
        // 设定 WIFI 频段

        setWifiFrequencyDriver("5G");

        setWifiFrequencyDriver("2.4G");
        closeportDriver();
    }
}

void driverSample(HMODULE hModu)
{
    g_hModu_Driver = hModu;
    int result = 0;

    result = openPortDriver(PRINTER_NAME);

```

```

if(result)
{
    // 设定打印机尺寸
    clearBufferDriver();
    setupDriver("104", "80", "2", "7", "0", "3", "0");
    sendcommandDriver("DIRECTION 1");
    closeportDriver();
}
result = openPortDriver(PRINTER_NAME);
if(result)
{
    // 印一张 barcode
    barcode_Driver("30", "30", "128", "100", "1", "0", "2", "2", "barcodeDriver1234567");
    printLabelDriver("1", "1");
    closeportDriver();
}
result = openPortDriver(PRINTER_NAME);
if(result)
{
    // 印一张文字
    clearBufferDriver();
    printFontDriver("50", "10", "TSS24.BF2", "0", "1", "1", "testFontDriver12345678");
    printLabelDriver("1", "1");
    closeportDriver();
}
result = openPortDriver(PRINTER_NAME);
if(result)
{
    // 印一张中文字
    clearBufferDriver();
    printFontUnicodeDriver("50", "10", "TSS24.BF2", "0", "1", "1", "Driver 默认简体中文测试：海天
米醋白米醋 1 瓶 450ml");
    printLabelDriver("1", "1");
    closeportDriver();
}
result = openPortDriver(PRINTER_NAME);
if(result)
{
    // 印一张 PCX 图片

```

```

QString currentDir = QDir::currentPath();
std::string filenameStr = (currentDir + "\\UL.PCX").toStdString();
clearBufferDriver();
downloadPcxDriver(filenameStr, "UL.PCX");
sendcommandDriver("PUTPCX 50,10,\"UL.PCX\"\\r\\n");
printLabelDriver("1", "1");
closeportDriver();
}
result = openPortDriver(PRINTER_NAME);
if(result)
{
    // 印一张 BMP
    QString currentDir = QDir::currentPath();
    std::string filenameStr = (currentDir + "\\CIRCLE.BMP").toStdString();
    clearBufferDriver();
    downloadBmpDriver(filenameStr, "CIRCLE.BMP");
    sendcommandDriver("PUTBMP 10,10,\"CIRCLE.BMP\"\\r\\n");
    printLabelDriver("1", "1");
    closeportDriver();
}
result = openPortDriver(PRINTER_NAME);
if(result)
{
    //印一张 windows font
    QString currentDir = QDir::currentPath();
    clearBufferDriver();
    std::string filenameStr = (currentDir + "\\impact.ttf").toStdString();
    downloadBmpDriver(filenameStr, "impact.ttf");
    windowsFontDriver(10, 100, 48, 0, 0, 0, "impact", "C# WIN TEST");
    printLabelDriver("1", "1");
    closeportDriver();
}
result = openPortDriver(PRINTER_NAME);
if(result)
{
    // 印一张 block text
    clearBufferDriver();
    printFontblockDriver("35", "15", "300", "90", "TSS24.BF2", "0", "1", "1", "0", "1",
"DriverPrintFontBlockTestPrintFontBlockTestPrintFontBlockTestPrintFontBlockTest");
}

```

```

        printLabelDriver("1", "1");
        closeportDriver();
    }
    result = openPortDriver(PRINTER_NAME);
    if(result)
    {
        // 印一张 qrcode
        qrcodeDriver("220", "260", "M", "3", "A", "0", "AABCB03abcN123");
        printLabelDriver("1", "1");
        closeportDriver();
    }
    result = openPortDriver(PRINTER_NAME);
    if(result)
    {
        // 设定 Reverse
        sendcommandDriver("TEXT 100,100,\"3\",0,1,1,\"REVERSE\"");
        printReverseDriver(90, 90, 128, 40);
        printLabelDriver("1", "1");
        closeportDriver();
    }
    result = openPortDriver(PRINTER_NAME);
    if(result)
    {
        // 直接打印 BMP
        if(BitmapDriver("0", "0", 400, 350, 1, "Thunder.png")!=0)
            printLabelDriver("1", "1");
        closeportDriver();
    }
    result = openPortDriver(PRINTER_NAME);
    if(result)
    {
        // 直接打印压缩 BMP
        if(CompressBitmapDriver("0", "0", 666, 357, "Cat1.jpg")!=0)
            printLabelDriver("1", "1");
        closeportDriver();
    }
}

```

Ethernet 打印范例

////===== 定义函式指标=====

```
typedef void (*sendcommand_Ethernet)(char*);
```

```
typedef int (*openport_Ethernet)(char*, int);
```

```
typedef void (*closeport_Ethernet)();
```

```
typedef char* (*printerstatus_Ethernet)();
```

```
typedef void (*setup_Ethernet)(char*, char*, char*, char*, char*, char*, char*);
```

```
typedef void (*barcode_Ethernet)(char*, char*, char*, char*, char*, char*, char*, char*, char*, char* );
```

```
typedef void (*printlabel_Ethernet)(char*,char*);
```

```
typedef void (*printerfont_Ethernet)(char*, char*, char*, char*, char*, char*, char* );
```

```
typedef void (*printerfontUnicode_Ethernet)(char*, char*, char*, char*, char*, char*, char*, wchar_t* );
```

```
typedef void (*clearbuffer_Ethernet)();
```

```
typedef void (*downloadpcx_Ethernet)(char*,char*);
```

```
typedef void (*downloadbmp_Ethernet)(char*, char* );
```

```
typedef void (*windowsfont_Ethernet)(int , int , int , int , int , int , char* , char* );
```

```
typedef void (*printerfontblock_Ethernet)(char* , char* , char* , char* , char* , char* , char* , char* , char* , char* , char* );
```

```
typedef void (*qrcode_Ethernet)(char* , char* , char* , char* , char* , char* , char* );
```

```
typedef void (*setDirectionAndMirror_Ethernet)(int , int );
```

```
typedef void (*setShift_Ethernet)(int);
```

```
typedef void (*printReverse_Ethernet)(int , int , int , int );
```

```
typedef void (*setAfterPrintAction_Ethernet)(int);
```

```
typedef void (*setOffset_Ethernet)(double);
```

```
typedef void (*setCutMode_Ethernet)(int , int );
```

```
typedef void (*genericDefault_Ethernet)();
```

```
typedef void (*sensorDefault_Ethernet)();
```

```
typedef void (*WifiFrequency_Ethernet)(char*);
```

```
typedef int (*Bitmap_Ethernet)(char* , char* , int , int , int , char* );
```

```
typedef int (*compressBitmap_Ethernet)(char* , char* , int , int , char* );
```

////===== 动态函式调用撰写 =====

```
void sendcommand_Eth(const std::string& command)
```

```
{
```

```
    sendcommand_Ethernet sendCommand = (sendcommand_Ethernet)GetProcAddress(g_hModu,
"sendcommand_Ethernet");
```

```
    if (!sendCommand) {
```

```

        qDebug() << "Failed to resolve openport_Ethernet function from CLR DLL";
        return;
    }

    sendCommand(const_cast<char*>(command.c_str()));
}

// 开启以太网信道
int openPortEth(const std::string& ipAddress, int portNumber)
{
    openport_Ethernet openPortEthFunc = (openport_Ethernet)GetProcAddress(g_hModu,
"openport_Ethernet");
    if (!openPortEthFunc) {
        qDebug() << "Failed to resolve openport_Ethernet function from CLR DLL";
        return 0;
    }

    int result = openPortEthFunc(const_cast<char*>(ipAddress.c_str()), portNumber);

    return result;
}

// 关闭以太网信道
void closePortEth(void)
{
    closeport_Ethernet closePortEthFunc = (closeport_Ethernet)GetProcAddress(g_hModu,
"closeport_Ethernet");
    if (!closePortEthFunc) {
        qDebug() << "Failed to resolve openport_Ethernet function from CLR DLL";
        return;
    }

    closePortEthFunc();
}

// 询问打印机状态
char* printerStatusEth()
{

```

```

printerstatus_Ethernet getStatus =(printerstatus_Ethernet)GetProcAddress(g_hModu,
"printerstatus_Ethernet");
    if (!getStatus) {
        qDebug() << "Failed to resolve openport_Ethernet function from CLR DLL";
        return NULL;
    }

    char* status = getStatus();
    return status;
}

// 发送设定值
void setupEth(const std::string& width, const std::string& height, const std::string& speed,
              const std::string& density, const std::string& sensor, const std::string& vertical,
              const std::string& offset)
{

    setup_Ethernet setPrinter = (setup_Ethernet)GetProcAddress(g_hModu, "setup_Ethernet");
    if (!setPrinter) {
        qDebug() << "Failed to resolve openport_Ethernet function from CLR DLL";
        return;
    }

    setPrinter(const_cast<char*>(width.c_str()), const_cast<char*>(height.c_str()),
               const_cast<char*>(speed.c_str()), const_cast<char*>(density.c_str()),
               const_cast<char*>(sensor.c_str()), const_cast<char*>(vertical.c_str()),
               const_cast<char*>(offset.c_str()));
}

// 打印条形码指令
void barcode_Eth(const std::string& x, const std::string& y, const std::string& type,
                 const std::string& height, const std::string& readable, const std::string& rotation,
                 const std::string& narrow, const std::string& wide, const std::string& code)
{

    barcode_Ethernet barcodeEth = (barcode_Ethernet)GetProcAddress(g_hModu, "barcode_Ethernet");
    if (!barcodeEth) {
        qDebug() << "Failed to resolve barcode_Ethernet function from CLR DLL";
        return;
    }

```



```

    }

    barcodeEth(const_cast<char*>(x.c_str()), const_cast<char*>(y.c_str()),
const_cast<char*>(type.c_str()),
                const_cast<char*>(height.c_str()), const_cast<char*>(readable.c_str()),
const_cast<char*>(rotation.c_str()),
                const_cast<char*>(narrow.c_str()), const_cast<char*>(wide.c_str()),
const_cast<char*>(code.c_str()));
}

```

// 发送打印指令

```

void printLabel_Eth(const std::string& setting, const std::string& copy) {

    printlabel_Ethernet printLabelEth = (printlabel_Ethernet)GetProcAddress(g_hModu,
"printlabel_Ethernet");
    if (!printLabelEth) {
        qDebug() << "Failed to resolve printLabel_Eth function from CLR DLL";
        return;
    }

    printLabelEth(const_cast<char*>(setting.c_str()), const_cast<char*>(copy.c_str()));
}

```

// 打印文字

```

void printFontEth(const std::string& x, const std::string& y, const std::string& fontSize,
                const std::string& rotation, const std::string& xMul, const std::string& yMul,
                const std::string& text)
{

    printerfont_Ethernet printerFontEth = (printerfont_Ethernet)GetProcAddress(g_hModu,
"printerfont_Ethernet");
    if (!printerFontEth) {
        qDebug() << "Failed to resolve printerfont_Ethernet function from CLR DLL";
        return;
    }

    const char* xChar = x.c_str();
    const char* yChar = y.c_str();
}

```

```

const char* fontSizeChar = fontSize.c_str();
const char* rotationChar = rotation.c_str();
const char* xMulChar = xMul.c_str();
const char* yMulChar = yMul.c_str();

const char* textChar = text.c_str();

printerFontEth(const_cast<char*>(xChar), const_cast<char*>(yChar),
const_cast<char*>(fontSizeChar),
                const_cast<char*>(rotationChar), const_cast<char*>(xMulChar),
const_cast<char*>(yMulChar),
                const_cast<char*>(textChar));
}

// 打印 Unicode 格式文字
void printFontUnicodeEth(const std::string& x, const std::string& y, const std::string& fontSize,
                        const std::string& rotation, const std::string& xMul, const std::string& yMul,
                        const QString& text)
{

    printerfontUnicode_Ethernet printerFontEthUnicode =
    (printerfontUnicode_Ethernet)GetProcAddress(g_hModu, "printerfontUnicode_Ethernet");
    if (!printerFontEthUnicode) {
        qDebug() << "Failed to resolve printerfontUnicode_Ethernet function from CLR DLL";
        return;
    }

    const char* xChar = x.c_str();
    const char* yChar = y.c_str();
    const char* fontSizeChar = fontSize.c_str();
    const char* rotationChar = rotation.c_str();
    const char* xMulChar = xMul.c_str();
    const char* yMulChar = yMul.c_str();

    QVector<wchar_t> wcharArray(text.length() + 1);
    text.toWCharArray(wcharArray.data());

```

```

        printerFontEthUnicode(const_cast<char*>(xChar), const_cast<char*>(yChar),
const_cast<char*>(fontSizeChar),
                                const_cast<char*>(rotationChar), const_cast<char*>(xMulChar),
const_cast<char*>(yMulChar),
                                wcharArray.data());
    }

```

// 清除 label 版面

```

void clearBufferEth(void)
{

```

```

    clearbuffer_Ethernet clearBufferEth = (clearbuffer_Ethernet)GetProcAddress(g_hModu,
"clearbuffer_Ethernet");
    if (!clearBufferEth) {
        qDebug() << "Failed to resolve clearbuffer_Ethernet function from CLR DLL";
        return;
    }

```

```

    clearBufferEth();
}

```

// 下载打印 PCX 图片

```

void downloadPcxEth(const std::string& filename, const std::string& imagename) {

```

```

    downloadpcx_Ethernet downloadPcxFunc = (downloadpcx_Ethernet)GetProcAddress(g_hModu,
"downloadpcx_Ethernet");
    if (!downloadPcxFunc) {
        qDebug() << "Failed to resolve downloadpcx_USB function from CLR DLL";
        return;
    }

```

```

    downloadPcxFunc(const_cast<char*>(filename.c_str()), const_cast<char*>(imagename.c_str())); // 将
std::string 转换为 const char*
}

```

// 下载打印 BMP 图片

```

void downloadBmpEth(const std::string& filename, const std::string& imagename) {

```

```

downloadbmp_Ethernet downloadBmpFunc = (downloadbmp_Ethernet)GetProcAddress(g_hModu,
"downloadbmp_Ethernet");
if (!downloadBmpFunc) {
    qDebug() << "Failed to resolve downloadpcx_USB function from CLR DLL";
    return;
}

```

```

downloadBmpFunc(const_cast<char*>(filename.c_str()), const_cast<char*>(imagename.c_str())); //
将 std::string 转换为 const char*
}

```

// 使用 font 打印文字

```

void windowsFontEth(int x, int y, int fontheight, int rotation, int fontstyle, int fontunderline, const
std::string& szFaceName, const std::string& content)
{

```

```

    windowsfont_Ethernet windowsfontEth = (windowsfont_Ethernet)GetProcAddress(g_hModu,
"windowsfont_Ethernet");
    if (!windowsfontEth) {
        qDebug() << "Failed to resolve downloadpcx_USB function from CLR DLL";
        return;
    }

```

```

    const char* szFaceNameChar = szFaceName.c_str();
    const char* contentChar = content.c_str();
    windowsfontEth(x, y, fontheight, rotation, fontstyle, fontunderline,
const_cast<char*>(szFaceNameChar), const_cast<char*>(contentChar));
}

```

// 打印文字块 (可换行)

```

void printFontblockEth(const std::string& x, const std::string& y, const std::string& width, const std::string&
height,
                        const std::string& fontname, const std::string& rotation, const std::string&
xmul,
                        const std::string& ymul, const std::string& space, const std::string& align,
                        const std::string& text)
{

```

```

printerfontblock_Ethernet printerFontBlockEth =
(printerfontblock_Ethernet)GetProcAddress(g_hModu, "printerfontblock_Ethernet");
if (!printerFontBlockEth) {
    qDebug() << "Failed to resolve downloadpcx_USB function from CLR DLL";
    return;
}

const char* xChar = x.c_str();
const char* yChar = y.c_str();
const char* widthChar = width.c_str();
const char* heightChar = height.c_str();
const char* fontnameChar = fontname.c_str();
const char* rotationChar = rotation.c_str();
const char* xmulChar = xmul.c_str();
const char* ymulChar = ymul.c_str();
const char* spaceChar = space.c_str();
const char* alignChar = align.c_str();
const char* textChar = text.c_str();
printerFontBlockEth(const_cast<char*>(xChar), const_cast<char*>(yChar),
const_cast<char*>(widthChar),
                    const_cast<char*>(heightChar), const_cast<char*>(fontnameChar),
const_cast<char*>(rotationChar),
                    const_cast<char*>(xmulChar), const_cast<char*>(ymulChar),
const_cast<char*>(spaceChar),
                    const_cast<char*>(alignChar), const_cast<char*>(textChar));
}
// 打印 qrcode
void qrcodeEth(const std::string& x, const std::string& y, const std::string& ECClevel, const std::string&
cellwidth,
              const std::string& mode, const std::string& rotation, const std::string& content)
{

qrcode_Ethernet qrcodeEth = (qrcode_Ethernet)GetProcAddress(g_hModu, "qrcode_Ethernet");
if (!qrcodeEth) {
    qDebug() << "Failed to resolve qrcodeEth function from CLR DLL";
    return;
}

```

```

const char* xChar = x.c_str();
const char* yChar = y.c_str();
const char* ECClevelChar = ECClevel.c_str();
const char* cellwidthChar = cellwidth.c_str();
const char* modeChar = mode.c_str();
const char* rotationChar = rotation.c_str();
const char* contentChar = content.c_str();
qrcodeEth(const_cast<char*>(xChar), const_cast<char*>(yChar), const_cast<char*>(ECClevelChar),
           const_cast<char*>(cellwidthChar), const_cast<char*>(modeChar),
const_cast<char*>(rotationChar),
           const_cast<char*>(contentChar));
}

// 设定打印方向与镜像
void setDirectionAndMirroEth(int direction,int mirror)
{
    setDirectionAndMirror_Ethernet setDAM =
(setDirectionAndMirror_Ethernet)GetProcAddress(g_hModu, "setDirectionAndMirror_Ethernet");

    if (!setDAM) {
        qDebug() << "Failed to resolve qrcodeEth function from CLR DLL";
        return;
    }

    setDAM(direction,mirror);
}

// 设定 shift
void setShiftEth(int shift)
{
    setShift_Ethernet setShiftEth = (setShift_Ethernet)GetProcAddress(g_hModu, "setShift_Ethernet");

    if (!setShiftEth) {
        qDebug() << "Failed to resolve qrcodeEth function from CLR DLL";
        return;
    }
}

```

```

    setShiftEth(shift);
}

// 打印反色区块
void printReverseEth(int x_start, int y_start, int x_width, int y_height)
{
    printReverse_Ethernet setReverseEth =
    (printReverse_Ethernet)GetProcAddress(g_hModu,"printReverse_Ethernet");

    if (!setReverseEth) {
        qDebug() << "Failed to resolve qrcodeEth function from CLR DLL";
        return;
    }

    setReverseEth(x_start,y_start,x_width,y_height);
}

// 设定打印后动作
void setAfterPrintModeEth(int mode)
{
    setAfterPrintAction_Ethernet setAPM
    =(setAfterPrintAction_Ethernet)GetProcAddress(g_hModu,"setAfterPrintAction_Ethernet");

    if (!setAPM) {
        qDebug() << "Failed to resolve qrcodeEth function from CLR DLL";
        return;
    }

    setAPM(mode);
}

// 设定 offset
void setOffsetEth(double offset)
{
    setOffset_Ethernet setOffset =(setOffset_Ethernet)GetProcAddress(g_hModu,"setOffset_Ethernet");

    if (!setOffset) {
        qDebug() << "Failed to resolve qrcodeEth function from CLR DLL";
        return;
    }
}

```

```

    }

    setOffset(offset);
}

// 设定裁切模式
void setCutModeEth(int mode, int piece)
{
    setCutMode_Ethernet setCutMode =
(setCutMode_Ethernet)GetProcAddress(g_hModu,"setCutMode_Ethernet");

    if (!setCutMode) {
        qDebug() << "Failed to resolve qrcodeEth function from CLR DLL";
        return;
    }

    setCutMode(mode,piece);
}

// 设定 generic 设定初始化
void genericDefaultEth(void)
{
    genericDefault_Ethernet setGenericDef =
(genericDefault_Ethernet)GetProcAddress(g_hModu,"genericDefault_Ethernet");

    if (!setGenericDef) {
        qDebug() << "Failed to resolve qrcodeEth function from CLR DLL";
        return;
    }

    setGenericDef();
}

// 设定 sensor 设定初始化
void sensorDefaultEth(void)
{
    sensorDefault_Ethernet setSensorDefEth =
(sensorDefault_Ethernet)GetProcAddress(g_hModu,"sensorDefault_Ethernet");

```



```

if (!setSensorDefEth) {
    qDebug() << "Failed to resolve qrCodeEth function from CLR DLL";
    return;
}

setSensorDefEth();
}

// 设定 wifi 频段
void setWifiFrequencyEth(const std::string& frequency)
{

    WifiFrequency_Ethernet wifiFrequencyEth = (WifiFrequency_Ethernet)GetProcAddress(g_hModu,
"WifiFrequency_Ethernet");
    if (!wifiFrequencyEth) {
        qDebug() << "Failed to resolve WifiFrequency_Ethernet function from CLR DLL";
        return;
    }

    const char* frequencyChar = frequency.c_str();

    wifiFrequencyEth(const_cast<char*>(frequencyChar));
}

// 直接打印 bitmap
int BitmapEth(const std::string& x, const std::string& y, int width, int height, int mode, const std::string&
filename)
{

    Bitmap_Ethernet bitmapEth = (Bitmap_Ethernet)GetProcAddress(g_hModu, "Bitmap_Ethernet");
    if (!bitmapEth) {
        qDebug() << "Failed to resolve Bitmap_Ethernet function from CLR DLL";
        return 0;
    }

    const char* xChar = x.c_str();
    const char* yChar = y.c_str();
    const char* filenameChar = filename.c_str();

```

```

        int result = bitmapEth(const_cast<char*>(xChar), const_cast<char*>(yChar), width, height, mode,
const_cast<char*>(filenameChar));
        return result;
    }

// 直接打印压缩的 bitmap
int CompressBitmapEth(const std::string& x, const std::string& y, int width, int height, const std::string&
filename)
{
    compressBitmap_Ethernet bitmapEth = (compressBitmap_Ethernet)GetProcAddress(g_hModu,
"compressBitmap_Ethernet");
    if (!bitmapEth) {
        qDebug() << "Failed to resolve Bitmap_Ethernet function from CLR DLL";
        return 0;
    }

    const char* xChar = x.c_str();
    const char* yChar = y.c_str();
    const char* filenameChar = filename.c_str();

    int result = bitmapEth(const_cast<char*>(xChar), const_cast<char*>(yChar), width,
height,const_cast<char*>(filenameChar));
    return result;
}

void sendAnnotationEth(void)
{
    sendcommand_Eth(TSPL_ANNOTATION_COMMAND);
}

//===== MainSampleFunction =====
#define SAMPLE_IP_ADDRESS "192.168.66.135"
#define SAMPLE_PORT_NUMBER 9100

void ethernetSettingSample(HMODULE hModu)
{

```

```
g_hModu = hModu;
int result = 0;
result = openPortEth(SAMPLE_IP_ADDRESS,SAMPLE_PORT_NUMBER);
if(result)
{
    // 设定方向与镜像
    sendAnnotationEth();
    setDirectionAndMirroEth(0,0);
    sendAnnotationEth();
    setDirectionAndMirroEth(1,0);
    sendAnnotationEth();
    setDirectionAndMirroEth(0,1);
    sendAnnotationEth();
    setDirectionAndMirroEth(1,1);

    // 设定 Shift
    sendAnnotationEth();
    setShiftEth(180);
    sendAnnotationEth();
    setShiftEth(50);

    // 设定 AfterPrinteAction
    sendAnnotationEth();
    setAfterPrintModeEth(0);
    sendAnnotationEth();
    setAfterPrintModeEth(1);
    sendAnnotationEth();
    setAfterPrintModeEth(2);
    sendAnnotationEth();
    setAfterPrintModeEth(3);

    // 设定 Offset
    sendAnnotationEth();
    setOffsetEth(50);
    sendAnnotationEth();
    setOffsetEth(5);

    // 设定切刀
    sendAnnotationEth();
```

```

setCutModeEth(0, 2);
sendAnnotationEth();
setCutModeEth(1, 5);

// 设定 genetic 初始化
sendAnnotationEth();
genericDefaultEth();

// 设定 WIFI 频段
sendAnnotationEth();
setWifiFrequencyEth("5G");
sendAnnotationEth();
setWifiFrequencyEth("2.4G");
}
}
////===== 实际调用范本 =====
#define SAMPLE_IP_ADDRESS "192.168.66.135"
#define SAMPLE_PORT_NUMBER 9100

void ethernetSettingSample(HMODULE hModu)
{
    g_hModu = hModu;
    int result = 0;
    result = openPortEth(SAMPLE_IP_ADDRESS,SAMPLE_PORT_NUMBER);
    if(result)
    {
        // 设定方向与镜像
        sendAnnotationEth();
        setDirectionAndMirroEth(0,0);
        sendAnnotationEth();
        setDirectionAndMirroEth(1,0);
        sendAnnotationEth();
        setDirectionAndMirroEth(0,1);
        sendAnnotationEth();
        setDirectionAndMirroEth(1,1);

        // 设定 Shift
        sendAnnotationEth();
        setShiftEth(180);
    }
}

```

```

sendAnnotationEth();
setShiftEth(50);

// 设定 AfterPrintAction
sendAnnotationEth();
setAfterPrintModeEth(0);
sendAnnotationEth();
setAfterPrintModeEth(1);
sendAnnotationEth();
setAfterPrintModeEth(2);
sendAnnotationEth();
setAfterPrintModeEth(3);

// 设定 Offset
sendAnnotationEth();
setOffsetEth(50);
sendAnnotationEth();
setOffsetEth(5);

// 设定切刀
sendAnnotationEth();
setCutModeEth(0, 2);
sendAnnotationEth();
setCutModeEth(1, 5);

// 设定 genetic 初始化
sendAnnotationEth();
genericDefaultEth();

// 设定 WIFI 频段
sendAnnotationEth();
setWifiFrequencyEth("5G");
sendAnnotationEth();
setWifiFrequencyEth("2.4G");
}
}

```

```

void ethernetSample(HMODULE hModu)
{

```

```

g_hModu = hModu;
int result = 0;
result = openPortEth(SAMPLE_IP_ADDRESS,SAMPLE_PORT_NUMBER);
if(result)
{
// 取得打印机信息
char* printerStatus = printerStatusEth();

// 设定打印机尺寸
clearBufferEth();
setupEth("104", "76", "2", "7", "0", "3", "0");
sendcommand_Eth("DIRECTION 1");

// 印一张 barcode
barcode_Eth("30", "30", "128", "100", "1", "0", "2", "2", "barcode1234567");
printLabel_Eth("1", "1");
// 印一张中文字
printFontUnicodeEth("50", "10", "TSS24.BF2", "0", "1", "1", "以太网路默认简体中文测试：海天米醋
白米醋 1 瓶 450ml");
printLabel_Eth("1", "1");
// 印一张文字
clearBufferEth();
printFontEth("50", "10", "TSS24.BF2", "0", "1", "1", "testFont12345678");
printLabel_Eth("1", "1");

// 印一张 PCX 图片
QString currentDir = QDir::currentPath();
std::string filenameStr = (currentDir + "\\UL.PCX").toStdString();
clearBufferEth();
downloadPcxEth(filenameStr, "UL.PCX");
sendcommand_Eth("PUTPCX 50,10,\"UL.PCX\"\\r\\n");
printLabel_Eth("1", "1");

// 印一张 BMP
currentDir = QDir::currentPath();
filenameStr = (currentDir + "\\CIRCLE.BMP").toStdString();
clearBufferEth();
downloadBmpEth(filenameStr, "CIRCLE.BMP");
sendcommand_Eth("PUTBMP 10,10,\"CIRCLE.BMP\"\\r\\n");

```

```

printLabel_Eth("1", "1");

//印一张 windows font
currentDir = QDir::currentPath();
clearBufferEth();
filenameStr = (currentDir + "\\impact.ttf").toStdString();
downloadBmpEth(filenameStr, "impact.ttf");
windowsFontEth(10, 100, 48, 0, 0, 0, "impact", "C# WIN TEST");
printLabel_Eth("1", "1");

// 印一张 block text
clearBufferEth();
printFontblockEth("35", "15", "300", "90", "TSS24.BF2", "0", "1", "1", "0", "1",
"PrintFontBlockTestPrintFontBlockTestPrintFontBlockTestPrintFontBlockTest");
printLabel_Eth("1", "1");

// 印一张 qrcode
qrcodeEth("220", "260", "M", "3", "A", "0", "AABCB03abcN123");
printLabel_Eth("1", "1");

// 设定 Reverse
sendcommand_Eth("TEXT 100,100,\"3\",0,1,1,\"REVERSE\"");
printReverseEth(90, 90, 128, 40);
printLabel_Eth("1", "1");

// 直接打印 BMP
if(BitmapEth("0", "0", 400, 350, 1, "Thunder.png")!=0)
    printLabel_Eth("1", "1");

// 直接打印压缩 BMP
if(CompressBitmapEth("0", "0", 666, 357, "Cat1.jpg")!=0)
    printLabel_Eth("1", "1");

closePortEth();
}
}

```

USB 打印范例

////===== 定义函式指标=====

```

typedef void (*sendcommand_USB)(char* command);

typedef int (*openport_USB)();
typedef void (*closeport_USB)();

typedef char* (*printerstatus_USB)();
typedef void (*setup_USB)(char*, char*, char*, char*, char*, char*, char*);
typedef void (*barcode_USB)(char*, char*, char*, char*, char*, char*, char*, char*, char*, char* );
typedef void (*printLabel_Usb)(char*,char*);
typedef void (*printerfont_USB)(char*, char*, char*, char*, char*, char*, char* );
typedef void (*printerfontUnicode_USB)(char*, char*, char*, char*, char*, char*, char*, wchar_t* );
typedef void (*clearbuffer_USB)();
typedef void (*downloadpcx_USB)(char*,char*);
typedef void (*downloadbmp_USB)(char*, char* );
typedef void (*windowsfont_USB)(int , int , int , int , int , int , char* , char* );
typedef void (*printerfontblock_USB)(char* , char* , char* , char* , char* , char* , char* , char* , char* , char* , char* , char* );
typedef void (*qrcode_USB)(char* , char* , char* , char* , char* , char* , char* );
typedef void (*setDirectionAndMirror_USB)(int , int );
typedef void (*setShift_USB)(int);
typedef void (*printReverse_USB)(int , int , int , int );
typedef void (*setAfterPrintAction_USB)(int);
typedef void (*setOffset_USB)(double);
typedef void (*setCutMode_USB)(int , int );
typedef void (*genericDefault_USB)();
typedef void (*sensorDefault_USB)();
typedef void (*WifiFrequency_USB)(char*);
typedef int (*Bitmap_USB)(char* , char* , int , int , int , char* );
typedef int (*compressBitmap_USB)(char* , char* , int , int , char* );
////===== 动态函数调用撰写 =====
void sendcommandUsb(const std::string& command)
{
    sendcommand_USB sendCommand = (sendcommand_USB)GetProcAddress(g_hModu_usb,
"sendcommand_USB");
    if (!sendCommand) {
        qDebug() << "Failed to resolve openport_Ethernet function from CLR DLL";
        return;
    }
}

```



```

        sendCommand(const_cast<char*>(command.c_str()));
    }
// 开启 USB 通道
int openPortUsb(void)
{
    openport_USB openPortUsb = (openport_USB)GetProcAddress(g_hModu_usb, "openport_USB");
    if (!openPortUsb) {
        qDebug() << "Failed to resolve openport_USB function from CLR DLL";
        return 0;
    }
    int result = openPortUsb();

    return result;
}

// 关闭 USB 通道
void closeportUSB(void)
{
    closeport_USB closePortUsb = (closeport_USB)GetProcAddress(g_hModu_usb, "closeport_USB");
    if (!closePortUsb) {
        qDebug() << "Failed to resolve closeport_USB function from CLR DLL";
        return ;
    }
    closePortUsb();
}

// 询问打印机状态
char* printerStatusUsb()
{
    printerstatus_USB getStatus =(printerstatus_USB)GetProcAddress(g_hModu_usb,
"printerstatus_USB");
    if (!getStatus) {
        qDebug() << "Failed to resolve openport_USB function from CLR DLL";
        return NULL;
    }
    char* status = getStatus();
    return status;
}

// 发送设定值
void setupUsb(const std::string& width, const std::string& height, const std::string& speed,

```

```

        const std::string& density, const std::string& sensor, const std::string& vertical,
        const std::string& offset)
{
    setup_USB setPrinter = (setup_USB)GetProcAddress(g_hModu_usb, "setup_USB");
    if (!setPrinter) {
        qDebug() << "Failed to resolve openport_USB function from CLR DLL";
        return;
    }
    setPrinter(const_cast<char*>(width.c_str()), const_cast<char*>(height.c_str()),
               const_cast<char*>(speed.c_str()), const_cast<char*>(density.c_str()),
               const_cast<char*>(sensor.c_str()), const_cast<char*>(vertical.c_str()),
               const_cast<char*>(offset.c_str()));
}

// 打印条形码指令
void barcode_Usb(const std::string& x, const std::string& y, const std::string& type,
                 const std::string& height, const std::string& readable, const std::string& rotation,
                 const std::string& narrow, const std::string& wide, const std::string& code)
{
    barcode_USB barcodeUsb = (barcode_USB)GetProcAddress(g_hModu_usb, "barcode_USB");
    if (!barcodeUsb) {
        qDebug() << "Failed to resolve barcode_USB function from CLR DLL";
        return;
    }

    barcodeUsb(const_cast<char*>(x.c_str()), const_cast<char*>(y.c_str()),
               const_cast<char*>(type.c_str()),
               const_cast<char*>(height.c_str()), const_cast<char*>(readable.c_str()),
               const_cast<char*>(rotation.c_str()),
               const_cast<char*>(narrow.c_str()), const_cast<char*>(wide.c_str()),
               const_cast<char*>(code.c_str()));
}

// 发送打印指令
void printLabelUsb(const std::string& setting, const std::string& copy) {

    printLabel_Usb printLabelUsb = (printLabel_Usb)GetProcAddress(g_hModu_usb, "printlabel_USB");

```

```

if (!printLabelUsb) {
    qDebug() << "Failed to resolve printLabelUsb function from CLR DLL";
    return;
}

// 调用 printLabelUsb 函数发送打印命令
printLabelUsb(const_cast<char*>(setting.c_str()), const_cast<char*>(copy.c_str()));
}

// 打印文字
void printFontUsb(const std::string& x, const std::string& y, const std::string& fontSize,
                  const std::string& rotation, const std::string& xMul, const std::string& yMul,
                  const std::string& text)
{
    printerfont_USB printerFontUsb = (printerfont_USB)GetProcAddress(g_hModu_usb,
"printerfont_USB");
    if (!printerFontUsb) {
        qDebug() << "Failed to resolve printerfont_USB function from CLR DLL";
        return;
    }

    const char* xChar = x.c_str();
    const char* yChar = y.c_str();
    const char* fontSizeChar = fontSize.c_str();
    const char* rotationChar = rotation.c_str();
    const char* xMulChar = xMul.c_str();
    const char* yMulChar = yMul.c_str();

    const char* textChar = text.c_str();

    printerFontUsb(const_cast<char*>(xChar), const_cast<char*>(yChar),
const_cast<char*>(fontSizeChar),
                  const_cast<char*>(rotationChar), const_cast<char*>(xMulChar),
const_cast<char*>(yMulChar),
                  const_cast<char*>(textChar));
}

// 打印 Unicode 格式文字

```

```

void printFontUnicodeUsb(const std::string& x, const std::string& y, const std::string& fontSize,
                        const std::string& rotation, const std::string& xMul, const std::string& yMul,
                        const QString& text)
{
    printerfontUnicode_USB printerFontUsbUnicode =
(printerfontUnicode_USB)GetProcAddress(g_hModu_usb, "printerfontUnicode_USB");
    if (!printerFontUsbUnicode) {
        qDebug() << "Failed to resolve printerfont_USB function from CLR DLL";
        return;
    }

    const char* xChar = x.c_str();
    const char* yChar = y.c_str();
    const char* fontSizeChar = fontSize.c_str();
    const char* rotationChar = rotation.c_str();
    const char* xMulChar = xMul.c_str();
    const char* yMulChar = yMul.c_str();

    QVector<wchar_t> wcharArray(text.length() + 1);
    text.toWCharArray(wcharArray.data());

    printerFontUsbUnicode(const_cast<char*>(xChar), const_cast<char*>(yChar),
const_cast<char*>(fontSizeChar),
                        const_cast<char*>(rotationChar), const_cast<char*>(xMulChar),
const_cast<char*>(yMulChar),
                        wcharArray.data());
}

// 清除 label 版面
void clearBufferUsb(void)
{
    clearbuffer_USB clearBufferUsb = (clearbuffer_USB)GetProcAddress(g_hModu_usb,
"clearbuffer_USB");
    if (!clearBufferUsb) {
        qDebug() << "Failed to resolve clearbuffer_USB function from CLR DLL";
        return;
    }
}

```

```

        clearBufferUsb();
    }

// 下载打印 PCX 图片
void downloadPcxUsb(const std::string& filename, const std::string& imagename) {

    downloadpcx_USB downloadPcxFunc = (downloadpcx_USB)GetProcAddress(g_hModu_usb,
"downloadpcx_USB");
    if (!downloadPcxFunc) {
        qDebug() << "Failed to resolve downloadpcx_USB function from CLR DLL";
        return;
    }

    downloadPcxFunc(const_cast<char*>(filename.c_str()), const_cast<char*>(imagename.c_str())); // 将
std::string 转换为 const char*
}

// 下载打印 BMP 图片
void downloadBmpUsb(const std::string& filename, const std::string& imagename) {

    downloadbmp_USB downloadBmpFunc = (downloadbmp_USB)GetProcAddress(g_hModu_usb,
"downloadbmp_USB");
    if (!downloadBmpFunc) {
        qDebug() << "Failed to resolve downloadpcx_USB function from CLR DLL";
        return;
    }

    downloadBmpFunc(const_cast<char*>(filename.c_str()), const_cast<char*>(imagename.c_str())); //
将 std::string 转换为 const char*
}

// 使用 font 打印文字
void windowsFontUsb(int x, int y, int fontheight, int rotation, int fontstyle, int fontunderline, const
std::string& szFaceName, const std::string& content)
{

```

```

    windowsfont_USB windowsfontUsb = (windowsfont_USB)GetProcAddress(g_hModu_usb,
"windowsfont_USB");
    if (!windowsfontUsb) {
        qDebug() << "Failed to resolve downloadpcx_USB function from CLR DLL";
        return;
    }

    const char* szFaceNameChar = szFaceName.c_str();
    const char* contentChar = content.c_str();
    windowsfontUsb(x, y, fontheight, rotation, fontstyle, fontunderline,
const_cast<char*>(szFaceNameChar), const_cast<char*>(contentChar));
}

// 打印文字块 (可换行)
void printFontblockUsb(const std::string& x, const std::string& y, const std::string& width, const std::string&
height,
                        const std::string& fontname, const std::string& rotation, const std::string&
xmul,
                        const std::string& ymul, const std::string& space, const std::string& align,
                        const std::string& text)
{

    printerfontblock_USB printerFontBlockUsb = (printerfontblock_USB)GetProcAddress(g_hModu_usb,
"printerfontblock_USB");
    if (!printerFontBlockUsb) {
        qDebug() << "Failed to resolve downloadpcx_USB function from CLR DLL";
        return;
    }

    const char* xChar = x.c_str();
    const char* yChar = y.c_str();
    const char* widthChar = width.c_str();
    const char* heightChar = height.c_str();
    const char* fontnameChar = fontname.c_str();
    const char* rotationChar = rotation.c_str();
    const char* xmulChar = xmul.c_str();
    const char* ymulChar = ymul.c_str();
    const char* spaceChar = space.c_str();

```

```

const char* alignChar = align.c_str();
const char* textChar = text.c_str();
printerFontBlockUsb(const_cast<char*>(xChar), const_cast<char*>(yChar),
const_cast<char*>(widthChar),
                    const_cast<char*>(heightChar), const_cast<char*>(fontnameChar),
const_cast<char*>(rotationChar),
                    const_cast<char*>(xmulChar), const_cast<char*>(ymulChar),
const_cast<char*>(spaceChar),
                    const_cast<char*>(alignChar), const_cast<char*>(textChar));
}
// 打印 qrcode
void qrcodeUsb(const std::string& x, const std::string& y, const std::string& ECClevel, const std::string&
cellwidth,
               const std::string& mode, const std::string& rotation, const std::string& content)
{
    qrcode_USB qrcodeUsb = (qrcode_USB)GetProcAddress(g_hModu_usb, "qrcode_USB");
    if (!qrcodeUsb) {
        qDebug() << "Failed to resolve qrcodeUsb function from CLR DLL";
        return;
    }

    const char* xChar = x.c_str();
    const char* yChar = y.c_str();
    const char* ECClevelChar = ECClevel.c_str();
    const char* cellwidthChar = cellwidth.c_str();
    const char* modeChar = mode.c_str();
    const char* rotationChar = rotation.c_str();
    const char* contentChar = content.c_str();
    qrcodeUsb(const_cast<char*>(xChar), const_cast<char*>(yChar), const_cast<char*>(ECClevelChar),
              const_cast<char*>(cellwidthChar), const_cast<char*>(modeChar),
const_cast<char*>(rotationChar),
              const_cast<char*>(contentChar));
}

// 设定打印方向与镜像
void setDirectionAndMirroUsb(int direction,int mirror)
{

```

```

    setDirectionAndMirror_USB setDAM = (setDirectionAndMirror_USB)GetProcAddress(g_hModu_usb,
"setDirectionAndMirror_USB");

```

```

    if (!setDAM) {
        qDebug() << "Failed to resolve qrcodeUsb function from CLR DLL";
        return;
    }

    setDAM(direction,mirror);
}

// 设定 shift
void setShiftUsb(int shift)
{
    setShift_USB setShiftUsb = (setShift_USB)GetProcAddress(g_hModu_usb, "setShift_USB");

    if (!setShiftUsb) {
        qDebug() << "Failed to resolve qrcodeUsb function from CLR DLL";
        return;
    }

    setShiftUsb(shift);
}

// 打印反色区块
void printReverseUsb(int x_start, int y_start, int x_width, int y_height)
{
    printReverse_USB setReverseUsb =
(printReverse_USB)GetProcAddress(g_hModu_usb,"printReverse_USB");

    if (!setReverseUsb) {
        qDebug() << "Failed to resolve qrcodeUsb function from CLR DLL";
        return;
    }

    setReverseUsb(x_start,y_start,x_width,y_height);
}

// 设定打印后动作

```



```

void setAfterPrintModeUsb(int mode)
{
    setAfterPrintAction_USB setAPM
=(setAfterPrintAction_USB)GetProcAddress(g_hModu_usb,"setAfterPrintAction_USB");

    if (!setAPM) {
        qDebug() << "Failed to resolve qrcodeUsb function from CLR DLL";
        return;
    }

    setAPM(mode);
}

// 设定 offset
void setOffsetUsb(double offset)
{
    setOffset_USB setOffset =(setOffset_USB)GetProcAddress(g_hModu_usb,"setOffset_USB");

    if (!setOffset) {
        qDebug() << "Failed to resolve qrcodeUsb function from CLR DLL";
        return;
    }

    setOffset(offset);
}

// 设定裁切模式
void setCutModeUsb(int mode, int piece)
{
    setCutMode_USB setCutMode =
(setCutMode_USB)GetProcAddress(g_hModu_usb,"setCutMode_USB");

    if (!setCutMode) {
        qDebug() << "Failed to resolve qrcodeUsb function from CLR DLL";
        return;
    }

    setCutMode(mode,piece);
}

```

```

// 设定 generic 设定初始化
void genericDefaultUsb(void)
{
    genericDefault_USB setGenericDef =
(genericDefault_USB)GetProcAddress(g_hModu_usb,"genericDefault_USB");

    if (!setGenericDef) {
        qDebug() << "Failed to resolve qrcodeUsb function from CLR DLL";
        return;
    }

    setGenericDef();
}

// 设定 sensor 设定初始化
void sensorDefaultUsb(void)
{
    sensorDefault_USB setSensorDefUsb =
(sensorDefault_USB)GetProcAddress(g_hModu_usb,"sensorDefault_USB");

    if (!setSensorDefUsb) {
        qDebug() << "Failed to resolve qrcodeUsb function from CLR DLL";
        return;
    }

    setSensorDefUsb();
}

// 设定 wifi 频段
void setWifiFrequencyUsb(const std::string& frequency)
{
    WifiFrequency_USB wifiFrequencyUsb = (WifiFrequency_USB)GetProcAddress(g_hModu_usb,
"WifiFrequency_USB");
    if (!wifiFrequencyUsb) {
        qDebug() << "Failed to resolve WifiFrequency_USB function from CLR DLL";
        return;
    }
}

```

```

const char* frequencyChar = frequency.c_str();

wifiFrequencyUsb(const_cast<char*>(frequencyChar));
}

// 直接打印 bitmap
int BitmapUsb(const std::string& x, const std::string& y, int width, int height, int mode, const std::string&
filename)
{

    Bitmap_USB bitmapUsb = (Bitmap_USB)GetProcAddress(g_hModu_usb, "Bitmap_USB");
    if (!bitmapUsb) {
        qDebug() << "Failed to resolve Bitmap_USB function from CLR DLL";
        return 0;
    }

    const char* xChar = x.c_str();
    const char* yChar = y.c_str();
    const char* filenameChar = filename.c_str();

    int result = bitmapUsb(const_cast<char*>(xChar), const_cast<char*>(yChar), width, height, mode,
const_cast<char*>(filenameChar));
    return result;
}

// 直接打印压缩的 bitmap
int CompressBitmapUsb(const std::string& x, const std::string& y, int width, int height, const std::string&
filename)
{

    compressBitmap_USB bitmapUsb = (compressBitmap_USB)GetProcAddress(g_hModu_usb,
"compressBitmap_USB");
    if (!bitmapUsb) {
        qDebug() << "Failed to resolve Bitmap_USB function from CLR DLL";
        return 0;
    }

```

```

}

const char* xChar = x.c_str();
const char* yChar = y.c_str();
const char* filenameChar = filename.c_str();

int result = bitmapUsb(const_cast<char*>(xChar), const_cast<char*>(yChar), width,
height,const_cast<char*>(filenameChar));
return result;
}

void sendAnnotationUsb(void)
{
    sendcommandUsb(TSPL_ANNOTATION_COMMAND);
}
////===== 实际调用范本 =====
void usbSettingTest(HMODULE hModu)
{
    g_hModu_usb = hModu;
    int result = 0;

    result=openPortUsb();
    if(result)
    {

        // 设定方向与镜像
        sendAnnotationUsb();
        setDirectionAndMirroUsb(0,0);
        sendAnnotationUsb();
        setDirectionAndMirroUsb(1,0);
        sendAnnotationUsb();
        setDirectionAndMirroUsb(0,1);
        sendAnnotationUsb();
        setDirectionAndMirroUsb(1,1);

        // 设定 Shift
        sendAnnotationUsb();
    }
}

```

```
setShiftUsb(180);
sendAnnotationUsb();
setShiftUsb(50);

// 设定 AfterPrintAction
sendAnnotationUsb();
setAfterPrintModeUsb(0);
sendAnnotationUsb();
setAfterPrintModeUsb(1);
sendAnnotationUsb();
setAfterPrintModeUsb(2);
sendAnnotationUsb();
setAfterPrintModeUsb(3);

// 设定 Offset
sendAnnotationUsb();
setOffsetUsb(50);
sendAnnotationUsb();
setOffsetUsb(5);

// 设定切刀
sendAnnotationUsb();
setCutModeUsb(0, 2);
sendAnnotationUsb();
setCutModeUsb(1, 5);

// 设定 genetic 初始化
sendAnnotationUsb();
genericDefaultUsb();

// 设定 WIFI 频段
sendAnnotationUsb();
setWifiFrequencyUsb("5G");
sendAnnotationUsb();
setWifiFrequencyUsb("2.4G");

// 设定 Shift
sendAnnotationUsb();
setShiftUsb(80);
```

```

        sendAnnotationUsb();
        setShiftUsb(50);
    }
}

void usbSample(HMODULE hModu)
{
    g_hModu_usb = hModu;
    int result = 0;

    result=openPortUsb();
    if(result)
    {
        // 取得打印机信息
        char* printerStatus = printerStatusUsb();

        // 设定打印机尺寸
        clearBufferUsb();
        setupUsb("104", "76", "2", "7", "0", "3", "0");
        sendcommandUsb("DIRECTION 1");

        // 印一张中文字
        printFontUnicodeUsb("50", "10", "TSS24.BF2", "0", "1", "1", "USB 接口默认简体中文测试：海天
米醋白米醋 1 瓶 450ml");
        printLabelUsb("1", "1");

        // 印一张 barcode
        barcode_Usb("30", "30", "128", "100", "1", "0", "2", "2", "barcode1234567");
        printLabelUsb("1", "1");

        // 印一张文字
        clearBufferUsb();
        printFontUsb("50", "10", "TSS24.BF2", "0", "1", "1", "testFont12345678");
        printLabelUsb("1", "1");

        // 印一张 PCX 图片
        QString currentDir = QDir::currentPath();
        std::string filenameStr = (currentDir + "\\UL.PCX").toStdString();
        clearBufferUsb();
    }
}

```

```
downloadPcxUsb(filenameStr, "UL.PCX");
sendcommandUsb("PUTPCX 50,10,\"UL.PCX\"\\r\\n");
printLabelUsb("1", "1");
```

```
// 印一张 BMP
```

```
currentDir = QDir::currentPath();
filenameStr = (currentDir + "\\CIRCLE.BMP").toStdString();
clearBufferUsb();
downloadBmpUsb(filenameStr, "CIRCLE.BMP");
sendcommandUsb("PUTBMP 10,10,\"CIRCLE.BMP\"\\r\\n");
printLabelUsb("1", "1");
```

```
//印一张 windows font
```

```
currentDir = QDir::currentPath();
clearBufferUsb();
filenameStr = (currentDir + "\\impact.ttf").toStdString();
downloadBmpUsb(filenameStr, "impact.ttf");
windowsFontUsb(10, 100, 48, 0, 0, 0, "impact", "C# WIN TEST");
printLabelUsb("1", "1");
```

```
// 印一张 block text
```

```
clearBufferUsb();
printFontblockUsb("35", "15", "300", "90", "TSS24.BF2", "0", "1", "1", "0", "1",
"PrintFontBlockTestPrintFontBlockTestPrintFontBlockTestPrintFontBlockTest");
printLabelUsb("1", "1");
```

```
// 印一张 qrcode
```

```
qrcodeUsb("220", "260", "M", "3", "A", "0", "AABCB03abcN123");
printLabelUsb("1", "1");
```

```
// 设定 Reverse
```

```
sendcommandUsb("TEXT 100,100,\"3\",0,1,1,\"REVERSE\"");
printReverseUsb(90, 90, 128, 40);
printLabelUsb("1", "1");
```

```
// 直接打印 BMP
```

```
if(BitmapUsb("0", "0", 400, 350, 1, "Thunder.png")!=0)
    printLabelUsb("1", "1");
```

```
// 直接打印压缩 BMP
if(CompressBitmapUsb("0", "0", 666, 357, "Cat1.jpg")!=0)
    printLabelUsb("1", "1");

closeportUSB();
}
}
```


● Jsp

1.需先汇入 jan-5.5.0.jar:

```
import com.sun.jna.Library;
```

```
import com.sun.jna.Native;
```

2.建立调用 dll 的 class 调用 dll

```
class GTSPL {

    interface GTSPL_SDK extends Library {

        GTSPL_SDK INSTANCE = (GTSPL_SDK) Native.load("GTSPL_SDK_C", GTSPL_SDK.class);

        int openport_USB();

        int openport(String PrinterName);

        ....

    }

}
```

3.范例程序:

```
System.load(System.getProperty("user.dir") + "\\GTSPL_SDK_C.dll");

//单机打印

GTSPL.GTSPL_SDK.INSTANCE.openport("Printer Model");

GTSPL.GTSPL_SDK.INSTANCE.setup("54", "30", "2", "3", "0", "3", "0");

GTSPL.GTSPL_SDK.INSTANCE.sendcommand("DIRECTION 1");

GTSPL.GTSPL_SDK.INSTANCE.setDirectionAndMirror(1, 1);

GTSPL.GTSPL_SDK.INSTANCE.setShift(50);

GTSPL.GTSPL_SDK.INSTANCE.setAfterPrintAction(2);

GTSPL.GTSPL_SDK.INSTANCE.setOffset(20);

GTSPL.GTSPL_SDK.INSTANCE.setCutMode(0,2);

GTSPL.GTSPL_SDK.INSTANCE.clearbuffer();

GTSPL.GTSPL_SDK.INSTANCE.sendcommand("TEXT 100,100,\"3\",0,1,1,\"REVERSE\");

GTSPL.GTSPL_SDK.INSTANCE.printReverse(90, 90, 128, 40);
```

```

GTSP.LGTSP.L_SDK.INSTANCE.barcode("30", "30", "128", "100", "1", "0", "2", "2",
"barcode1234567");

GTSP.LGTSP.L_SDK.INSTANCE.qrcode("50", "100", "H", "4", "A", "0", "QRcode1234567");

GTSP.LGTSP.L_SDK.INSTANCE.printerfont("50", "10", "2", "0", "1", "1", "Print Font 123456");

GTSP.LGTSP.L_SDK.INSTANCE.downloadbmp(System.getProperty("user.dir") + "\\CIRCLE.BMP",
"CIRCLE.BMP");

GTSP.LGTSP.L_SDK.INSTANCE.windowfont(5, 150, 48, 0, 0, 0, "arial", "JAVA WIN DRIVER");

GTSP.LGTSP.L_SDK.INSTANCE.sendcommand("PUTBMP 150,30,\"CIRCLE.BMP\"");

GTSP.LGTSP.L_SDK.INSTANCE.printerfontblock("15", "15", "790", "90", "0", "0", "8", "8", "20",
"2", "We stand behind our products with one of the most comprehensive support programs in the
Auto-ID industry.");

```

//初始化

```

GTSP.LGTSP.L_SDK.INSTANCE.genericDefault();

GTSP.LGTSP.L_SDK.INSTANCE.sensorDefault();

GTSP.LGTSP.L_SDK.INSTANCE.rfidSetupDefault();

```

//BLOCK 打印

```

WString str = new WString("-- 默认简体中文测试：海天米醋白米醋 1 瓶 450ml --");

GTSP.LGTSP.L_SDK.INSTANCE.printerfontblockUnicode("15", "15", "790", "90", "TSS24.BF2", "0", "1",
"1", "0", "1", str);

GTSP.LGTSP.L_SDK.INSTANCE.printlabel("1", "1");

GTSP.LGTSP.L_SDK.INSTANCE.closeport();

```

//简中打印

WString str=new WString("默认简体中文测试"); //需 jni 的 WString 才能正确传送中文

```

GTSP.LGTSP.L_SDK.INSTANCE.clearbuffer();

GTSP.LGTSP.L_SDK.INSTANCE.printerfontUnicode("100", "10", "TSS24.BF2", "0", "1", "1", str);

GTSP.LGTSP.L_SDK.INSTANCE.printlabel(1, 1);

//繁中打印

WString str=new Wstring("默認繁體中文測試");//需 jni 的 WString 才能正确传送中文

GTSP.LGTSP.L_SDK.INSTANCE.clearbuffer();

GTSP.LGTSP.L_SDK.INSTANCE. printerfontUnicode("100", "10", " TST24.BF2", "0", "1", "1", str);

GTSP.LGTSP.L_SDK.INSTANCE.printlabel(1, 1);

GTSP.LGTSP.L_SDK.INSTANCE.closeport();

GTSP.LGTSP.L_SDK.INSTANCE.getDLLVersion(0);


//指定 USB 传输接口

//侦测多台打印机

String PrinterName = GTSP.LGTSP.L_SDK.INSTANCE.detectUSBStr_USB();

String[] PrinterNameStringArray = PrinterName.split("\\n+");

GTSP.LGTSP.L_SDK.INSTANCE.openport_USB("Printer Model");

//单机打印

GTSP.LGTSP.L_SDK.INSTANCE.openport_USB();

GTSP.LGTSP.L_SDK.INSTANCE.setup_USB("54", "30", "2", "3", "0", "3", "0");

GTSP.LGTSP.L_SDK.INSTANCE.sendcommand_USB("DIRECTION 1");

GTSP.LGTSP.L_SDK.INSTANCE.setDirectionAndMirror_USB(1, 1);

GTSP.LGTSP.L_SDK.INSTANCE.setShift_USB(50);

GTSP.LGTSP.L_SDK.INSTANCE.setAfterPrintAction_USB(2);

GTSP.LGTSP.L_SDK.INSTANCE.setOffset_USB(20);

GTSP.LGTSP.L_SDK.INSTANCE.setCutMode_USB(0,2);

GTSP.LGTSP.L_SDK.INSTANCE.clearbuffer_USB();

```

```

GTSP.LGTSP.L_SDK.INSTANCE.barcode_USB("30", "30", "128", "100", "1", "0", "2", "2",
"barcode1234567");

GTSP.LGTSP.L_SDK.INSTANCE.qrcode_USB("50", "100", "H", "4", "A", "0", "QRcode1234567");

GTSP.LGTSP.L_SDK.INSTANCE.printerfont_USB("50", "10", "2", "0", "1", "1", "Print Font 123456");

GTSP.LGTSP.L_SDK.INSTANCE.windowfont_USB(5, 150, 48, 0, 0, 0, "arial", "JAVA WIN DRIVER");

GTSP.LGTSP.L_SDK.INSTANCE.downloadpcx_USB(System.getProperty("user.dir")+ "\\UL.PCX",
"UL.PCX");

GTSP.LGTSP.L_SDK.INSTANCE.sendcommand_USB("PUTPCX 50,10,\"UL.PCX\"");

GTSP.LGTSP.L_SDK.INSTANCE.printerfontblock_USB("15", "15", "790", "90", "0", "0", "8", "8", "20",
"2","We stand behind our products with one of the most comprehensive support programs in the
Auto-ID industry.");

```

//初始化

```

GTSP.LGTSP.L_SDK.INSTANCE.genericDefault_USB();

GTSP.LGTSP.L_SDK.INSTANCE.sensorDefault_USB();

GTSP.LGTSP.L_SDK.INSTANCE.rfidSetupDefault_USB();

```

```
String status = GTSP.LGTSP.L_SDK.INSTANCE.printerstatus_USB();
```

//简中打印

WString str=new WString("默认简体中文测试"); //需 jni 的 WString 才能正确传送中文

```

GTSP.LGTSP.L_SDK.INSTANCE.clearbuffer_USB();

GTSP.LGTSP.L_SDK.INSTANCE.printerfontUnicode_USB ("100", "10", "TSS24.BF2", "0", "1", "1", str);

GTSP.LGTSP.L_SDK.INSTANCE.printlabel_USB (1, 1);

```

//繁中打印

WString str=new Wstring("默認繁體中文測試");//需 jni 的 WString 才能正确传送中文

```

GTSP.LGTSP.L_SDK.INSTANCE.clearbuffer_USB();

GTSP.LGTSP.L_SDK.INSTANCE.printerfontUnicode_USB ("100", "10", " TST24.BF2", "0", "1", "1", str);

GTSP.LGTSP.L_SDK.INSTANCE.printlabel_USB(1, 1);

GTSP.LGTSP.L_SDK.INSTANCE.closeport_USB();

GTSP.LGTSP.L_SDK.INSTANCE.getDLLVersion_USB(0);

```

//指定 Ethernet 传输接口

```

GTSP.LGTSP.L_SDK.INSTANCE.openport_Ethernet(IP,Port);

GTSP.LGTSP.L_SDK.INSTANCE.setup_Ethernet("54", "30", "2", "3", "0", "3", "0");

GTSP.LGTSP.L_SDK.INSTANCE.sendcommand_Ethernet("DIRECTION 1");

GTSP.LGTSP.L_SDK.INSTANCE.setOffset_Ethernet(20);

GTSP.LGTSP.L_SDK.INSTANCE.setCutMode_Ethernet(0,2);

GTSP.LGTSP.L_SDK.INSTANCE.setDirectionAndMirror_Ethernet(1, 1);

GTSP.LGTSP.L_SDK.INSTANCE.setShift_Ethernet(50);

GTSP.LGTSP.L_SDK.INSTANCE.setAfterPrintAction_Ethernet(2);

GTSP.LGTSP.L_SDK.INSTANCE.clearbuffer_Ethernet();

GTSP.LGTSP.L_SDK.INSTANCE.sendcommand_Ethernet("TEXT 100,100,\"3\",0,1,1,\"REVERSE\");

GTSP.LGTSP.L_SDK.INSTANCE.printReverse_Ethernet(90, 90, 128, 40);

GTSP.LGTSP.L_SDK.INSTANCE.barcode_Ethernet("30", "30", "128", "100", "1", "0", "2", "2",
"barcode1234567");

GTSP.LGTSP.L_SDK.INSTANCE.qrcode_Ethernet("50", "100", "H", "4", "A", "0", "QRcode1234567");

GTSP.LGTSP.L_SDK.INSTANCE.printerfont_Ethernet("50", "10", "2", "0", "1", "1", "Print Font
123456");

GTSP.LGTSP.L_SDK.INSTANCE.windowfont_Ethernet(5, 150, 48, 0, 0, 0, "arial", "JAVA WIN DRIVER");

GTSP.LGTSP.L_SDK.INSTANCE.downloadpcx_Ethernet(System.getProperty("user.dir")+ "\\UL.PCX",

```

```
"UL.PCX");
```

```
GTSP.LGTSP.L_SDK.INSTANCE.sendcommand_Ethernet("PUTPCX 50,10,\"UL.PCX\\");
```

```
GTSP.LGTSP.L_SDK.INSTANCE.printerfontblock_Ethernet("15", "15", "790", "90", "0", "0", "8", "8",
"20", "2", "We stand behind our products with one of the most comprehensive support programs in
the Auto-ID industry.");
```

```
//初始化
```

```
GTSP.LGTSP.L_SDK.INSTANCE.genericDefault_Ethernet();
```

```
GTSP.LGTSP.L_SDK.INSTANCE.sensorDefault_Ethernet();
```

```
GTSP.LGTSP.L_SDK.INSTANCE.rfidSetupDefault_Ethernet();
```

```
String status = GTSP.LGTSP.L_SDK.INSTANCE.printerstatus_Ethernet();
```

```
//簡中打印
```

```
WString str=new WString("默认简体中文测试");//需 jni 的 WString 才能正確傳送中文
```

```
GTSP.LGTSP.L_SDK.INSTANCE.clearbuffer_Ethernet();
```

```
GTSP.LGTSP.L_SDK.INSTANCE.printerfontUnicode_Ethernet("100", "10", "TSS24.BF2", "0", "1", "1",
str);
```

```
GTSP.LGTSP.L_SDK.INSTANCE.printlabel_Ethernet(1, 1);
```

```
//繁中打印
```

```
WString str=new Wstring("默認繁體中文測試");//需 jni 的 WString 才能正確傳送中文
```

```
GTSP.LGTSP.L_SDK.INSTANCE.clearbuffer_Ethernet();
```

```
GTSP.LGTSP.L_SDK.INSTANCE. printerfontUnicode_Ethernet("100", "10", " TST24.BF2", "0", "1", "1",
str);
```

```
GTSP.LGTSP.L_SDK.INSTANCE.printlabel_Ethernet(1, 1);
```

```
GTSP.LGTSP.L_SDK.INSTANCE.closeport_Ethernet();
```

```
GTSP.LGTSP.L_SDK.INSTANCE.getDLLVersion_Ethernet(0);
```

4.DLL 放置位置:

〔 利用 IDE 开发时 〕

GTSP.L_SDK.dll 放在 jdk 的 bin 文件夹下

如: C:\Program Files\Java\jdk1.8.0_221\bin

GTSP.L_SDK_C.dll 放在 java 项目文件夹下

〔 未安装 jdk 直接执行.jar 档时 〕

GTSP.L_SDK.dll 放在 jre 的 bin 文件夹下

如: C:\Program Files\Java\jre1.8.0_251\bin

GTSP.L_SDK_C.dll 放在.jar 档相同路径下

.jar 档相同路径下, 还需要有 lib 文件夹, 且里面要有 jna-5.5.0.jar

5.Driver 模式下, 需要安装打印机的驱动才能执行。

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					
EAN128M	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		
PLESSEY	PLESSEY.		V	V	V		
CPOST	China post.					V	
ITF14	ITF14.		V	V	V		13
EAN14	EAN14.	V					13
11	Code 11.		V	V	V		

TELEPEN	Telepen. *Since V6.89EZ.		V	V	V		
TELEPENN	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
LOGMARS	A special use of Code 39. *Since V6.88EZ.		V	V	V		