

---

# POSTEK PPLE

## API Function Reference

Barcode label printer

Version 4.0.0

Shenzhen POSTEK Technology Development Co., Ltd.

2024

---

## API library file description

Name: CDFPSK.dll

Version: 4.0.0

Copyright (c) 2024 Shenzhen POSTEK Technology Development Co., Ltd. All rights reserved.

## Purpose

This API library provides a set of commands for users of Shenzhen POSTEK Technology Development Co., Ltd. barcode label printers, making it easier to write applications for the Windows 9X, NT, 2000, XP, Windows 7, Windows 8 and similar operating systems.

This API library only supports POSTEK products.

## Abbreviations

PPL: the first printer programming language developed by Shenzhen POSTEK Technology Development Co., Ltd. (Printer Program Language E).

API: Application Programming Interface.

Dots: a pixel is a unit of measurement in computer science. Originally the smallest unit of a television image, in the printer field it refers to the smallest imaging unit a printer can produce: 1 dot equals one inch divided by the printer's maximum resolution.

- For a 203 DPI printer,  $1 \text{ dot} = 25.4 \text{ mm} / 203 \approx 0.125 \text{ mm}$  ( $1 \text{ dot} = 1000 / 203 \approx 5 \text{ mil}$ );
- For a 300 DPI printer,  $1 \text{ dot} = 25.4 \text{ mm} / 300 \approx 0.085 \text{ mm}$  ( $1 \text{ dot} = 1000 / 300 \approx 3 \text{ mil}$ );
- For a 600 DPI printer,  $1 \text{ dot} = 25.4 \text{ mm} / 600 \approx 0.042 \text{ mm}$  ( $1 \text{ dot} = 1000 / 600 \approx 2 \text{ mil}$ ).

Integer conversions used in this manual:

- For a 203 DPI printer,  $1 \text{ mm} = 8 \text{ dots}$ .
- For a 300 DPI printer,  $1 \text{ mm} = 12 \text{ dots}$ .
- For a 600 DPI printer,  $1 \text{ mm} = 24 \text{ dots}$ .

TrueType Font: a removable font designed for use with the Windows operating system.

- Any installed TrueType Font can be used by this function.

---

## Before You Begin

### Port

- 1, Before sending any commands to the printer, a port must be opened; otherwise all API calls that operate on the printer will fail.
- 2, Only one port can be open at a time; the current port must be closed before another can be opened.

### CDFPSK.ini configuration file description

INI-file configuration feature. The INI file is placed in the DLL's directory; the available settings are listed below.

logMode=0 Disable the logging feature.

logMode=1 Enable the logging feature and write the log in the DLL's directory. // If not configured, logging is disabled by default.

tph=2 Set the printer resolution to 200 dpi.

tph=3 Set the printer resolution to 300 dpi. // If not configured, the default is 300 dpi.

Currently used to enable center and right alignment of the human-readable text below a Code128 auto barcode.

Lang=CN Return error messages from the DLL in Chinese.

Lang=EN Return error messages from the DLL in English. // If not configured, the DLL detects the OS UI language automatically: Chinese systems get Chinese, all others get English.

The following are only supported by the multi-byte version of the DLL:

textEncode=0 Strings passed to the functions are interpreted as GBK.

textEncode=1 Strings passed to the functions are interpreted as UTF-8. // Default is UTF-8 if not configured.

Takes effect for strings passed to PTK\_DrawText, PTK\_DrawTextEx, PTK\_DrawText\_TrueType and PTK\_DrawText\_TrueType.

### String

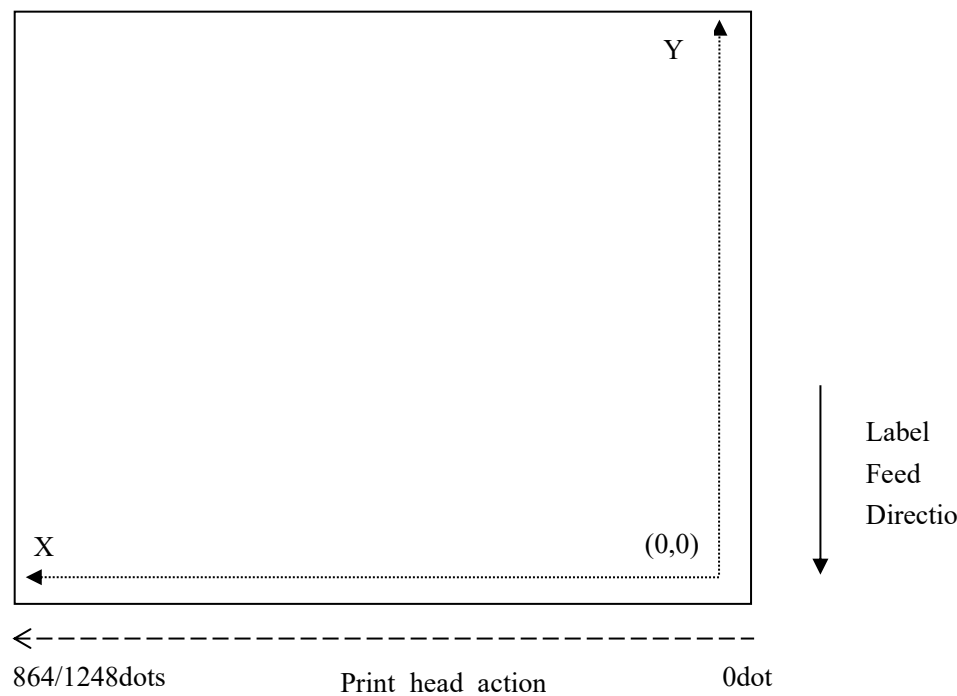
- \* Strings use the double quote (") as the start and end markers;
- \* All print commands and names are case sensitive;
- \* <CR> is decimal "13" (or hexadecimal "0DH") in US-ASCII, i.e. the "carriage return" character;
- \* The backslash (\) has the following uses:

---

<u>Character</u>	<u>Input</u>
"	\"
\	\\
0x00 - 0x7F	\x00 - \x7F

Coordinate system of a barcode label printer

As shown in the figure below:



---

## Function Overview

### Logging feature

Function Name	Description
<a href="#">PTK_GetErrorInfo</a>	Used to parse error codes.
<a href="#">PTK_OpenLogMode</a>	Enables the logging feature and specifies the path where logs are written.
<a href="#">PTK_CloseLogMode</a>	Disables the logging feature.

Note: If both the INI file and an API function are used to enable/disable logging, the API function call takes precedence.

### Port operations

Function Name	Description
<a href="#">PTK_GetAllPrinterUSBInfo</a>	Retrieves USB port information for all currently USB-connected printers.
<a href="#">PTK_OpenUSBPort</a>	Opens a USB port
<a href="#">PTK_CloseUSBPort</a>	Close the opened USB port
<a href="#">PTK_OpenSerialPort</a>	Opens a serial port
<a href="#">PTK_CloseSerialPort</a>	Close the opened serial port
<a href="#">PTK_OpenPrinter</a>	Opens a printer driver
<a href="#">PTK_ClosePrinter</a>	Close the opened printer driver
<a href="#">PTK_OpenParallelPort</a>	Opens a parallel port
<a href="#">PTK_CloseParallelPort</a>	Close the opened parallel port
<a href="#">PTK_OpenTextPort</a>	Creates a file into which data sent to the printer will be written.
<a href="#">PTK_CloseTextPort</a>	Close the file that was opened.
<a href="#">PTK_Connect</a>	Connects to a printer over the network port
<a href="#">PTK_Connect_Timer</a>	Connects to a printer over the network port, with a configurable connection timeout.
<a href="#">PTK_CloseConnect</a>	Closes the network connection to the printer.
<a href="#">PTK_OpenUSBPort_Buff</a>	Creates a buffer into which data sent to the printer will be written, and opens a USB port.
<a href="#">PTK_Connect_Timer_Buff</a>	Creates a buffer into which data sent to the printer will be written, and opens a network port.
<a href="#">PTK_OpenPrinter_Buff</a>	Creates a buffer into which data sent to the printer will be written, and opens a driver port.
<a href="#">PTK_CloseBuffPort</a>	Frees the buffer and closes the open port.
<a href="#">PTK_WriteBuffToPrinter</a>	Sends the contents of the buffer to the open port.

<a href="#">PTK_SendFile</a>	Sends the contents of a file to the printer
------------------------------	---

## Printer settings

Function Name	Description
<a href="#">PTK_SetCharSets</a>	Sets the character set.
<a href="#">PTK_PrintConfiguration</a>	Instruct the printer to print the self-test page.
<a href="#">PTK_MediaDetect</a>	Instruct the printer to perform media calibration.
<a href="#">PTK_FeedMedia</a>	Instruct the printer to feed one label.
<a href="#">PTK_UserFeed</a>	Instruct the printer to feed a fixed length of media forward.
<a href="#">PTK_UserBackFeed</a>	Instruct the printer to feed a fixed length of media backward.
<a href="#">PTK_EnableFLASH</a>	Instruct the printer to enable FLASH storage; subsequent calls to storage-related API functions will store data in the printer's FLASH.
<a href="#">PTK_DisableFLASH</a>	Disable FLASH storage
<a href="#">PTK_CutPage</a>	Sets the cut frequency. Preserved across power cycles.
<a href="#">PTK_CutPageEx</a>	Sets the cut frequency. Not preserved across power cycles.
<a href="#">PTK_SetCoordinateOrigin</a>	Sets the origin of the print coordinate system.
<a href="#">PTK_GetUtilityInfo</a>	Retrieves the commonly used configuration information saved in the printer's FLASH.
<a href="#">PTK_SetUtilityInfoProc</a>	Configures the printer's commonly used information.
<a href="#">PTK_SetUtilityInfo</a>	Sends commonly used printer configuration information to the printer's FLASH to be saved.
<a href="#">PTK_GetAllPrinterInfo</a>	Retrieves all configuration information saved in the printer's FLASH.
<a href="#">PTK_SetAllPrinterInfo</a>	Sets all printer information and saves it to FLASH.
<a href="#">PTK_ErrorReport_USBInterrupt</a>	Retrieves the printer's current status in real time via a USB interrupt.
<a href="#">PTK_GetPrinterStatus_Net</a>	Retrieves the printer's current status over the network. Only supported on non-touchscreen printers.
<a href="#">PTK_GetPrinterStatus_Net_OX</a>	Retrieves the printer's current status over the network. Only supported on OX-series printers.
<a href="#">PTK_GetLabelPrintInfo</a>	Retrieves label print information, remaining media and the key action after a print error.

## Label settings

Function Name	Description
<a href="#">PTK_ClearBuffer</a>	Clears the contents of the printer's buffer.
<a href="#">PTK_SetPrintSpeed</a>	Sets the current print speed. Not preserved across power cycles.
<a href="#">PTK_SetDarkness</a>	Sets the current print darkness. Not preserved across power cycles.

<a href="#">PTK_SetDirection</a>	Sets the current print direction for the label. Not preserved across power cycles.
<a href="#">PTK_SetLabelHeight</a>	Sets the current label height, the height of the positioning gap / black line / notch, and the positioning offset. Not preserved across power cycles.
<a href="#">PTK_SetLabelWidth</a>	Sets the current label width. Not preserved across power cycles.
<a href="#">PTK_PrintLabel</a>	Instruct the printer to start printing the label content.
<a href="#">PTK_PrintLabelFeedback</a>	Instruct the printer to start printing the label content, then read the printer's current status after printing completes.

## Prints text

Function Name	Description
<a href="#">PTK_DrawText</a>	Prints one line of text
<a href="#">PTK_DrawText_MultiLine</a>	Prints one line of text using the printer's built-in fonts; supports automatic line wrapping.
<a href="#">PTK_DrawTextEx</a>	Prints one line of text, serial number or variable (use together with the form-printing functions).
<a href="#">PTK_DrawText_TrueType</a>	Prints one line of TrueType-font text using the Windows font library.
<a href="#">PTK_DrawText_TrueType_MultiLine</a>	Prints one line of TrueType-font text using the Windows font library; supports automatic line wrapping.
<a href="#">PTK_RenameDownloadFont</a>	Maps a font downloaded to the printer to one of the A-Z font IDs used by PTK_DrawText.

## Prints an image

Function Name	Description
<a href="#">PTK_ChangeIMGtoPCX</a>	Converts an image to PCX format and generates a PCX file with the same name in the same directory.
<a href="#">PTK_PcxGraphicsList</a>	Prints a list of the names of the images stored in the printer's RAM or FLASH memory.
<a href="#">PTK_PcxGraphicsDel</a>	Deletes an image stored on the printer.
<a href="#">PTK_AnyGraphicsDownload</a>	Downloads an image to the printer from a file path.
<a href="#">PTK_DrawPcxGraphics</a>	Prints an image stored on the printer
<a href="#">PTK_AnyGraphicsPrint</a>	Prints an image directly from a file path.
<a href="#">PTK_AnyGraphicsDownloadFromMemory</a>	Stores an image on the printer from image data.
<a href="#">PTK_AnyGraphicsPrintFromMemory</a>	Prints an image from image data.
<a href="#">PTK_AnyGraphicsPrint_Base64</a>	Prints an image from base64-encoded image data.

---

## Prints a dot-matrix image

Function Name	Description
<a href="#">PTK_BinGraphicsList</a>	Prints a list of the names of the images stored in the printer's RAM or FLASH memory.
<a href="#">PTK_BinGraphicsDel</a>	Deletes a binary image stored on the printer.
<a href="#">PTK_BinGraphicsDownload</a>	Stores a binary-format image to the printer
<a href="#">PTK_RecallBinGraphics</a>	Prints a binary image stored on the printer
<a href="#">PTK_DrawBinGraphics</a>	Directly defines and prints a binary image.

## Prints a line

Function Name	Description
<a href="#">PTK_DrawRectangle</a>	Draws an empty rectangle on the label
<a href="#">PTK_DrawLineXor</a>	Draws a straight line, applying an XOR operation where it intersects existing content
<a href="#">PTK_DrawLineOr</a>	Draws a straight line, applying an OR operation where it intersects existing content
<a href="#">PTK_DrawDiagonal</a>	Draws a diagonal line, applying an OR operation where it intersects existing content
<a href="#">PTK_DrawWhiteLine</a>	Draws a white straight line

## Prints a 2D barcode

Function Name	Description
<a href="#">PTK_DrawBar2D_QR</a>	Prints a QR code
<a href="#">PTK_DrawBar2D_QREx</a>	Prints a QR code as an image and saves it as an image on the printer to be printed. Supports legacy firmware.
<a href="#">PTK_DrawBar2D_HANXIN</a>	Prints a Han Xin code
<a href="#">PTK_DrawBar2D_Pdf417</a>	Prints a PDF417 code
<a href="#">PTK_DrawBar2D_Pdf417Ex</a>	Prints a PDF417 barcode as an image. Supports legacy firmware.
<a href="#">PTK_DrawBar2D_MaxiCode</a>	Prints a MaxiCode
<a href="#">PTK_DrawBar2D_DATAMATRIX</a>	Prints a Data Matrix code



---

Prints a 1D barcode

Function Name	Description
<a href="#">PTK_DrawBarcode</a>	Prints various types of 1D barcodes, where the printed content is a constant string.
<a href="#">PTK_DrawBarcodeEx</a>	Prints various types of 1D barcodes, where the printed content can be a constant string, a serial number or a variable.

---

## Print form and related functions

Function Name	Description
<a href="#">PTK_GetStorageList</a>	Retrieves the names of the forms, fonts or images stored in FLASH.
<a href="#">PTK_FormList</a>	Prints a list of the names of the forms stored in the printer's RAM or FLASH memory.
<a href="#">PTK_FormDel</a>	Deletes a form stored on the printer.
<a href="#">PTK_FormDownload</a>	Tells the printer to start storing a form.
<a href="#">PTK_FormEnd</a>	Tells the printer that form storage is complete.
<a href="#">PTK_ExecForm</a>	Runs a form. Equivalent to executing every API call in the form once.
<a href="#">PTK_DefineVariable</a>	Defines a variable on the printer.
<a href="#">PTK_DefineCounter</a>	Defines a serial number on the printer.
<a href="#">PTK_Download</a>	Tell the printer to start assigning initial values to variables or serial numbers.
<a href="#">PTK_DownloadInitVar</a>	Initializes variables or serial numbers in the order in which they were defined.
<a href="#">PTK_PrintLabelAuto</a>	Instruct the printer to start printing labels; used inside a form, in combination with serial numbers or variables.
<a href="#">PTK_FormPrinting</a>	Print form

---

## RFID tag read/write functions

Function Name	Description
<a href="#">PTK_RFIDCalibrate</a>	RFID and HF tag probing calibration
<a href="#">PTK_RWRFIDLabel</a>	Write RFID tag data
<a href="#">PTK_RWRFIDLabelEx</a>	Write RFID tag data (not cleared)
<a href="#">PTK_SetRFLabelPWAndLockRFLabel</a>	Sets the RFID tag password and locks the RFID tag.
<a href="#">PTK_EncodeRFIDPC</a>	Write the RFID PC value or the Chinese national-standard encoding header
<a href="#">PTK_SetRFID</a>	Sets the RFID tag print parameters.
<a href="#">PTK_ReadRFIDLabelData</a>	Read RFID tag data
<a href="#">PTK_ReadRFIDLabelDataEx</a>	Read RFID tag data
<a href="#">PTK_RFIDEndPrintLabel</a>	Prints one label, then returns the RFID tag data that was printed.
<a href="#">PTK_RFIDEndPrintLabelFeedBack</a>	Prints one label, then returns the RFID tag data that was printed along with the printer status.
<a href="#">PTK_ReadRFIDSetting</a>	Returns data settings after printing a UHF RFID label.
<a href="#">PTK_PrintAndCallback</a>	Prints a UHF or HF RFID label and reports back the data.
<a href="#">PTK_SetReadRFIDForwardSpeed</a>	Sets the speed at which the label advances to the optimal read/write position when reading RFID data.
<a href="#">PTK_SetReadRFIDBackSpeed</a>	Sets the speed at which the label is rolled back to the print line when reading RFID data.

---

## HF tag read/write functions

Function Name	Description
<a href="#">PTK_RFIDCalibrate</a>	RFID and HF tag probing calibration
<a href="#">PTK_RWHFLabel</a>	Write HF tag data
<a href="#">PTK_SetHFRFID</a>	Sets HF tag information.
<a href="#">PTK_ReadHFLabelData</a>	Read HF tag data
<a href="#">PTK_ReadHFLabelUID</a>	Read HF tag UID
<a href="#">PTK_ReadHFRFIDSetting</a>	Returns data settings after printing an HF RFID label.
<a href="#">PTK_ReadHFTagDataPrintAuto</a>	During printing, reads the data from a specified block of the HF tag first, then prints it onto the label.
<a href="#">PTK_ReadHFTagUIDPrintAuto</a>	During printing, reads the HF tag UID first, then prints it onto the label.
<a href="#">PTK_SetHFAFI</a>	Sets the AFI value of the HF tag.
<a href="#">PTK_SetHFSFID</a>	Sets the DSFID value of the HF tag.
<a href="#">PTK_SetHFEAS</a>	Sets the EAS data.
<a href="#">PTK_HFDecrypt</a>	Decrypts an HF tag (only the 14443A protocol is supported).
<a href="#">PTK_LockHFLabel</a>	Locks an HF tag (only the 14443A protocol is supported).
<a href="#">PTK_LockHFIdentifier</a>	Locks the AFI / DSFID of a 15693 tag.
<a href="#">PTK_LockHFBlock</a>	Locks blocks on a 15693 / NTAG tag.
<a href="#">PTK_SetHFKey</a>	Sets the key.
<a href="#">PTK_SetHFCRCCommand</a>	Verifies, modifies and/or locks the CRC password.
<a href="#">PTK_SetHFPrivateCommand</a>	Verifies and modifies the private-mode password.
<a href="#">PTK_LockHFUser</a>	Locks the user area.
<a href="#">PTK_SetHFCFG10</a>	Sets the CFG Set 0x10 function parameter.
<a href="#">PTK_SetHFCFG80</a>	Sets the CFG Set 0x80 function parameter.

---

## Function Reference

### Logging feature

#### *PTK\_GetErrorInfo*

Description: parses an error code.

#### Prototype:

```
int _stdcall PTK_GetErrorInfo(int error_n, LPTSTR errorInfo, DWORD infoSize);
```

#### Parameters:

error\_n: the return value of any API function in this DLL.

errorInfo: buffer that stores the parsed string.

infoSize: size of the buffer allocated for errorInfo.

#### Return Value:

0:OK

-1: failed to parse the error code (no such error code).

4: infoSize is too small; errorInfo cannot hold the parsed information.

#### Example:

```
int nErrorcode = 0;
TCHAR buff[128] = { 0 };

PTK_OpenLogMode(_T("./log.txt"));
nErrorcode = PTK_OpenUSBPort(255);
PTK_GetErrorInfo(nErrorcode, buff, sizeof(buff));
MessageBox(buff);
PTK_PrintLabel(1, 1); // Print one label
PTK_CloseUSBPort ();
PTK_CloseLogMode();
```

#### *PTK\_OpenLogMode*

Description: enables the logging feature and specifies the path where the log file is created.

#### Note:

1. If this function is used and the log mode in CDFPSK.ini is also enabled, the log will be written to the path specified by this API function. However, if the log mode in CDFPSK.ini has not been disabled, CDFPSK\_log.txt will still be generated in the DLL's directory.
2. If this function is not used and the log mode in CDFPSK.ini is enabled, the log will be written to CDFPSK\_log.txt in the DLL's directory.

---

**Prototype:**

```
int _stdcall PTK_OpenLogMode(LPTSTR filePath);
```

**Parameters:**

filePath: the path where the generated log file will be written.

**Return Value:**

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

**Example:**

```
PTK_OpenLogMode(_T("./log.txt"));
PTK_OpenUSBPort(255);
PTK_PrintLabel(1, 1); // Print one label
PTK_CloseUSBPort();
PTK_CloseLogMode();
```

### *PTK\_CloseLogMode*

Description: disables the logging feature.

**Prototype:**

```
int _stdcall PTK_CloseLogMode(void);
```

**Parameters:**

None

**Return Value:**

0 -> OK

**Example:**

```
PTK_OpenLogMode(_T("./log.txt"));
PTK_OpenUSBPort(255);
PTK_PrintLabel(1, 1); // Print one label
PTK_CloseUSBPort();
PTK_CloseLogMode();
```

### *PTK\_GetLastError*

Description: retrieves the most recent error code from the Windows library functions.

---

**Prototype:**

```
int _stdcall PTK_GetLastError(void);
```

**Parameters:**

None

**Return Value:**

Return value of GetLastError()

Reference documents:

[https://docs.microsoft.com/en-us/previous-versions/aa911366\(v=msdn.10\)](https://docs.microsoft.com/en-us/previous-versions/aa911366(v=msdn.10))

<https://docs.microsoft.com/en-us/previous-versions/aa914935%28v%3dmsdn.10%29>

**Example:**

```
int errCode = PTK_GetLastError();
```

### *PTK\_WSAGetLastError*

Description: retrieves the most recent error code from the Windows networking library functions.

**Prototype:**

```
int _stdcall PTK_WSAGetLastError(void);
```

**Parameters:**

None

**Return Value:**

Return value of WSAGetLastError()

Reference documents:

[https://docs.microsoft.com/en-us/previous-versions/aa915624\(v=msdn.10\)](https://docs.microsoft.com/en-us/previous-versions/aa915624(v=msdn.10))

<https://docs.microsoft.com/en-us/previous-versions/aa924071%28v%3dmsdn.10%29>

**Example:**

```
int errCode = PTK_WSAGetLastError();
```

## Encoding conversion

### *PTK\_UTF8toGBK*

Description: converts a UTF-8 string to a GBK string.

**Prototype:**

```
int _stdcall PTK_UTF8toGBK(char *utf8Str, char *gbkStr, int gbkStrCount);
```

---

Parameters:

utf8Str: the UTF-8 string content to convert.  
gbkStr: buffer that stores the converted GBK string.  
gbkStrCount: maximum number of elements that gbkStr can hold.

Return Value:

0 -> OK  
For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
char str[16] = "POSTEK 2020";// On a Chinese Windows system the input string is in GBK encoding
char utf8[48] = { 0 };
char gbk[48] = { 0 };
PTK_GBKtoUTF8(str, utf8, 48);
PTK_UTF8toGBK(utf8, gbk, 48);
```

### *PTK\_GBKtoUTF8*

Description: converts a GBK string to a UTF-8 string.

Prototype:

```
int _stdcall PTK_GBKtoUTF8(char* gbkStr, char *utf8Str, int utf8StrCount);
```

Parameters:

gbkStr: the GBK string content to convert.  
utf8Str: buffer that stores the converted UTF-8 string.  
utf8StrCount: maximum number of elements that utf8Str can hold.

Return Value:

0 -> OK  
For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
char str[16] = "POSTEK 2020";// On a Chinese Windows system the input string is in GBK encoding
char utf8[48] = { 0 };
PTK_GBKtoUTF8(str, utf8, 48);
```

## Port operations

### *PTK\_GetAllPrinterUSBInfo*

Description: retrieves USB port information for all currently USB-connected printers.

Note: Only supports reading via USB. Connect the USB cable and turn on the printer before reading. This function



---

can be called without first opening a port.

**Prototype:**

```
int _stdcall PTK_GetAllPrinterUSBInfo(LPTSTR USBInfo, DWORD infoSize);
```

**Parameters:**

USBInfo: buffer used to store USB port information.

infoSize: size of the USBInfo buffer.

**Return Value:**

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

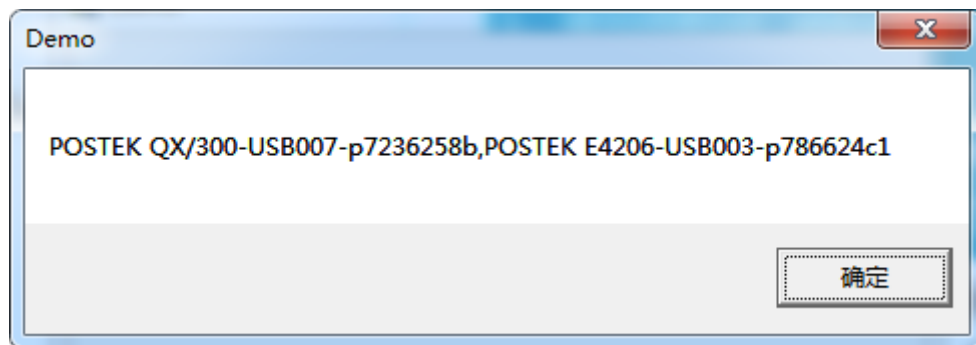
**Example:**

```
TCHAR buff[1024] = { 0 };
```

```
PTK_GetAllPrinterUSBInfo(buff, sizeof(buff));
```

```
MessageBox(buff);
```

Per-printer data format (entries for different printers are separated by commas): printer name - USB port number - USB ID.



*[PTK\\_OpenUSBPort](#)*

**Description:** opens a USB port.

**Prototype:**

```
int _stdcall PTK_OpenUSBPort(unsigned int port);
```

**Parameters:**

port: USB port number. Range: 1~255.

**Note:**

You can use PTK\_GetAllPrinterUSBInfo to inspect information about the USB port to which the printer is connected.

Passing 255 automatically opens the USB port of one connected printer. If multiple printers are connected, the one with the lowest USB ID is opened first.

---

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

**Example:**

```
PTK_OpenUSBPort(255);  
PTK_PrintLabel(1, 1); // Print one label  
PTK_CloseUSBPort ();
```

### *PTK\_CloseUSBPort*

Description: closes the opened USB port.

**Prototype:**

```
int _stdcall PTK_CloseUSBPort(void);
```

Parameters:

None

Return Value:

0 -> OK

**Example:**

```
PTK_OpenUSBPort(255);  
PTK_PrintLabel(1, 1); // Print one label  
PTK_CloseUSBPort ();
```

### *PTK\_OpenSerialPort*

Description: opens a serial port.

**Prototype:**

```
int _stdcall PTK_OpenSerialPort(unsigned int port, unsigned int bRate);
```

Parameters:

port: serial port number. Range: 1~255.

bRate: the printer's serial port baud rate. Valid values: 9600, 19200, 38400, 57600.

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

---

**Example:**

```
PTK_OpenSerialPort(28, 57600);  
PTK_PrintLabel(1, 1); // Print one label  
PTK_CloseSerialPort();
```

### *PTK\_CloseSerialPort*

Description: closes the opened serial port.

**Prototype:**

```
int _stdcall PTK_CloseSerialPort(void);
```

**Parameters:**

None

**Return Value:**

0 -> OK

**Example:**

```
PTK_OpenSerialPort(28, 57600);  
PTK_PrintLabel(1, 1); // Print one label  
PTK_CloseSerialPort();
```

### *PTK\_OpenPrinter*

Description: opens a printer driver.

Note: The driver port only supports sending data to the printer and cannot read data from it.

**Prototype:**

```
int _stdcall PTK_OpenPrinter(LPTSTR printerName);
```

**Parameters:**

printerName: printer name.

**Note:**

You can use PTK\_GetAllPrinterUSBInfo to inspect the printer name.

Alternatively, open "Devices and Printers" on the PC to view the printer driver name.

**Return Value:**

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

---

**Example:**

```
PTK_OpenPrinter(_T("POSTEK C168/300s"));
PTK_PrintLabel(1, 1); // Print one label
PTK_ClosePrinter();
```

*PTK\_ClosePrinter*

Description: closes the opened printer driver.

**Prototype:**

```
int _stdcall PTK_ClosePrinter(void);
```

Parameters:

None

Return Value:

0 -> OK

**Example:**

```
PTK_OpenPrinter(_T("POSTEK C168/300s"));
PTK_PrintLabel(1, 1); // Print one label
PTK_ClosePrinter();
```

*PTK\_OpenParallelPort*

Description: opens a parallel port.

**Prototype:**

```
int _stdcall PTK_OpenParallelPort(unsigned int port);
```

Parameters:

port: parallel port number. Range: 1~3.

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

**Example:**

```
PTK_OpenParallelPort(1)
```

---

```
PTK_PrintLabel(1, 1); // Print one label
PTK_CloseParallelPort();
```

### *PTK\_CloseParallelPort*

Description: closes the opened parallel port.

#### Prototype:

```
int _stdcall PTK_CloseParallelPort(void);
```

#### Parameters:

None

#### Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

#### Example:

```
PTK_OpenParallelPort(1)
PTK_PrintLabel(1, 1); // Print one label
PTK_CloseParallelPort();
```

### *PTK\_OpenTextPort*

Description: creates a file into which data sent to the printer will be written.

**Note: Each call recreates the file, overwriting any previous content.**

#### Prototype:

```
int _stdcall PTK_OpenTextPort(LPTSTR fn);
```

#### Parameters:

fn: path of the file to create.

#### Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

#### Example:

```
PTK_OpenTextPort(_T("command.txt"));
PTK_AnyGraphicsPrint(0, 0, _T("P1"), _T("Koala.jpg"), 1, 0, 0, 0);
PTK_PrintLabel(1, 1);
```

---

```
PTK_CloseTextPort();

PTK_OpenUSBPort(255);
PTK_SendFile(_T("command.txt"));
PTK_CloseUSBPort();
```

### *PTK\_CloseTextPort*

Description: closes the opened file.

#### Prototype:

```
int _stdcall PTK_CloseTextPort(void);
```

Parameters:

None

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

#### Example:

```
PTK_OpenTextPort(_T("command.txt"));
PTK_AnyGraphicsPrint(0, 0, _T("P1"), _T("Koala.jpg"), 1, 0, 0, 0);
PTK_PrintLabel(1, 1);
PTK_CloseTextPort();

PTK_OpenUSBPort(255);
PTK_SendFile(_T("command.txt"));
PTK_CloseUSBPort();
```

### *PTK\_Connect*

Description: connects to a printer over the network port.

#### Prototype:

```
int _stdcall PTK_Connect(LPTSTR IPAddr, unsigned int netPort);
```

Parameters:

IPAddr: the printer's IP address.

netPort: the printer's network port (normally 9100).

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

---

### Example:

```
PTK_Connect(_T("199.9.10.245"), 9100);  
PTK_PrintLabel(1, 1); // Print one label  
PTK_CloseConnect();
```

### *PTK\_Connect\_Timer*

Description: connects to a printer over the network port, with a configurable connection timeout.

### Prototype:

```
int _stdcall PTK_Connect_Timer(LPTSTR IPAddr, unsigned int netPort, unsigned int time_sec);
```

### Parameters:

IPAddr: the printer's IP address.

netPort: the printer's network port (normally 9100).

time\_sec: maximum time to wait for a connection, in seconds.

### Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

### Example:

```
PTK_Connect_Timer(_T("199.9.10.245"), 9100, 3); // Returns after 3 seconds if the connection has  
not been established  
PTK_PrintLabel(1, 1); // Print one label  
PTK_CloseConnect();
```

### *PTK\_CloseConnect*

Description: closes the network connection to the printer.

### Prototype:

```
int _stdcall PTK_CloseConnect(void);
```

### Parameters:

None

### Return Value:

0 -> OK

---

### Example:

```
PTK_Connect_Timer(_T("199.9.10.245"), 9100);  
PTK_PrintLabel(1, 1); // Print one label  
PTK_CloseConnect();
```

### *PTK\_OpenUSBPort\_Buff*

Description: creates a buffer into which data sent to the printer will be written, and opens a USB port.

Note: Calling PTK\_WriteBuffToPrinter sends the entire buffer contents to the open port at once.

### Prototype:

```
int _stdcall PTK_OpenUSBPort_Buff(unsigned int portNum);
```

### Parameters:

port: USB port number. Range: 1~255.

#### Note:

You can use PTK\_GetAllPrinterUSBInfo to inspect information about the USB port to which the printer is connected.

Passing 255 automatically opens the USB port of one connected printer. If multiple printers are connected, the one with the lowest USB ID is opened first.

### Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

### Example:

```
PTK_OpenUsbPort_Buff(255);  
  
PTK_DrawText(100, 50, 0, 6, 8, 8, 'N', _T("POSTEK POSTEK 2020"));  
PTK_PrintLabel(1, 1);  
PTK_WriteBuffToPrinter();  
  
PTK_CloseBuffPort();
```

### *PTK\_Connect\_Timer\_Buff*

Description: creates a buffer into which data sent to the printer will be written, and opens a network port.

Note: Calling PTK\_WriteBuffToPrinter sends the entire buffer contents to the open port at once.

### Prototype:



---

```
int _stdcall PTK_Connect_Timer_Buff(LPTSTR IPAddr, unsigned int netPort, unsigned int time_sec);
```

Parameters:

IPAddr: the printer's IP address.  
netPort: the printer's network port (normally 9100).  
time\_sec: maximum time to wait for a connection, in seconds.

Return Value:

0 -> OK  
For other return values, call PTK\_GetErrorInfo to parse them.

**Example:**

```
PTK_Connect_Timer_Buff(_T("199.9.10.233"), 9100, 10);  
PTK_DrawText(100, 50, 0, 6, 8, 8, 'N', _T("POSTEK POSTEK 2020"));  
PTK_PrintLabel(1, 1);  
PTK_WriteBuffToPrinter();  
PTK_CloseBuffPort();
```

### *PTK\_OpenPrinter\_Buff*

Description: creates a buffer into which data sent to the printer will be written, and opens a driver port.

Note: Calling PTK\_WriteBuffToPrinter sends the entire buffer contents to the open port at once.

Prototype:

```
int _stdcall PTK_OpenPrinter_Buff(LPTSTR printerName);
```

Parameters:

printerName: printer name.

Note:

You can use PTK\_GetAllPrinterUSBInfo to inspect the printer name.  
Alternatively, open "Devices and Printers" on the PC to view the printer driver name.

Return Value:

0 -> OK  
For other return values, call PTK\_GetErrorInfo to parse them.

**Example:**

```
PTK_OpenPrinter_Buff(_T("POSTEK TX6"));
```

---

```
PTK_DrawText(100, 50, 0, 6, 8, 8, 'N', _T("POSTEK POSTEK 2020"));
PTK_PrintLabel(1, 1);
PTK_WriteBuffToPrinter();
PTK_CloseBuffPort();
```

### *PTK\_CloseBuffPort*

Description: frees the buffer and closes the open port.

#### Prototype:

```
int _stdcall PTK_CloseBuffPort(void);
```

#### Parameters:

None

#### Return Value:

0 -> OK

#### Example:

```
PTK_OpenPrinter_Buff(_T("POSTEK TX6"));
PTK_DrawText(100, 50, 0, 6, 8, 8, 'N', _T("POSTEK POSTEK 2020"));
PTK_PrintLabel(1, 1);
PTK_WriteBuffToPrinter();
PTK_CloseBuffPort();
```

### *PTK\_WriteBuffToPrinter*

Description: sends the contents of the buffer to the open port.

Note: Do not call any API function that reads data while in buffered mode, or the port will block.

#### Prototype:

```
int _stdcall PTK_WriteBuffToPrinter(void);
```

#### Parameters:

None

#### Return Value:

0 -> OK

#### Example:

```
PTK_OpenPrinter_Buff(_T("POSTEK TX6"));
PTK_DrawText(100, 50, 0, 6, 8, 8, 'N', _T("POSTEK POSTEK 2020"));
PTK_PrintLabel(1, 1);
PTK_WriteBuffToPrinter();
PTK_CloseBuffPort();
```

---

### *PTK\_SendFile*

Description: sends the contents of a file to the printer.

Prototype:

```
int _stdcall PTK_SendFile(LPTSTR filePath);
```

Parameters:

filePath: path of the file to be sent.

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
PTK_OpenTextPort(_T("command.txt"));
PTK_AnyGraphicsPrint(0, 0, _T("P1"), _T("Koala.jpg"), 1, 0, 0, 0);
PTK_PrintLabel(1, 1);
PTK_CloseTextPort();

PTK_OpenUSBPort(255);
PTK_SendFile(_T("command.txt"));
PTK_CloseUSBPort();
```

## Printer settings

### *PTK\_SetCharSets*

Description:

The PTK\_SetCharSets function sets the character set.

Prototype:

```
int PTK_SetCharSets(unsigned int BitValue, UCHAR CharSets, LPTSTR CountryCode);
```

Parameters:

---

BitValue: data bits. 8 indicates 8-bit, 7 indicates 7-bit.

CharSets: character set.

CharSets(CharSets=8)	Code page	Country/region of use
0	CP437	English - US: US English.
1	CP850	Latin1: regions using Western European languages.
2	CP852	Latin2: regions using Central European languages.
3	CP860	Portuguese: used under Portuguese DOS; also suitable for Spanish and Italian.
4	CP863	French Canadian: used under Canadian DOS to write French (mainly in Quebec).
5	CP865	Nordic: used under Danish and Norwegian DOS to write Nordic languages (except Icelandic).
6	CP857	Turkish: used under Turkish DOS to write Turkish.
7	CP861	Icelandic: used under Icelandic DOS to write Icelandic (and other Nordic languages).
8	CP862	Hebrew: used under Israeli DOS to write Hebrew.
9	CP855	Cyrillic: previously used widely in Serbia, Macedonia and Bulgaria for writing Cyrillic, although CP866 was more common.
10	CP866	Cyrillic CIS 1: used under Cyrillic Russian DOS to write Cyrillic.
11	CP737	Greek: used under Greek DOS to write Greek.
12	CP851	Greek 1: used under Greek DOS to write Greek, German and some French. Now largely replaced by CP869.
13	CP869	Greek 2: used under Greek DOS to write Greek, but less widely than CP737.
A	CP1252/ Windows1252	A single-byte character encoding for the Latin alphabet, used by default in legacy Microsoft Windows components. It covers English and many European languages including Spanish, French, German and Portuguese, and is the most widely used single-byte character encoding in the world.
B	CP1250/ Windows1250	A Microsoft Windows code page used to represent text in Central and Eastern European languages written in the Latin script, such as Polish, Czech, Slovak, Hungarian, Slovenian, Bosnian, Croatian, Serbian (Latin script), Romanian (before the 1993 spelling reform) and Albanian.
C	CP1251/ Windows1251	A Microsoft Windows code page covering languages that use the Cyrillic alphabet, such as Russian, Bulgarian, Serbian Cyrillic and others. It is the most widely used encoding for Bulgarian, Serbian and Macedonian.
D	CP1253/ Windows1253	A Microsoft Windows code page used to write Greek; now largely replaced by Unicode.
E	CP1254/ Windows1254	A Microsoft Windows code page used to write Turkish; now largely replaced by Unicode.
F	CP1255/ Windows1255	A Microsoft Windows code page used to write Hebrew; now largely replaced by Unicode.
G	CP936/ GBK	Chinese: a character set widely used in China. Under Chinese DOS it can write Simplified Chinese, Traditional Chinese, and some Japanese, Korean and Russian.
U	CP65001/ UTF-8	UTF-8: the most common encoding of the Unicode

---

CharSets(CharSets =8)	Code page	Country/region of use
	UTF-8	character set, covering languages from all countries.

CountryCode: country code of the programmable keyboard (KDU).

Return Value:

0 -> OK;

For other return values, see the section "CDFPSK.dll error return code descriptions".

Example:

```
Int return = PTK_SetCharSets (8,'N',"001" );
```

### *PTK\_PrintConfiguration*

Description: instructs the printer to print the self-test page.

Prototype:

```
int _stdcall PTK_PrintConfiguration(void);
```

Parameters:

None

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
PTK_OpenUSBPort (255);
PTK_PrintConfiguration();
PTK_CloseUSBPort ();
```

### *PTK\_MediaDetect*

Description: instructs the printer to perform media calibration.

Prototype:

```
int _stdcall PTK_MediaDetect(void);
```

Parameters:

None

Return Value:

0 -> OK

---

For other return values, call PTK\_GetErrorInfo to parse them.

**Example:**

```
PTK_OpenUSBPort(255);  
PTK_MediaDetect();  
PTK_CloseUSBPort();
```

*PTK\_FeedMedia*

Description: instructs the printer to feed one label.

**Prototype:**

```
int _stdcall PTK_FeedMedia(void);
```

Parameters:

None

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

**Example:**

```
PTK_OpenUSBPort(255);  
PTK_FeedMedia();  
PTK_CloseUSBPort();
```

*PTK\_UserFeed*

Description: instructs the printer to feed a fixed length of media forward.

**Prototype:**

```
int _stdcall PTK_UserFeed(unsigned int feedLen);
```

Parameters:

feedLen: feed length, in dots.

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

**Example:**

---

```
PTK_OpenUSBPort(255);  
PTK_UserFeed(360);  
PTK_CloseUSBPort ();
```

### *PTK\_UserBackFeed*

Description: instructs the printer to feed a fixed length of media backward.

#### Prototype:

```
int _stdcall PTK_UserBackFeed(unsigned int feedLen);
```

#### Parameters:

feedLen: feed length, in dots.

#### Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

#### Example:

```
PTK_OpenUSBPort(255);  
PTK_UserBackFeed(360);  
PTK_CloseUSBPort ();
```

### *PTK\_EnableFLASH*

Description: instructs the printer to enable FLASH storage; subsequent calls to storage-related API functions will store data in the printer's FLASH.

Note: Used to store forms and images; see the related APIs for details.

#### Prototype:

```
int _stdcall PTK_EnableFLASH(void);
```

#### Parameters:

None

#### Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

#### Example:

---

### Example 1: Store a form

```
PTK_OpenUSBPort(255);  
/* Store a form */  
PTK_EnableFLASH();// Enable FLASH storage  
PTK_FormDel(_T("F1"));// Delete any form with the same name to avoid name collisions  
PTK_FormDownload(_T("F1"));// Instruct the printer to start storing the form content  
PTK_DrawTextEx(100, 100, 0, 6, 2, 2, 'N', _T("1234567"), FALSE);// Print one line of text  
PTK_FormEnd();// Tell the printer that form storage is complete  
PTK_DisableFLASH();// Disable FLASH storage  
PTK_CloseUSBPort();
```

### Example 2: Store an image

```
PTK_OpenUSBPort(255);  
PTK_EnableFLASH();// Enable FLASH storage  
PTK_PcxGraphicsDel(_T("P1"));  
PTK_AnyGraphicsDownload(_T("P1"), _T("koala.jpg"), 1, 0, 0, 0);  
PTK_DisableFLASH();// Disable FLASH storage  
PTK_CloseUSBPort();
```

### Note:

Use PTK\_GetStorageList to view the forms and images stored in FLASH.

### *PTK\_DisableFLASH*

Description: disables FLASH storage.

### Prototype:

```
int _stdcall PTK_DisableFLASH(void);
```

### Parameters:

None

### Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

### Example:

#### Example 1: Store a form



---

```
PTK_OpenUSBPort(255);
/* Store a form */
PTK_EnableFLASH();// Enable FLASH storage
PTK_FormDel(_T("F1"));// Delete any form with the same name to avoid name collisions
PTK_FormDownload(_T("F1"));// Instruct the printer to start storing the form content
PTK_DrawTextEx(100, 100, 0, 6, 2, 2, 'N', _T("1234567"), FALSE);// Print one line of text
PTK_FormEnd();// Tell the printer that form storage is complete
PTK_DisableFLASH();// Disable FLASH storage
PTK_CloseUSBPort();
```

Example 2: Store an image

```
PTK_OpenUSBPort(255);
PTK_EnableFLASH();// Enable FLASH storage
PTK_PcxGraphicsDel(_T("P1"));
PTK_AnyGraphicsDownload(_T("P1"), _T("koala.jpg"), 1, 0, 0, 0);
PTK_DisableFLASH();// Disable FLASH storage
PTK_CloseUSBPort();
```

Note:

Use `PTK_GetStorageList` to view the forms and images stored in FLASH.

### *PTK\_CutPage*

**Description:** sets the cut frequency. Preserved across power cycles.

**Prototype:**

```
int _stdcall PTK_CutPage(unsigned int page)
```

**Parameters:**

page: cut frequency, i.e. the number of labels printed between cuts.

**Return Value:**

0 -> OK

For other return values, call `PTK_GetErrorInfo` to parse them.

**Example:**

```
PTK_OpenUSBPort(255);
PTK_CutPage(3);
PTK_PrintLabel(6, 1);// Print 6 labels
PTK_CloseUSBPort();
```

---

### *PTK\_CutPageEx*

Description: sets the cut frequency. Not preserved across power cycles.

Prototype:

```
int _stdcall PTK_CutPageEx(unsigned int page)
```

Parameters:

page: cut frequency, i.e. the number of labels printed between cuts.

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
PTK_OpenUSBPort(255);
PTK_CutPageEx(3);
PTK_PrintLabel(6, 1); // Print 6 labels; if cutter mode is enabled, a cut is made after every 3
labels
PTK_CloseUSBPort();
```

### *PTK\_SetCoordinateOrigin*

Description: sets the origin of the print coordinate system.

Prototype:

```
int _stdcall PTK_SetCoordinateOrigin(unsigned int px, unsigned int py);
```

Parameters:

px: X (horizontal) coordinate.

py: Y (vertical) coordinate.

Note: See "Coordinate system of a barcode label printer".

Note: when the label is printed in reverse direction, the origin is also reversed. See PTK\_SetDirection.

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```

PTK_OpenUSBPort(255);
PTK_SetCoordinateOrigin(100,100);
PTK_DrawTextEx(0, 0, 0, 6, 3, 3, 'N', _T("ABC123"), FALSE);
PTK_PrintLabel(1, 1);
PTK_CloseUSBPort();

```

### PTK\_GetUtilityInfo

Description: retrieves the commonly used configuration information saved in the printer's FLASH.

Note: Only supports reading via serial port and USB.

Prototype:

```
int _stdcall PTK_GetUtilityInfo(unsigned int infoNum, LPTSTR data, unsigned int dataSize);
```

Parameters:

infoNum: index of the information item to retrieve.

data: buffer that stores the retrieved information.

dataSize: size of the data buffer.

infoNum	Printer information represented by infoNum	Returned length of data	Parsing of data
1	Printer language	1	"0": Chinese. "1": English.
2	Print method	1	"0": direct thermal. "1": thermal transfer.
3	Media sensing mode	1	"0": see-through. "1": bottom reflective. "2": top reflective. Note: For small-form-factor models, the DIP switch settings take precedence.
4	Print darkness	2	"00": use the value set by PTK_SetDarkness. "01": darkness 1. "02": darkness 2. ..... "20": darkness 20.
5	Print speed	2	"00": use the value set by PTK_SetPrintSpeed. "01" speed 1 ips "02" speed 2 ips "03" speed 2.5 ips "04" speed 3 ips "05" speed 3.5 ips "06" speed 4 ips "07" speed 5 ips "08" speed 6 ips "09" speed 7 ips "10" speed 8 ips Note: Set this according to the maximum speed supported by the printer model.
6	Print direction	1	"0": normal direction (the coordinate system of a barcode label printer).

			"1": reverse direction.
7	Firmware version number	4	Printer firmware version, e.g. "7.58".
8	Firmware identifier	11	Unique identifier for the printer model and firmware combination For example, "00.8083.909".
9	Printer command type	1	"0":PPLE "1":PPLZ
11	Cutter mode switch	1	"0": disabled. "1": enabled. Note: For small-form-factor models, the DIP switch settings take precedence. Note: The cutter and the peeler cannot be installed at the same time.
12	Cut frequency	2	"01": cut after every 1 label printed. "02" cut after every 2 labels printed ..... "99" cut after every 99 labels printed
13	Blade position	2	"00": blade position offset 0. "01" blade position offset 1 ..... "99" blade position offset 99 <b>Note: This parameter is set to its factory default. Do not change it without guidance from qualified service personnel.</b>
14	Peel-off mode switch	1	"0": disabled. "2": enabled. Note: For small-form-factor models, the DIP switch settings take precedence. Note: The cutter and the peeler cannot be installed at the same time.
15	Serial port baud rate	1	"0":9600 "1":19200 "2":38400 "3":57600 "4":115200 Note: For small-form-factor models, the DIP switch settings take precedence.
16	Serial port parity bit	1	"0": no parity.
17	Serial port data bits	1	"0": 8 data bits.
18	IPv4 address	Maximum 15 characters	IP address string For example, "192.168.1.2".
19	Subnet mask	Maximum 15 characters	Subnet mask string For example, "255.255.255.0".
20	Gateway	Maximum 15 characters	Gateway string For example, "192.168.1.1".
21	Network port	4	Network port string For example, "9100".
22	Horizontal offset	4	Units: dots For example, "0012" equals 1 mm on a 300 DPI machine. For example, "0024" equals 3 mm on a 203 DPI machine. Note: For unit conversions, see "Dots: pixels".
23	Vertical offset	4	Units: dots For example, "0012" equals 1 mm on a 300

			DPI machine. For example, "0024" equals 3 mm on a 203 DPI machine. Note: For unit conversions, see "Dots: pixels".
24	Tear-off offset	4	Units: dots For example, "0012" equals 1 mm on a 300 DPI machine. For example, "0024" equals 3 mm on a 203 DPI machine. Note: For unit conversions, see "Dots: pixels".
25	Top-of-page (TPH) offset	4	Units: dots For example, "0012" equals 1 mm on a 300 DPI machine. For example, "0024" equals 3 mm on a 203 DPI machine. Note: For unit conversions, see "Dots: pixels".
26	Cut offset	4	Units: dots For example, "0012" equals 1 mm on a 300 DPI machine. For example, "0024" equals 3 mm on a 203 DPI machine. Note: For unit conversions, see "Dots: pixels".
27	Peel-off offset	4	Units: dots For example, "0012" equals 1 mm on a 300 DPI machine. For example, "0024" equals 3 mm on a 203 DPI machine. Note: For unit conversions, see "Dots: pixels".
28	Date and time	14 Year: 4 digits Month: 2 digits Day: 2 digits Hour: 2 digits Minute: 2 digits Second: 2 digits	Year, month, day, hour, minute, second For example, 13:54:37 on January 22, 2020. Pass "20200122135437".
29	Total length printed	6	Units: meters For example, "000050" indicates that 50 meters have been printed.
30	Tear-off mode switch	1	"0": disabled. "1": enabled. Note: For small-form-factor models, the DIP switch settings take precedence.
31	Current printer status	3	See "Printer status code descriptions" for the meaning of the values.

---

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
enum
{
    POSOriginalInfo = 0, // Raw information: all printer information
    POSLanguage = 1,     // Printer language C/E
    POSThermal = 2,      // Print method: direct thermal / thermal transfer
    POSSensorMode = 3,   // Media sensing mode: see-through / bottom reflective / top reflective
    POSDark = 4, // Print darkness 0-20
    POSSpeed = 5,        // Print speed 1-8
    POSDirect = 6,       // Print direction
    POSVersion = 7,      // Print firmware version
    POSPrintNum = 8,     // Printer part number
    POSCMDType = 9,      // Printer command type
    POSDPI = 10, // Printer resolution
    POSCUTMode = 11, // Cutter mode
    POSCUTPage = 12, // Cutter frequency
    POSCUTPos = 13, // Blade position
    POSPEELMode = 14,   // Peel-off mode
    POSCOMSpeed = 15,   // Serial port baud rate
    POSCOMParity = 16,  // Serial port parity bit
    POSCOMLength = 17, // Serial port data bits
    POSIP = 18, // Printer ID
    POSIPMask = 19, // Printer subnet mask
    POSGateWay = 20,   // Printer gateway
    POSPort = 21, // Printer network port
    POSOffsetX = 22, // Printer horizontal offset
    POSOffsetY = 23, // Vertical offset
    POSOffsetPaneel = 24, // Tear-off offset
    POSOffsetTph = 25, // Top-of-page (TPH) offset
    POSOffsetCut = 26, // Cut offset
    POSOffsetPeel = 27, // Peel-off offset
    POSDate = 28, // Printer's current time
    POSPrintSum = 29, // Total length printed by this printer
    POSPTearMode = 30, // Tear-off mode
    POSStatus = 31,   // Printer status
};
TCHAR buff[1024] = { 0 };
```

```

PTK_OpenUSBPort(255);
PTK_GetUtilityInfo(POSVersion, buff, sizeof(buff));
MessageBox(buff);
PTK_CloseUSBPort();

```

### *PTK\_SetUtilityInfoProc*

Description: configures the printer's commonly used information.

Note:

1. The printer information configured by this function is saved to FLASH and persists across power cycles.
2. Must be used together with PTK\_GetUtilityInfo and PTK\_SetUtilityInfo.

Call PTK\_SetUtilityInfo to take effect.

Prototype:

```
int _stdcall PTK_SetUtilityInfoProc(LPTSTR _GInfo, unsigned int infoNum, LPTSTR info);
```

Parameters:

\_GInfo: the raw printer information returned by a call to PTK\_GetUtilityInfo.

infoNum: index of the information item to set, formatted as follows:

Note: Not all information items can be set.

info: printer information to configure, formatted as follows:

Note: If the supplied string has an incorrect length, the setting will fail when sent to the printer.

infoNum	Printer information represented by infoNum	Length of info	info
1	Printer language	1	"0": Chinese. "1": English.
2	Print method	1	"0": direct thermal. "1": thermal transfer.
3	Media sensing mode	1	"0": see-through. "1": bottom reflective. "2": top reflective. Note: For small-form-factor models, the DIP switch settings take precedence.
4	Print darkness	2	"00": use the value set by PTK_SetDarkness. "01": darkness 1. "02": darkness 2. ..... "20": darkness 20.
5	Print speed	2	"00": use the value set by PTK_SetPrintSpeed. "01" speed 1 ips "02" speed 2 ips "03" speed 2.5 ips "04" speed 3 ips "05" speed 3.5 ips "06" speed 4 ips

			"07" speed 5 ips "08" speed 6 ips "09" speed 7 ips "10" speed 8 ips Note: Set this according to the maximum speed supported by the printer model.
6	Print direction	1	"0": normal direction (the coordinate system of a barcode label printer). "1": reverse direction.
9	Printer command type	1	"0":PPLE "1":PPLZ
11	Cutter mode switch	1	"0": disabled. "1": enabled. Note: For small-form-factor models, the DIP switch settings take precedence. Note: The cutter and the peeler cannot be installed at the same time.
12	Cut frequency	2	"01": cut after every 1 label printed. "02" cut after every 2 labels printed ..... "99" cut after every 99 labels printed
13	Blade position	2	"00": blade position offset 0. "01" blade position offset 1 ..... "99" blade position offset 99 Note: This parameter is set to its factory default. Do not change it without guidance from qualified service personnel.
14	Peel-off mode switch	1	"0": disabled. "2": enabled. Note: For small-form-factor models, the DIP switch settings take precedence. Note: The cutter and the peeler cannot be installed at the same time.
15	Serial port baud rate	1	"0":9600 "1":19200 "2":38400 "3":57600 "4":115200 Note: For small-form-factor models, the DIP switch settings take precedence.
18	IPv4 address	Maximum 15 characters	IP address string For example, "192.168.1.2".
19	Subnet mask	Maximum 15 characters	Subnet mask string For example, "255.255.255.0".
20	Gateway	Maximum 15 characters	Gateway string For example, "192.168.1.1".
21	Network port	4	Network port string For example, "9100".
22	Horizontal offset	4	Units: dots For example, "0012" equals 1 mm on a 300 DPI machine. For example, "0024" equals 3 mm on a 203 DPI machine. Note: For unit conversions, see "Dots: pixels".



23	Vertical offset	4	Units: dots For example, "0012" equals 1 mm on a 300 DPI machine. For example, "0024" equals 3 mm on a 203 DPI machine. Note: For unit conversions, see "Dots: pixels".
24	Tear-off offset	4	Units: dots For example, "0012" equals 1 mm on a 300 DPI machine. For example, "0024" equals 3 mm on a 203 DPI machine. Note: For unit conversions, see "Dots: pixels".
25	Top-of-page (TPH) offset	4	Units: dots For example, "0012" equals 1 mm on a 300 DPI machine. For example, "0024" equals 3 mm on a 203 DPI machine. Note: For unit conversions, see "Dots: pixels".
26	Cut offset	4	Units: dots For example, "0012" equals 1 mm on a 300 DPI machine. For example, "0024" equals 3 mm on a 203 DPI machine. Note: For unit conversions, see "Dots: pixels".
27	Peel-off offset	4	Units: dots For example, "0012" equals 1 mm on a 300 DPI machine. For example, "0024" equals 3 mm on a 203 DPI machine. Note: For unit conversions, see "Dots: pixels".
28	Date and time	14 Year: 4 digits Month: 2 digits Day: 2 digits Hour: 2 digits Minute: 2 digits Second: 2 digits	Year, month, day, hour, minute, second For example, 13:54:37 on January 22, 2020. Pass "20200122135437".
30	Tear-off mode switch	1	"0": disabled. "1": enabled. Note: For small-form-factor models, the DIP switch settings take precedence.

---

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
enum
{
    POSLanguage = 1,      // Printer language Chinese/English
    POSThermal = 2,       // Print method: direct thermal / thermal transfer
    POSSensorMode = 3,    // Media sensing mode: see-through / bottom reflective / top reflective
    POSDark = 4, // Print darkness 0-20
    POSSpeed = 5,         // Print speed 1-8
    POSDirect = 6,        // Print direction
    POSCMDType = 9,       // Printer command type
    POSCUTMode = 11,      // Cutter mode
    POSCUTPage = 12,      // Cut frequency
    POSCUTPos = 13, // Blade position
    POSPEELMode = 14,     // Peel-off mode
    POSCOMSpeed = 15,     // Serial port baud rate
    POSIP = 18, // Printer IP
    POSIPMask = 19, // Printer subnet mask
    POSGateWay = 20,      // Printer gateway
    POSPort = 21, // Printer port
    POSOffsetX = 22, // Printer horizontal offset
    POSOffsetY = 23, // Vertical offset
    POSOffsetPaneel = 24, // Tear-off offset
    POSOffsetTph = 25,    // Top-of-page (TPH) offset
    POSOffsetCut = 26,    // Cut offset
    POSOffsetPeel = 27,   // Peel-off offset
    POSDate = 28, // Printer's current time
    POSPTearMode = 30,    // Tear-off mode
};
TCHAR buff[1024] = { 0 };
PTK_OpenUSBPort(255);
PTK_GetUtilityInfo(0, buff, sizeof(buff));
PTK_SetUtilityInfoProc(buff, POSDark, _T("08")); // Set the print darkness to 8
PTK_SetUtilityInfoProc(buff, POSSpeed, _T("05")); // Set the print speed to 3.5 ips
PTK_SetUtilityInfoProc(buff, POSDirect, _T("1")); // Set the print direction to reverse
PTK_SetUtilityInfo(buff);
PTK_CloseUSBPort();
```

### *PTK\_SetUtilityInfo*

Description: sends commonly used printer configuration information to the printer's FLASH to be saved.

---

Note: This function waits for the printer to finish saving before returning.

**Prototype:**

```
int _stdcall PTK_SetUtilityInfo(LPTSTR _G1Info);
```

**Parameters:**

\_G1Info: the printer configuration information returned after calling PTK\_GetUtilityInfo and PTK\_SetUtilityInfoProc.

**Return Value:**

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

**Example:**

```
enum
{
    POSLanguage = 1,      // Printer language Chinese/English
    POSThermal = 2,       // Print method: direct thermal / thermal transfer
    POSSensorMode = 3,    // Media sensing mode: see-through / bottom reflective / top reflective
    POSDark = 4, // Print darkness 0-20
    POSSpeed = 5,         // Print speed 1-8
    POSDirect = 6,        // Print direction
    POSCMDType = 9,       // Printer command type
    POSCUTMode = 11,      // Cutter mode
    POSCUTPage = 12,      // Cut frequency
    POSCUTPos = 13, // Blade position
    POSPEELMode = 14,     // Peel-off mode
    POSCOMSpeed = 15,     // Serial port baud rate
    POSIP = 18, // Printer IP
    POSIPMask = 19, // Printer subnet mask
    POSGateWay = 20,      // Printer gateway
    POSPort = 21, // Printer port
    POSOffsetX = 22, // Printer horizontal offset
    POSOffsetY = 23, // Vertical offset
    POSOffsetPaneel = 24, // Tear-off offset
    POSOffsetTph = 25,    // Top-of-page (TPH) offset
    POSOffsetCut = 26,    // Cut offset
    POSOffsetPeel = 27,   // Peel-off offset
    POSDate = 28, // Printer's current time
    POSPTearMode = 30,    // Tear-off mode
};

TCHAR buff[1024] = { 0 };
PTK_OpenUSBPort(255);
PTK_GetUtilityInfo(0, buff, sizeof(buff));
```

---

```

PTK_SetUtilityInfoProc(buff, POSDark, _T("08")); // Set the print darkness to 8
PTK_SetUtilityInfoProc(buff, POSSpeed, _T("05")); // Set the print speed to 3.5 ips
PTK_SetUtilityInfoProc(buff, POSDirect, _T("1")); // Set the print direction to reverse
PTK_SetUtilityInfo(buff);
PTK_CloseUSBPort();

```

### *PTK\_GetAllPrinterInfo*

**Description:** retrieves all configuration information saved in the printer's FLASH.

**Note:** This function is currently only supported on printers with firmware version 7.58 or later.

**Note:** Supports reading via serial port, USB and network.

**Prototype:**

```
int _stdcall PTK_GetAllPrinterInfo(int infoNum, BOOL fileflag, LPTSTR data, DWORD dataSize)
```

**Parameters:**

**infoNum:** index of the printer information item to retrieve.

**fileflag:** whether to create a file in the current directory to save. FALSE - no; TRUE - yes.

Note: If you select yes, a PrinterConfig.txt file will be created in the DLL's directory.

**data:** buffer that stores the printer configuration information.

**dataSize:** size of the data buffer.

infoNum	Printer information represented by infoNum	Parsing of data
0	All information	Returned data format: see the sample returned data in the example. START indicates the start of the data; \r\n (newline) is the separator; CRC is the CRC checksum over all data; END\r\n indicates the end of the data;
1	Printer name	Printer model name
2	Printer language	"C": Chinese. "E": English.
3	Serial port baud rate	"9600": printer serial port baud rate 9600. "19200": printer serial port baud rate 19200. "38400": printer serial port baud rate 38400. "57600": printer serial port baud rate 57600. "115200": printer serial port baud rate 115200.
4	Serial port parity bit	"N": no parity.
5	Serial port data bits	"8": 8 data bits.
6	Serial port stop bit	"1": 1 stop bit.
7	Firmware version number	For example, "7.58" indicates that the firmware version is 7.58.
8	Firmware identifier	For example, "00.1033.934" is the unique firmware identifier for this model.
9	Command type	"0": PPLE "1": PPLZ
10	HF tag protocol type	"0": none. "1": ISO 15693 protocol.

		"2": ISO 14443A protocol.
11	Total printer FLASH capacity	Units: MB For example, "32" indicates a FLASH capacity of 32 MB.
12	Total printer RAM capacity	Units: MB For example, "32" indicates a RAM capacity of 32 MB.
13	Number of labels printed	Units: labels For example, "7425" indicates that 7425 labels have been printed.
14	Length of labels printed	Units: meters For example, "125" indicates a printed label length of 125 meters.
15	Ribbon detection switch	"0": off. "1": on.
16	Tear-off mode switch	"0": off. "1": on.
17	Tear-off offset	Units: dots For example, "12" indicates a tear-position offset of 12 dots. Note: For unit conversions, see "Dots: pixels".
18	Cut mode switch	"0": off. "1": on.
19	Cut offset	Units: dots For example, "12" indicates a cut-position offset of 12 dots. Note: For unit conversions, see "Dots: pixels".
20	Cut frequency	Units: per label For example, "3" indicates a cut after every 3 labels printed.
21	Blade position	Units: ms For example, "25" indicates that the cutter stop position is offset by 25 ms of rotation.
22	Peel-off mode switch	"0": off. "1": on.
23	Peel-off offset	Units: dots For example, "12" indicates a peel-position offset of 12 dots. Note: For unit conversions, see "Dots: pixels".
24	Print darkness	Range: 0~20. The larger the value, the darker the print result. For example, "8" indicates a print darkness of 8.
25	Print speed	Units: ips, with one decimal place. For example, "2.5" indicates a print speed of 2.5 ips. "3.0" represents a print speed of 3.0 ips.
26	Print method	"0": direct thermal. "1": thermal transfer.
27	Media sensing mode	"0": see-through. "1": bottom reflective. "2": top reflective.

28	Media calibration feed length	Units: mm For example, "200" indicates that 200 mm of media is fed during calibration.
29	Print direction	"0": normal direction (the coordinate system of a barcode label printer). "1": reverse direction.
30	Vertical offset	Units: dots For example, "12" equals 1 mm on a 300 DPI machine. For example, "24" equals 3 mm on a 203 DPI machine. Note: For unit conversions, see "Dots: pixels".
31	Horizontal offset	Units: dots For example, "12" equals 1 mm on a 300 DPI machine. For example, "24" equals 3 mm on a 203 DPI machine. Note: For unit conversions, see "Dots: pixels".
32	Top-of-page (TPH) offset	Units: dots For example, "12" equals 1 mm on a 300 DPI machine. For example, "24" equals 3 mm on a 203 DPI machine. Note: For unit conversions, see "Dots: pixels".
33	RFID feature switch	"0": off. "1": on.
34	RFID write power	Units: dB For example, "25" indicates a write power of 25 dB.
35	RFID read power	Units: dB For example, "18" indicates a write power of 18 dB.
36	RFID probing offset	Units: dots For example, "12" equals 1 mm on a 300 DPI machine. For example, "24" equals 3 mm on a 203 DPI machine. Note: For unit conversions, see "Dots: pixels".
37	RFID read/write retry count	Units: times For example, "3" indicates up to 3 retries after an RFID tag read/write failure.
38	RFID password feature switch	"0": off. "1": on.
39	RFID R6 permanent lock switch	"0": off. "1": on.
40	RFID frequency band selection	Returns the country / region abbreviation (Region Name); see the appendix for details. <a href="#">Table - Country / region frequency bands</a> Default: "PRC".
41	Wireless type	"none": none. "bluetooth": Bluetooth. "wifi": Wi-Fi.

42	MAC address	Hexadecimal addresses separated by spaces For example, "00 1B E7 0B 58 45".
43	DHCP switch	"0": off. "1": on.
44	IP address	The printer's IP address, e.g. "199.9.10.245".
45	Subnet mask	The printer's subnet mask, e.g. "255.255.255.0".
46	Gateway	The gateway used by the printer to connect to the network, e.g. "199.9.10.1".
47	Network port	The printer's network port, e.g. "9100".
48	Current printer status	See "Printer status code descriptions" for the meaning of the values.
49	Image resolution conversion	"N": disabled. "S": 600 dpi -> 300 dpi; the image is reduced to one quarter of its original size. "B": 300 dpi -> 600 dpi; the image is enlarged by a factor of four. Note: Not supported on machines whose print head resolution is 200 dpi.
50	Overall scaling	"N": disabled. "S": reduce the entire label content to one quarter of its original size (only supported on 300 dpi printers). "B": enlarge the entire label content by a factor of four (only supported on 600 dpi printers).
51	USBID	The printer's USB serial number, unique to each printer.

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
TCHAR buff[1024] = { 0 };
PTK_OpenUSBPort(255);
PTK_GetAllPrinterInfo(0, FALSE, buff, sizeof(buff));
MessageBox(buff);
PTK_CloseUSBPort();
```

### *PTK\_SetAllPrinterInfo*

Description: sets all printer information and saves it to FLASH (corresponds to PTK\_GetPrinterInfo).

Note: This function is currently only supported on printers with firmware version 7.58 or later.

Prototype:

```
int _stdcall PTK_SetAllPrinterInfo(int infoNum, LPTSTR printerInfo);
```

Parameters:

infoNum: index of the printer information item to retrieve. Some fixed-value items cannot be set.

printerInfo: value to be set, in the following format:

infoNum	Printer information represented by infoNum	printerInfo value
2	Printer language	"C": Chinese. "E": English.
3	Serial port baud rate	"9600": printer serial port baud rate 9600. "19200": printer serial port baud rate 19200. "38400": printer serial port baud rate 38400. "57600": printer serial port baud rate 57600. "115200": printer serial port baud rate 115200.
9	Command type	"0": PPLE "1": PPLZ
10	HF tag protocol type	"0": none. "1": ISO 15693 protocol. "2": ISO 14443A protocol.
13	Number of labels printed	Units: labels For example, "7425" indicates that 7425 labels have been printed.
14	Length of labels printed	Units: meters For example, "125" indicates a printed label length of 125 meters.
15	Ribbon detection switch	"0": off. "1": on.
16	Tear-off mode switch	"0": off. "1": on.
17	Tear-off offset	Units: dots For example, "12" indicates a tear-position offset of 12 dots. Note: For unit conversions, see "Dots: pixels".
18	Cut mode switch	"0": off. "1": on.
19	Cut offset	Units: dots For example, "12" indicates a cut-position offset of 12 dots. Note: For unit conversions, see "Dots: pixels".
20	Cut frequency	Units: per label For example, "3" indicates a cut after every 3 labels printed.
21	Blade position	Units: ms For example, "25" indicates that the cutter stop position is offset by 25 ms of rotation.
22	Peel-off mode switch	"0": off. "1": on.
23	Peel-off offset	Units: dots For example, "12" indicates a peel-position offset of 12 dots. Note: For unit conversions, see "Dots: pixels".
24	Print darkness	Range: 0~20. The larger the value, the darker the print result.



		For example, "8" indicates a print darkness of 8.
25	Print speed	Units: ips, with one decimal place. For example, "2.5" indicates a print speed of 2.5 ips. "3.0" represents a print speed of 3.0 ips.
26	Print method	"0": direct thermal. "1": thermal transfer.
27	Media sensing mode	"0": see-through. "1": bottom reflective. "2": top reflective.
28	Media calibration feed length	Units: mm For example, "200" indicates that 200 mm of media is fed during calibration.
29	Print direction	"0": normal direction (the coordinate system of a barcode label printer). "1": reverse direction.
30	Vertical offset	Units: dots For example, "12" equals 1 mm on a 300 DPI machine. For example, "24" equals 3 mm on a 203 DPI machine. Note: For unit conversions, see "Dots: pixels".
31	Horizontal offset	Units: dots For example, "12" equals 1 mm on a 300 DPI machine. For example, "24" equals 3 mm on a 203 DPI machine. Note: For unit conversions, see "Dots: pixels".
32	Top-of-page (TPH) offset	Units: dots For example, "12" equals 1 mm on a 300 DPI machine. For example, "24" equals 3 mm on a 203 DPI machine. Note: For unit conversions, see "Dots: pixels".
33	RFID feature switch	"0": off. "1": on.
34	RFID write power	Units: dB For example, "25" indicates a write power of 25 dB.
35	RFID read power	Units: dB For example, "18" indicates a write power of 18 dB.
36	RFID probing offset	Units: dots For example, "12" equals 1 mm on a 300 DPI machine. For example, "24" equals 3 mm on a 203 DPI machine. Note: For unit conversions, see "Dots: pixels".
37	RFID read/write retry count	Units: times For example, "3" indicates up to 3 retries after an RFID tag read/write failure.

38	RFID password feature switch	"0": off. "1": on.
39	RFID R6 permanent lock switch	"0": off. "1": on.
40	RFID frequency band selection	Serial Interface Region Code; see the appendix for details. <a href="#">Table - Country / region frequency bands</a> For example, passing "6" represents "PRC" (China).
41	Wireless type	"N": none. "B": Bluetooth. "W": Wi-Fi.
43	DHCP switch	"0": off. "1": on.
44	IP address	The printer's IP address, e.g. "199.9.10.245".
45	Subnet mask	The printer's subnet mask, e.g. "255.255.255.0".
46	Gateway	The gateway used by the printer to connect to the network, e.g. "199.9.10.1".
47	Network port	The printer's network port, e.g. "9100".
49	Image resolution conversion	"N": disabled. "S": 600 dpi -> 300 dpi; the image is reduced to one quarter of its original size. "B": 300 dpi -> 600 dpi; the image is enlarged by a factor of four. Note: Not supported on machines whose print head resolution is 200 dpi.
50	Overall scaling	"N": disabled. "S": reduce the entire label content to one quarter of its original size (only supported on 300 dpi printers). "B": enlarge the entire label content by a factor of four (only supported on 600 dpi printers).

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
PTK_OpenUSBPort(255);
PTK_SetAllPrinterInfo(2, _T("C")); // Set the printer language to Chinese
PTK_CloseUSBPort();
```

### *PTK\_ErrorReport\_USBInterrupt*

Description: retrieves the printer's current status in real time via a USB interrupt. See "Printer status code descriptions" for the meaning of the values.

---

**Prototype:**

```
int _stdcall PTK_ErrorReport_USBInterrupt(int* status);
```

**Parameters:**

status: integer buffer that receives the status code. See "Printer status code descriptions" for the meaning of the values.

**Return Value:**

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

**Example:**

```
int status = 0;
PTK_OpenUSBPort(255);
PTK_ErrorReport_USBInterrupt(&status);
PTK_CloseUSBPort ();
```

### *PTK\_GetPrinterStatus\_Net*

Description: retrieves the printer's current status over the network. Only supported on non-touchscreen printers. See "Printer status code descriptions" for the meaning of the values.

**Prototype:**

```
int _stdcall PTK_GetPrinterStatus_Net(LPTSTR printerStatus);
```

**Parameters:**

printerStatus: integer buffer that receives the status code. The data is a 4-byte ASCII code; see "Printer status code descriptions" for the meaning of the values.

**Return Value:**

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

**Example:**

```
TCHAR status[1024] = { 0 };
PTK_Connect_Timer(_T("199.9.10.245"), 9100, 3);
PTK_GetPrinterStatus_Net(status);
PTK_CloseConnect();
MessageBox(status);
```

---

### *PTK\_GetPrinterStatus\_Net\_OX*

Description: retrieves the printer's current status over the network. Only supported on OX-series printers. See "Printer status code descriptions" for the meaning of the values.

#### Prototype:

```
int _stdcall PTK_GetPrinterStatus_Net_OX(LPTSTR printerStatus);
```

#### Parameters:

printerStatus: integer buffer that receives the status code. See "OX-series printer status code descriptions" for the meaning of the values.

#### Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

#### Example:

```
TCHAR status[1024] = { 0 };
PTK_Connect_Timer(_T("199.9.10.245"), 9100, 3);
PTK_GetPrinterStatus_Net_OX(status);
PTK_CloseConnect();
MessageBox(status);
```

### *PTK\_GetLabelPrintInfo*

Description: retrieves label print information, remaining media and the key action after a print error. Only available on OX-platform firmware V1.7.0.15 and later.

#### Prototype:

```
int _stdcall PTK_GetLabelPrintInfo(LPTSTR indexInfo, LPTSTR Info);
```

#### Parameters:

indexInfo: information items to retrieve. A single value or multiple values may be specified; when specifying multiple values, separate them with an ASCII comma ",". For example, to choose 1, 3, 7 and 8, pass

The format is: "1,2,3,7,8".

The following information items are currently supported:

- {1, "printPage\_afterLeaveFactory"}, // Total number of labels printed
- {2, "printLen\_afterLeaveFactory"}, // Total label length printed (meters)
- {3, "printPage\_afterBooting"}, // Number of labels printed since power-on
- {4, "printPage\_lastTask"}, // Number of labels printed in the most recent job
- {5, "optKey\_afterError"}, // Key action after a print error
- {6, "printPage\_HaveParsed"}, // Number of labels parsed in the current job

---

```
{7, "printPage_MediaLeft"}, // Remaining media (label) %  
{8, "printPage_RibbonLeft"}, // Remaining ribbon %
```

The values of `optKey_afterError` are defined as follows:

```
KEY_NO = 0, // No key  
KEY_CANCEL_SHORT = 3, // Cancel error  
KEY_FEED_LONG = 5, // Label calibration  
KEY_CONTINUE_PRINT = 7, // Resume print  
KEY_REPEAT_PRINT = 8, // Reprint  
KEY_CANCEL_TASK = 9, // Delete task and cancel error
```

The values of `printPage_MediaLeft` and `printPage_RibbonLeft` are defined as follows:

```
0~100: percentage;  
101: Unknown, used for initialization;  
102: Invalid, probing is disabled;  
103: Low remaining material.
```

Info: buffer that stores the retrieved information. The data type is JSON.

Tip: To retrieve this information over the network, start a thread that connects to port 9200.

Return Value:

```
0 -> OK
```

For other return values, call `PTK_GetErrorInfo` to parse them.

Example:

```
PTK_OpenUSBPort(255);  
TCHAR data[1024] = { 0 };  
PTK_GetLabelPrintInfo (_T("1,2,3,4,5,6,7,8"), data);  
PTK_CloseUSBPort ();
```

Sample retrieved data:

```
{  
  "printPage_afterLeaveFactory": 39307,  
  "printLen_afterLeaveFactory": 2956,  
  "printPage_afterBootting": 3,  
  "printPage_lastTask": 2,  
  "optKey_afterError": 0,  
  "printPage_HaveParsed": 0,  
  "printPage_MediaLeft": 101,  
  "printPage_RibbonLeft": 102  
}
```

## Label settings

### *PTK\_ClearBuffer*

Description: clears the contents of the printer's buffer.

---

Note:

1. Before sending the content of a new label to the printer, it is recommended to use this command to clear any existing data in the printer's buffer, to prevent residual data from causing subsequent API calls to fail.
2. Do not call this function while a form is being stored, or form creation will fail.

Prototype:

```
int _stdcall PTK_ClearBuffer(void);
```

Parameters:

None

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
PTK_OpenUSBPort(255);
PTK_ClearBuffer();
PTK_DrawTextEx(100, 100, 0, 6, 3, 3, 'N', _T("ABC123"), FALSE);
PTK_PrintLabel(1, 1);
PTK_CloseUSBPort();
```

*PTK\_SetPrintSpeed*

Description: sets the current print speed. Not preserved across power cycles.

Note:

1. The maximum print speed differs by printer model (see the corresponding user manual). If the value set is greater than the printer's maximum print speed, the setting is ignored.
2. If the print speed has been set to a nonzero value via FLASH (for example, via the LCD screen), this API setting has no effect.
3. The PTK\_EnableFLASH function is only used to save forms and images; it has no effect on this API.

Prototype:

```
int _stdcall PTK_SetPrintSpeed(unsigned int speed);
```

Parameters:

speed: range 0-10 or 10-100.

p1 value	p1 value	Speed
2	20	2.0 ips (50.80 mm/s)
/	25	2.5 ips (63.50 mm/s)
3	30	3.0 ips (76.20 mm/s)
/	35	3.5 ips (88.90 mm/s)
4	40	4.0 ips (101.60 mm/s)

---

5	50	5.0 ips (127.00 mm/s)
6	60	6.0 ips (152.40 mm/s)
7	70	7.0 ips (177.80 mm/s)
8	80	8.0 ips (203.20 mm/s)
9	90	9.0 ips (228.60 mm/s)
10	100	10.0 ips (254.00 mm/s)

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
PTK_OpenUSBPort(255);
PTK_ClearBuffer();
PTK_SetPrintSpeed(3);
PTK_DrawTextEx(100, 100, 0, 6, 3, 3, 'N', _T("ABC123"), FALSE);
PTK_PrintLabel(1, 1);
PTK_CloseUSBPort();
```

### *PTK\_SetDarkness*

Description: sets the current print darkness. Not preserved across power cycles.

Note:

- 1, If the print darkness is not set, the default is 10.**
- 2, If the print darkness has been set to a nonzero value via FLASH (for example, via the LCD screen), this API setting has no effect.**
- 3. The PTK\_EnableFLASH function is only used to save forms and images; it has no effect on this API.**

Prototype:

```
int _stdcall PTK_SetDarkness(unsigned int dark);
```

Parameters:

dark: label print darkness. Range: 0~20. The larger the value, the darker the print result.

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

---

```
PTK_OpenUSBPort(255);
PTK_ClearBuffer();
PTK_SetDarkness(12);
PTK_DrawTextEx(100, 100, 0, 6, 3, 3, 'N', _T("ABC123"), FALSE);
PTK_PrintLabel(1, 1);
PTK_CloseUSBPort();
```

### *PTK\_SetDirection*

Description: sets the current print direction for the label. Not preserved across power cycles.

Note:

- 1. If the printer direction has been set via FLASH, the direction set by this API function takes precedence until power is lost.**
- 2. The PTK\_EnableFLASH function is only used to save forms and images; it has no effect on this API.**

Prototype:

```
int _stdcall PTK_SetDirection(TCHAR direct);
```

Parameters:

direct: direction. Value: B or T.

T -> print normally starting from the top-left corner of the label. See: Coordinate system of a barcode label printer.

B -> print starting from the bottom-right corner of the label, opposite direction to T;

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
PTK_OpenUSBPort(255);
PTK_ClearBuffer();
PTK_SetDirection('B');
PTK_DrawTextEx(100, 100, 0, 6, 3, 3, 'N', _T("ABC123"), FALSE);
PTK_PrintLabel(1, 1);
PTK_CloseUSBPort();
```



---

### *PTK\_SetLabelHeight*

Description: sets the current label height, the height of the positioning gap / black line / notch, and the positioning offset. Not preserved across power cycles.

Note:

- 1. If not set, the height measured by label calibration is used.**
- 2. The PTK\_EnableFLASH function is only used to save forms and images; it has no effect on this API.**

Prototype:

```
int _stdcall PTK_SetLabelHeight(unsigned int lheight, unsigned int gapH, int gapOffset, BOOL bFlag);
```

Parameters:

lheight: label height, in dots. Range: 0-65535;

gapH: height of the positioning gap / black line / notch between labels, in dots,  
Range: 0-65535;

The value of gapH depends on the label positioning mode:

Gap mode (GAP MODE): default mode. gapH is set to the height of the gap,

Notch positioning is a special case of gap mode and is also classified as gap mode;

Black line mode (BLACK LINE MODE): gapH is set to the height of the black line;

Continuous mode (CONTINUOUS MODE): gapH is set to 0. In this mode the media sensor only detects whether media has run out.

gapOffset: positioning offset for the gap / black line / notch, in dots.

bFlag: whether gapOffset is effective. TRUE – effective; FALSE – ineffective.

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
PTK_OpenUSBPort(255);  
PTK_ClearBuffer();  
PTK_SetLabelHeight(160, 24, 0, FALSE);  
PTK_DrawTextEx(100, 100, 0, 6, 3, 3, 'N', _T("ABC123"), FALSE);  
PTK_PrintLabel(1, 1);  
PTK_CloseUSBPort();
```

### *PTK\_SetLabelWidth*

Description: sets the current label width. Not preserved across power cycles.

---

Note: The PTK\_EnableFLASH function only applies to saving forms and images; it has no effect on this API.

**Prototype:**

```
int _stdcall PTK_SetLabelWidth(unsigned int lwidth);
```

**Parameters:**

lwidth: label width, in dots.

**Return Value:**

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

**Example:**

```
PTK_OpenUSBPort(255);
PTK_ClearBuffer();
PTK_SetLabelWidth(1228);
PTK_DrawTextEx(100, 100, 0, 6, 3, 3, 'N', _T("ABC123"), FALSE);
PTK_PrintLabel(1, 1);
PTK_CloseUSBPort();
```

*PTK\_PrintLabel*

Description: instructs the printer to start printing the label content.

Note: After setting up the print content, this function must be called last to start printing.

**Prototype:**

```
int _stdcall PTK_PrintLabel(unsigned int number, unsigned int cpnumber);
```

**Parameters:**

number: number of labels to print. Range: 1-65535;

cpnumber: number of copies of each label. Range: 1-65535;

**Note:**

cpnumber may be used together with serial numbers; see the example.

**Return Value:**

0 -> OK

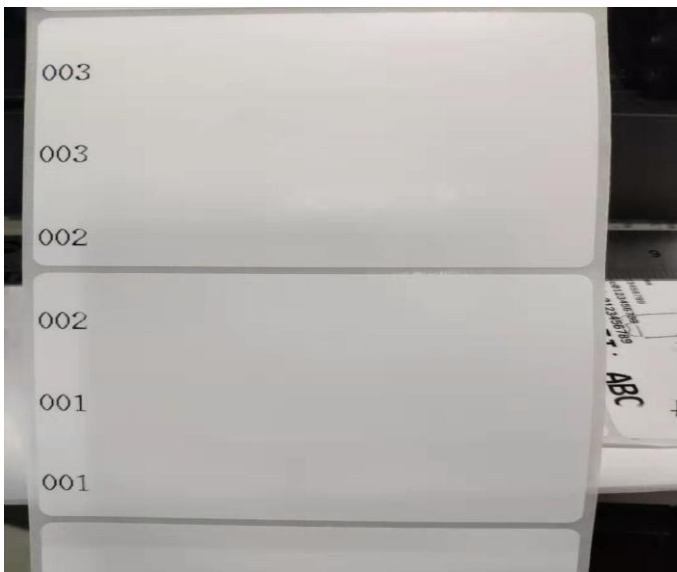
For other return values, call PTK\_GetErrorInfo to parse them.

**Example:**

---

```
PTK_OpenUSBPort(255);
PTK_ClearBuffer();
PTK_SetLabelHeight(200, 0, 0, FALSE); // Set the label to continuous media with a height of 200
dots
PTK_DefineCounter(0, 6, 'N', _T("+1"), _T("\nEnter\Code:")); // Define serial number C0
PTK_DrawTextEx(20, 20, 0, 6, 2, 2, 'N', _T("C0"), TRUE); // Print C0
PTK_Download();
PTK_DownloadInitVar(_T("001")); // Initialize C0
PTK_PrintLabel(3, 2); // Start printing labels: print 3 labels, each duplicated 2 times
PTK_CloseUSBPort();
```

Print result:



### *[PTK\\_PrintLabelFeedback](#)*

Description: instructs the printer to start printing the label content, then reads the printer's current status after printing completes.

Note: This function is only supported on firmware V7.60 and later, and must not be used together with PTK\_PrintLabel.

Prototype:

```
int _stdcall PTK_PrintLabelFeedback(LPTSTR data, DWORD dataSize);
```

Parameters:

data: buffer that stores the printer status code.

Note: The return value is formatted as wlxxxx, where xxxx is the printer status code.

dataSize: size of the buffer allocated for data.

---

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
TCHAR buff[1024] = { 0 };  
PTK_OpenUSBPort(255);  
PTK_PrintLabelFeedback(buff, sizeof(buff));  
MessageBox(buff);  
PTK_CloseUSBPort();
```

---

## Prints text

### *PTK\_DrawText*

Description: prints one line of text.

Note: PTK\_PrintLabel must be called last to start printing.

Prototype:

```
int _stdcall PTK_DrawText(unsigned int px, unsigned int py, unsigned int pdirec, unsigned int pFont,
    unsigned int pHorizontal, unsigned int pVertical, TCHAR pColor, LPCTSTR pstr);
```

Parameters:

px: set the X coordinate, in dots.

py: set the Y coordinate, in dots.

pdirec: select the print direction of the text. 0 - no rotation; 1 - rotate 90 deg; 2 - rotate 180 deg; 3 - rotate 270 deg.

pFont: select font.

Note: With the Windows DLL, PTK\_DrawText\_TrueType or PTK\_DrawText\_TrueTypeEx can use the fonts in C:\Windows\Fonts directly without downloading them.

p4	Description
1	Built-in dot-matrix Western font 1
2	Built-in dot-matrix Western font 2
3	Built-in dot-matrix Western font 3
4	Built-in dot-matrix Western font 4
5	Built-in dot-matrix Western font 5
6	Built-in 24x24 dot-matrix Chinese font
7	Built-in Chinese TrueType font
8	Built-in Western TrueType font
A~Z	User-downloaded TrueType font

pHorizontal: when pFont is set to a built-in font (1~6), pHorizontal sets the horizontal scaling factor for the dot matrix. Range: 1-24.

When pFont is set to a TrueType font (A~Z), pHorizontal sets the font width in pixels (no maximum).

Note: On POSTEK V6 series printers, when pFont is set to 6 (built-in Chinese font), pHorizontal sets the font width in pixels (no maximum), as with TrueType fonts.

pVertical: when pFont is set to a built-in font (1~6), pVertical sets the vertical scaling factor for the dot matrix. Range: 1-24.

When pFont is set to a TrueType font (A~Z), pVertical sets the font height in pixels (no maximum).

Note: On POSTEK V6 series printers, when pFont is set to 6 (built-in Chinese font), pVertical sets the font height in pixels (no maximum), as with TrueType fonts.

pColor: 'N' (ASCII value 78) prints normal text (e.g. black text on a white background),

'R' (ASCII value 82) prints reverse-color text (e.g. white text on a black background).

---

pstr: a string of 1-100 characters.

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
PTK_OpenUSBPort(255);
PTK_ClearBuffer();
PTK_DrawText(20, 20, 0, 6, 2, 2, 'N', _T("Test print text 6,ABC123"));
PTK_RenameDownloadFont(1, 'A', _T("arial")); // Can only be used after the arial font has been
downloaded through the utility; renames arial to printer font A
PTK_DrawText(20, 120, 0, 'A', 48, 48, 'N', _T("TrueType"));
PTK_PrintLabel(1, 1); // Start printing labels
PTK_CloseUSBPort();
```

### *PTK\_DrawText\_MultiLine*

Description: prints one line of text using the printer's built-in fonts; supports automatic line wrapping.

Note: PTK\_PrintLabel must be called last to start printing.

Prototype:

```
int _stdcall PTK_DrawText_MultiLine(unsigned int x, unsigned int y, unsigned int rotation, LPCTSTR
font,
    unsigned int width, unsigned int height, TCHAR reverse, unsigned int boxWidth, unsigned int
linesMax,
    unsigned int lineSpace, TCHAR align, LPCTSTR data);
```

Parameters:

x: set the X coordinate, in dots.

y: set the Y coordinate, in dots.

rotation: select the print direction of the text. 0 - no rotation; 1 - rotate 90 deg;  
2 - rotate 180 deg; 3 - rotate 270 deg.

font: select font.

Note: With the Windows DLL, PTK\_DrawText\_TrueType or PTK\_DrawText\_TrueTypeEx can use the fonts in C:\Windows\Fonts directly without downloading them.

p4	Description
1	Built-in dot-matrix Western font 1
2	Built-in dot-matrix Western font 2
3	Built-in dot-matrix Western font 3
4	Built-in dot-matrix Western font 4
5	Built-in dot-matrix Western font 5
6	Built-in 24x24 dot-matrix Chinese font
7	Built-in Chinese TrueType font

---

8	Built-in Western TrueType font
A~Z	User-downloaded TrueType font

width: when font is set to a built-in font (1~6), width sets the horizontal scaling factor for the dot matrix. Range: 1-24. When font is set to a TrueType font (A~Z), width sets the font width in pixels (no maximum).

Note: On POSTEK V6 series printers, when font is set to 6 (built-in Chinese font), width sets the font width in pixels (no maximum), as with TrueType fonts.

height: when font is set to a built-in font (1~6), height sets the vertical scaling factor for the dot matrix. Range: 1-24.

When font is set to a TrueType font (A~Z), height sets the font height in pixels (no maximum).

Note: On POSTEK V6 series printers, when font is set to 6 (built-in Chinese font), height sets the font height in pixels (no maximum), as with TrueType fonts.

reverse: text type.

'N' - print normal text (e.g. black text on a white background);

'R' - print reverse-color text (e.g. white text on a black background).

boxWidth: text-box width, in dots. Range: 0~65535.

linesMax: maximum number of lines in the text box. Range: 0~65535.

lineSpace: line-spacing offset, in dots. Range: -9999~9999.

align: alignment mode. (This parameter is currently ignored.)

'L' - left aligned (default);

'C' - center aligned;

'R' - right aligned.

data: the character data to be printed.

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
PTK_OpenUSBPort(255);
PTK_ClearBuffer();
PTK_SetLabelHeight(500, 0, 0, FALSE); // Set the label to continuous media with a height of 300
dots
PTK_DrawText_MultiLine (20, 20, 0, 7, 30, 30, 'N', 240, 50, 'L', _T("Test auto-wrap print text
7, ABCEFGHIJKLMNOP, 124567890-==++;\nNext line"));
PTK_PrintLabel(1, 1); // Start printing labels
PTK_CloseUSBPort();
```

---

## *PTK\_DrawTextEx*

Description: prints one line of text, serial number or variable.

Note: Use together with the form-related printing functions.

Note: PTK\_PrintLabel must be called last to start printing.

### Prototype:

```
int _stdcall PTK_DrawTextEx(unsigned int px, unsigned int py, unsigned int pdirec, unsigned int pFont, unsigned int pHorizontal, unsigned int pVertical, TCHAR pColor, LPCTSTR pstr, BOOL Variable);
```

### Parameters:

px: set the X coordinate, in dots.

py: set the Y coordinate, in dots.

pdirec: select the print direction of the text. 0 - no rotation; 1 - rotate 90 deg; 2 - rotate 180 deg; 3 - rotate 270 deg.

pFont: select a built-in font or a soft font.

1-5: the five built-in Western fonts of the printer;

6: the printer's built-in Chinese font;

On printers other than the POSTEK V6 series, 6 represents the printer's built-in 24x24 Simplified Chinese font;

When the printer is a POSTEK V6 series printer, 6 indicates the printer's built-in SimHei font.

'A'-'Z': downloaded soft fonts. (Downloaded through the utility software.)

Note: With the Windows DLL, PTK\_DrawText\_TrueType or PTK\_DrawText\_TrueTypeEx can use the fonts in C:\Windows\Fonts directly without downloading them.

Value	Description
1	Western font 1
2	Western font 2
3	Western font 3
4	Western font 4
5	Western font 5 (uppercase letters only)
6	Chinese font
'A'~'Z'	Soft font

pHorizontal: when pFont is set to a built-in font (1~6), pHorizontal sets the horizontal scaling factor for the dot matrix. Range: 1-24.

When pFont is set to a TrueType font (A~Z), pHorizontal sets the font width in pixels (no maximum).

Note: On POSTEK V6 series printers, when pFont is set to 6 (built-in Chinese font), pHorizontal sets the font width in pixels (no maximum), as with TrueType fonts.

pVertical: when pFont is set to a built-in font (1~6), pVertical sets the vertical scaling factor for the dot matrix. Range: 1-24.



---

When pFont is set to a TrueType font (A~Z), pVertical sets the font height in pixels (no maximum).

**Note:** On POSTEK V6 series printers, when pFont is set to 6 (built-in Chinese font), pVertical sets the font height in pixels (no maximum), as with TrueType fonts.

pColor: 'N' (ASCII value 78) prints normal text (e.g. black text on a white background),

'R' (ASCII value 82) prints reverse-color text (e.g. white text on a black background).

pstr: a string of 1-100 characters. Users may freely combine "DATA", Cn and Vn into a composite string,

"DATA": a constant string. The double quote (") must be used as the start and end markers, e.g. "POSTEK Printer".

Cn: serial number value. This serial number must have been previously defined.

Vn: variable string. This variable string must have been previously defined.

Example: "text1"Cn"text2"Vn.

R: indicates that the content to print is the current UHF RFID tag TID (64-bit) data. (Note: this feature is only supported on RFID printers.)

Variable: TRUE indicates that the current string contains a variable operation; \" must be added to enclose the constant strings to be printed.

For example: PTK\_DrawTextEx (50,30,0,2,1,1,'N','\"123456789\"C0\",TRUE);

FALSE indicates that the current string does not contain a variable operation and that \" need not be added.

For example, PTK\_DrawTextEx (50,30,0,2,1,1,'N',' 123456789\",FALSE); and

PTK\_DrawText(50,30,0,2,1,1,'N',' 123456789\") have the same effect;

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

**Example:**

```
PTK_OpenUSBPort(255);
```

```
PTK_ClearBuffer();
```

```
PTK_SetLabelHeight(500, 0, 0, FALSE); // Set the label to continuous media with a height of 300 dots
```

```
PTK_DefineCounter(0, 6, 'N', _T("+1"), _T("\\Enter\\Code:")); // Define serial number C0
```

```
PTK_DefineCounter(1, 6, 'N', _T("+3"), _T("\\Enter\\Code:")); // Define serial number C1
```

```
PTK_DrawTextEx(20, 20, 0, 6, 2, 2, 'N', _T("\\Test serial number 0\\C0"), TRUE); // Print C0
```

```
PTK_DrawTextEx(20, 120, 0, 6, 2, 2, 'N', _T("\\Test serial number 1\\C1"), TRUE); // Print C1
```

```
PTK_DrawTextEx(20, 220, 0, 6, 2, 2, 'N', _T("Test 2 plain text"), FALSE); // Print plain text
```

```
PTK_Download();
```

```
PTK_DownloadInitVar(_T("111")); // Initialize C0
```

```
PTK_DownloadInitVar(_T("222")); // Initialize C1
```

```
PTK_PrintLabel(3, 1); // Start printing labels
```

---

```
PTK_CloseUSBPort();
```

Print result:



### *PTK\_DrawText\_TrueType*

Description: prints one line of TrueType-font text using the Windows font library.

Note: PTK\_PrintLabel must be called last to start printing.

Prototype:

```
int _stdcall PTK_DrawText_TrueType(unsigned int x, unsigned int y, unsigned int FHeight, unsigned int FWidth, LPCTSTR FType, unsigned int Fspin, unsigned int FWeight, BOOL FItalic, BOOL FUnline, BOOL FStrikeOut, LPCTSTR data);
```

Parameters:

x: set the X coordinate, in dots;

y: set the Y coordinate, in dots;

FHeight: font height, in dots;

FWidth: font width, in dots;

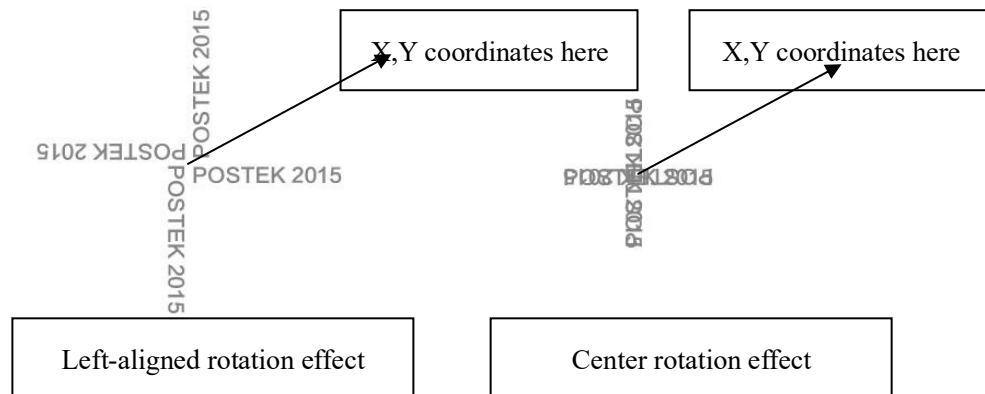
**\* To print a font at normal proportions, set FWidth to 0;**

FType: font name;

Fspin: text rotation angle. 1 -> left-aligned 0 deg, 2 -> left-aligned 90 deg, 3 -> left-aligned 180 deg, 4 -> left-aligned 270 deg.

---

5 -> centered 0 deg, 6 -> centered 90 deg, 7 -> centered 180 deg, 8 -> centered 270 deg



Fweight: font weight.

0 and 400 -> 400 Normal,  
100 -> Very Thin, 200 -> Extra Thin,  
300 -> Thin , 500 -> Medium,  
600 -> Semi Bold , 700 -> Bold ,  
800 -> Extra Bold , 900 -> Black.

Fitalic: italic. 0 -> FALSE, 1 -> TRUE;

Funderline: underline text. 0 -> FALSE, 1 -> TRUE;

FstrikeOut: strikethrough text. 0 -> FALSE, 1 -> TRUE;

data: string content.

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
PTK_OpenUSBPort(255);  
PTK_ClearBuffer();  
PTK_DrawText_TrueType(50, 50, 48, 0, _T("SimHei"), 1, 400, FALSE, FALSE, FALSE, _T("POSTEK  
SimHei"));  
PTK_DrawText_TrueType(50, 150, 48, 0, _T("SimSun"), 1, 400, FALSE, FALSE, FALSE, _T("POSTEK  
SimSun"));  
PTK_PrintLabel(1, 1); // Start printing labels  
PTK_CloseUSBPort();
```

---

### *PTK\_DrawText\_TrueType\_MultiLine*

Description: prints one line of TrueType-font text using the Windows font library.

Note: PTK\_PrintLabel must be called last to start printing.

Prototype:

```
int _stdcall PTK_DrawText_TrueType_MultiLine(unsigned int x, unsigned int y, unsigned int FHeight,
unsigned int FWidth, LPCTSTR FType, unsigned int Fspin, unsigned int FWeight, BOOL FItalic, BOOL
FUnline, BOOL FStrikeOut, unsigned int lineMaxWidth, unsigned int alignment, LPCTSTR data)
```

Parameters:

x: set the X coordinate, in dots;

y: set the Y coordinate, in dots;

FHeight: font height, in dots;

FWidth: font width, in dots;

**\* To print a font at normal proportions, set FWidth to 0;**

FType: font name;

Fspin: text rotation angle. 1 -> left-aligned 0 deg, 2 -> left-aligned 90 deg, 3 -> left-aligned 180 deg, 4 -> left-aligned 270 deg.

5 -> centered 0 deg, 6 -> centered 90 deg, 7 -> centered 180 deg, 8 -> centered 270 deg

Fweight: font weight.

0 and 400 -> 400 Normal,

100 -> Very Thin, 200 -> Extra Thin,

300 -> Thin , 500 -> Medium,

600 -> Semi Bold , 700 -> Bold ,

800 -> Extra Bold , 900 -> Black.

Fitalic: italic. 0 -> FALSE, 1 -> TRUE;

Funline: underline text. 0 -> FALSE, 1 -> TRUE;

FstrikeOut: strikethrough text. 0 -> FALSE, 1 -> TRUE;

lineMaxWidth: maximum width of a single line of text (in pixels). When text exceeds this width it wraps automatically. The parameter type is a positive integer;

alignment: text alignment option. Alignment is based on a text box of width lineMaxWidth, not on the entire label width. Parameter values and descriptions are as follows:

0 -> Left aligned, 1 -> Horizontal center, 2 -> Right aligned, 3 -> Left aligned (truncate English words at line breaks), 4 -> Horizontal center (truncate English words at line breaks), 5 -> Right aligned (truncate English words at line breaks);

data: string content.

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

---

### Example:

```
PTK_OpenUSBPort(255);
PTK_ClearBuffer();
PTK_DrawText_TrueType_MultiLine (50, 50, 48, 0, _T("SimHei"), 1, 400, FALSE, FALSE, FALSE, 250, 3,
_T("POSTEK SimHei"));
PTK_PrintLabel(1, 1); // Start printing labels
PTK_CloseUSBPort();
```

### *PTK\_RenameDownloadFont*

**Description:** maps a font downloaded to the printer to one of the A-Z font IDs used by PTK\_DrawText.

**Note:** The names of the fonts downloaded to the printer can be retrieved with PTK\_GetStorageList.

### Prototype:

```
int _stdcall PTK_RenameDownloadFont(unsigned int StoreType, TCHAR Fontname, LPTSTR DownloadFontName);
```

### Parameters:

**StoreType:** storage location for the downloaded font in the printer. 0: SDRAM, 1: FLASH.

**Tip:** Fonts downloaded to the printer's SDRAM are erased when the printer powers off; fonts downloaded to FLASH are preserved across power cycles.

**Fontname:** ID used to rename the downloaded font. Range: A-Z.

**DownloadFontName:** the name of the downloaded font as stored in the printer.

### Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

### Example:

```
PTK_OpenUSBPort(255);
PTK_ClearBuffer();
PTK_DrawText(20, 20, 0, 6, 2, 2, 'N', _T("Test print text 6,ABC123"));
PTK_RenameDownloadFont(1, 'A', _T("arial")); // Can only be used after the arial font has been
downloaded through the utility; renames arial to printer font A
PTK_DrawText(20, 120, 0, 'A', 48, 48, 'N', _T("TrueType"));
PTK_PrintLabel(1, 1); // Start printing labels
PTK_CloseUSBPort();
```

---

## Prints an image

### *PTK\_ChangeIMGtoPCX*

Description: converts an image to PCX format and generates a PCX file with the same name in the same directory.

Note: This API call does not require an open port.

#### Prototype:

```
int _stdcall PTK_ChangeIMGtoPCX(LPTSTR filePath);
```

#### Parameters:

filePath: path of the image to be converted.

#### Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

#### Example:

```
PTK_ChangeIMGtoPCX(_T("Koala.jpg"));
```

### *PTK\_PcxGraphicsList*

Description: prints a list of the names of the images stored in the printer's RAM or FLASH memory.

Note: This function does not require PTK\_PrintLabel to be called in order to print.

#### Prototype:

```
int _stdcall PTK_PcxGraphicsList(void);
```

#### Parameters: none

#### Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

#### Example:

```
PTK_OpenUSBPort(255);  
PTK_PcxGraphicsList();  
PTK_CloseUSBPort();
```

---

### *PTK\_PcxGraphicsDel*

Description: deletes an image stored on the printer.

Note: If the image does not exist in the printer, this API call has no effect.

Prototype:

```
int _stdcall PTK_PcxGraphicsDel(LPTSTR pcxname);
```

Parameters:

pcxname: the name under which the image is stored inside the printer. Maximum length: 16 characters.

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
PTK_OpenUSBPort(255);  
PTK_PcxGraphicsDel(_T("P1"));  
PTK_CloseUSBPort();
```

### *PTK\_AnyGraphicsDownload*

Description: downloads an image to the printer from a file path.

Prototype:

```
int _stdcall PTK_AnyGraphicsDownload(LPTSTR pcxname, LPTSTR filePath, float ratio,  
    unsigned int width, unsigned int height, unsigned int iDire);
```

**Note: If FLASH is not enabled, the image is not preserved after power-off.**

Parameters:

pcxname: the name under which the image is stored inside the printer. Maximum length: 16 characters;

After the image has been stored on the printer, this name must be used with

PTK\_DrawPcxGraphics to retrieve and print the image.

filePath: path to the image file (HTTP URLs are supported). Currently supported formats: bmp, jpg, png, tif, ico, pcx.

Note: When the path is a URL, it must begin with "http://" or "https://".

ratio: scaling factor. width and height only take effect when this parameter is 0.

width: the width after scaling (in dots). Pass 0 to keep the original width.

height: the height after scaling (in dots). Pass 0 to keep the original height.

iDire: rotation angle.

0- 0°

1- 90°

2- 180°

3- 270°

---

4 - vertical mirror flip

5 - horizontal mirror flip

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
PTK_OpenUSBPort(255);  
//PTK_EnableFLASH();// Enable FLASH storage  
PTK_PcxGraphicsDel(_T("P1"));  
PTK_AnyGraphicsDownload(_T("P1"), _T("koala.jpg"), 1, 0, 0, 0);  
PTK_AnyGraphicsDownload(_T("P2"), _T("http://www.postek.com.cn/cn/skins/images/logo.jpg"), 1, 0, 0,  
0);  
//PTK_DisableFLASH();// Disable FLASH storage  
PTK_CloseUSBPort();
```

### *PTK\_DrawPcxGraphics*

Description: prints an image stored on the printer.

Note: The image names stored on the printer can be retrieved with PTK\_GetStorageList or PTK\_PcxGraphicsList.

Note: PTK\_PrintLabel must be called last to start printing.

Prototype:

```
int _stdcall PTK_DrawPcxGraphics(unsigned int px, unsigned int py, LPTSTR pcxname);
```

Parameters:

px: set the X coordinate, in dots;

py: set the Y coordinate, in dots;

pcxname: the name under which the image is stored inside the printer. Maximum length: 16 characters;

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:



---

```
PTK_OpenUSBPort(255);
//PTK_EnableFLASH();// Enable FLASH storage
PTK_PcxGraphicsDel(_T("P1"));
PTK_AnyGraphicsDownload(_T("P1"), _T("koala.jpg"), 0.5, 0, 0, 0);
//PTK_DisableFLASH();// Disable FLASH storage
PTK_DrawPcxGraphics(0, 0, _T("P1"));
PTK_DrawPcxGraphics(0, 500, _T("P1"));
PTK_PrintLabel(1, 1);
PTK_CloseUSBPort();
```

### *PTK\_AnyGraphicsPrint*

Description: prints an image directly from a file path.

Note: PTK\_PrintLabel must be called last to start printing.

#### Prototype:

```
int _stdcall PTK_AnyGraphicsPrint(unsigned int px, unsigned int py, LPTSTR pcxname, LPTSTR filePath,
    float ratio, unsigned int width, unsigned int height, unsigned int iDire);
```

#### Parameters:

px: set the X coordinate, in dots;

py: set the Y coordinate, in dots;

pcxname: the name under which the image is stored inside the printer. Maximum length: 16 characters;

filePath: path to the image file (HTTP URLs are supported). Currently supported formats: bmp, jpg, png, tif, ico, pcx.

Note: When the path is a URL, it must begin with "http://" or "https://".

ratio: scaling factor. width and height only take effect when this parameter is 0.

width: the width after scaling (in dots). Pass 0 to keep the image's original width.

height: the height after scaling (in dots). Pass 0 to keep the image's original height.

iDire: rotation angle.

0 - 0 deg

1 - 90 deg

2 - 180 deg

3 - 270 deg

4 - vertical mirror flip

5 - horizontal mirror flip

#### Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

#### Example:

---

```

PTK_OpenUSBPort(255);
PTK_SetDirection('B');
//PTK_EnableFLASH();// Enable FLASH storage
PTK_AnyGraphicsPrint(0, 0, _T("P1"), _T("Koala.jpg"),0.5, 0, 0, 0);// Performs delete, download
and print in one call
PTK_AnyGraphicsPrint(0, 400, _T("P2"), _T("http://www.postek.com.cn/cn/skins/images/logo.jpg"),1, 0,
0, 0);
//PTK_DisableFLASH();// Disable FLASH storage
PTK_DrawPcxGraphics(0, 500, _T("P1"));// Print image P1
PTK_DrawPcxGraphics(600, 0, _T("P2"));// Print image P2
PTK_PrintLabel(1, 1);// Prints two copies of P1 and two copies of P2 in total
PTK_CloseUSBPort();

```

### *PTK\_AnyGraphicsDownloadFromMemory*

Description: stores an image on the printer from image data.

Note: This function requires the correct image type, image size and image data; otherwise the store will fail.

Prototype:

```

int _stdcall PTK_AnyGraphicsDownloadFromMemory(LPTSTR pcxname, unsigned int imageType, unsigned int
imageSize, float ratio, unsigned int width, unsigned int height, unsigned int iDire, unsigned char*
imageBuffer);

```

Parameters:

pcxname: the name under which the image is stored inside the printer. Maximum length: 16 characters;

imageType: image format. Currently supported formats:

1:bmp      3:jpg      4:png      5:ico      6:tif      8:pcx

imageSize: size of the image file.

ratio: scaling factor. width and height only take effect when this parameter is 0.

width: the width after scaling (in dots). Pass 0 to keep the image's original width.

height: the height after scaling (in dots). Pass 0 to keep the image's original height.

iDire: rotation angle.

0 - 0 deg

1 - 90 deg

2 - 180 deg

3 - 270 deg

4 - vertical mirror flip

5 - horizontal mirror flip

imageBuffer: image data to be printed.

---

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
unsigned char imagebuff[1024 * 1024] = { 0 };// Defined globally

int imagesize = 0;
FILE * fp = NULL;
char file_name[256] = "Koala.jpg";

PTK_OpenUSBPort(255);
/* Read the image data into memory */
fopen_s(&fp, file_name, "rb");
imagesize = fread(imagebuff, sizeof(unsigned char), sizeof(imagebuff) - 1, fp);
fclose(fp);
/* Store the image into the printer */
PTK_PcxGraphicsDel(_T("P1"));
PTK_AnyGraphicsDownloadFromMemory(_T("P1"), 3, imagesize, 1, 0, 0, 0, imagebuff);
PTK_DrawPcxGraphics(0, 0, _T("P1"));
PTK_PrintLabel(1, 1);
PTK_CloseUSBPort();
```

### *PTK\_AnyGraphicsPrintFromMemory*

Description: prints an image from image data.

Note: This function requires the correct image type, image size and image data; otherwise the store will fail.

Note: PTK\_PrintLabel must be called last to start printing.

Prototype:

```
int _stdcall PTK_AnyGraphicsPrintFromMemory(int px, int py, LPTSTR pcxname, unsigned int imageType,
unsigned int imageSize, float ratio, unsigned int width, unsigned int height, unsigned int iDire,
unsigned char* imageBuffer);
```

Parameters:

px: set the X coordinate, in dots;

py: set the Y coordinate, in dots;

pcxname: the name under which the image is stored inside the printer. Maximum length: 16 characters;

imageType: image format. Currently supported formats:

1:bmp          3:jpg          4:png          5:ico          6:tif          8:pcx

imageSize: size of the image file.

ratio: scaling factor. width and height only take effect when this parameter is 0.

---

width: the width after scaling (in dots). Pass 0 to keep the image's original width.  
height: the height after scaling (in dots). Pass 0 to keep the image's original height.  
iDire: rotation angle.  
0 - 0 deg  
1 - 90 deg  
2 - 180 deg  
3 - 270 deg  
4 - vertical mirror flip  
5 - horizontal mirror flip  
imageBuffer: image data to be printed.

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
unsigned char imagebuff[1024 * 1024] = { 0 };// Defined globally

int imagesize = 0;
FILE * fp = NULL;
char file_name[256] = "Koala.jpg";

PTK_OpenUSBPort(255);
/* Read the image data into memory */
fopen_s(&fp, file_name, "rb");
imagesize = fread(imagebuff, sizeof(unsigned char), sizeof(imagebuff) - 1, fp);
fclose(fp);
/* Print the image */
PTK_AnyGraphicsPrintFromMemory(0, 0, _T("P1"), 3, imagesize, 1, 0, 0, 0, imagebuff);
PTK_PrintLabel(1, 1);
PTK_CloseUSBPort();
```

### *PTK\_AnyGraphicsPrint\_Base64*

Description: prints an image from base64-encoded image data.

Note: PTK\_PrintLabel must be called last to start printing.

Prototype:

```
int __stdcall PTK_AnyGraphicsPrintFromMemory(int px, int py, unsigned int imageType, float ratio,
unsigned int width, unsigned int height, unsigned int iDire, LPCTSTR imageBuffer);
```

Parameters:

px: set the X coordinate, in dots;

---

py: set the Y coordinate, in dots;  
imageType: image format. Currently supported formats:  
1:bmp      3:jpg      4:png      5:ico      6:tif      8:pcx  
ratio: scaling factor. width and height only take effect when this parameter is 0.  
width: the width after scaling (in dots). Pass 0 to keep the image's original width.  
height: the height after scaling (in dots). Pass 0 to keep the image's original height.  
iDire: rotation angle.  
0 - 0 deg  
1 - 90 deg  
2 - 180 deg  
3 - 270 deg  
4 - vertical mirror flip  
5 - horizontal mirror flip  
imageBuffer: base64-encoded image data to be printed.

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
PTK_OpenUSBPort(255);  
PTK_AnyGraphicsPrint_Base64(10, 10, 3, 1, 0, 0, 0, image_base64);  
PTK_PrintLabel(1, 1);  
PTK_CloseUSBPort();
```

**Prints a dot-matrix image**

*PTK\_BinGraphicsList*

Description: prints a list of the names of the images stored in the printer's RAM or FLASH memory.

Note: Behaves the same as PTK\_PcxGraphicsList.

Prototype:

```
int _stdcall PTK_BinGraphicsList(void);
```

Parameters:

None

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

---

```
PTK_OpenUSBPort(255);  
PTK_BinGraphicsList();  
PTK_CloseUSBPort();
```

### *PTK\_BinGraphicsDel*

Description: deletes a binary image stored on the printer.

Prototype:

```
int _stdcall PTK_BinGraphicsDel(LPTSTR binname);
```

Parameters:

binname: the name under which the image is stored in the printer. Maximum length: 16 characters.

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
PTK_OpenUSBPort(255);  
PTK_BinGraphicsDel(_T("bo64X64"));  
PTK_PrintLabel(1, 1);
```

### *PTK\_BinGraphicsDownload*

Description: stores a binary-format image on the printer.

Prototype:

```
int _stdcall PTK_BinGraphicsDownload(LPTSTR binname, unsigned int pbyte, unsigned int pH, unsigned  
char * Gdata);
```

Parameters:

binname: the name under which the image is stored in the printer. Maximum length: 16 characters.

pbyte: number of bytes per row of data. If the number of bits in a row is not a multiple of 8, the value is rounded up.

pH: image height, in dots.

Gdata: binary image data. A bit value of 1 prints content, 0 does not print.

Note:

The dot-matrix layout uses row-by-row order, MSB first, negative encoding.

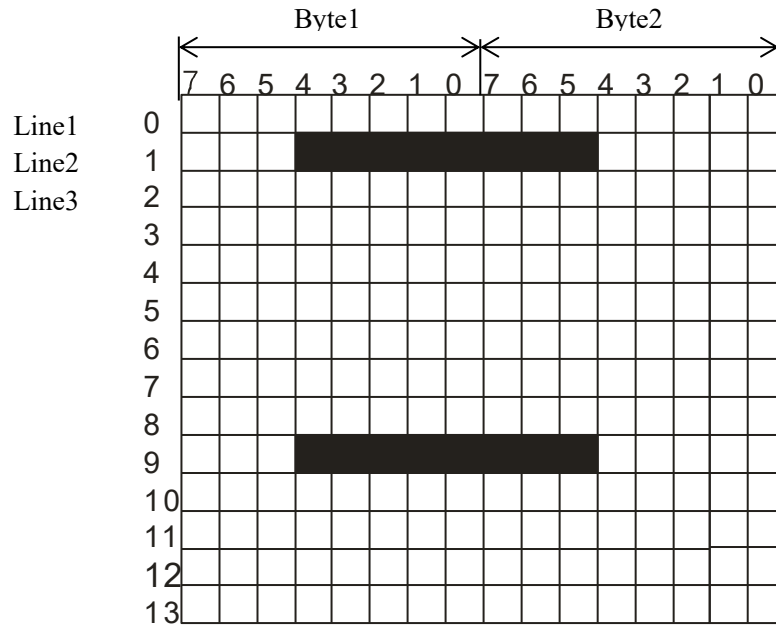
Example:

The data is transmitted in the order: Line1 Byte1 (0x00), Line1 Byte2 (0x00), Line2  
Byte1 (0x1f), Line2 Byte2 (0xe0), Line3 Byte1 (0x00), Line3 Byte2

---

(0x00), ...

The dashed areas are non-image regions; their bit values are 0.



Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
unsigned char font_bo[] =  
{  
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  
    0x00, 0x20, 0x00, 0x00, 0x06, 0x04, 0x00, 0x00, 0x00, 0x18, 0x00, 0x00,  
    0x07, 0x87, 0x00, 0x00, 0x00, 0x1E, 0x00, 0x00, 0x07, 0xC3, 0xC0, 0x00,  
    0x00, 0x1F, 0x00, 0x00, 0x07, 0x81, 0xF0, 0x00, 0x00, 0x1C, 0x00, 0x00,  
    0x07, 0x80, 0xF0, 0x00, 0x00, 0x1C, 0x00, 0x00, 0x07, 0x80, 0x78, 0x00,  
    0x00, 0x1C, 0x00, 0x00, 0x07, 0x80, 0x38, 0x00, 0x00, 0x1C, 0x00, 0x00,  
    0x07, 0x80, 0x31, 0x80, 0x00, 0x1C, 0x00, 0x00, 0x07, 0x80, 0x03, 0xC0,  
    0x00, 0x1C, 0x0F, 0xFF, 0xFF, 0xFF, 0xFF, 0xE0, 0x00, 0x1C, 0x07, 0xFF,  
    0xFF, 0xFF, 0xFF, 0xF0, 0x00, 0x1C, 0x00, 0x00, 0x07, 0x80, 0x00, 0x00,  
    0x00, 0x1C, 0x00, 0x00, 0x07, 0x80, 0x00, 0x00, 0x00, 0x1C, 0x00, 0x00,  
    0x07, 0x80, 0x00, 0x00, 0x00, 0x1C, 0x00, 0xC0, 0x07, 0x80, 0x18, 0x00,  
    0x00, 0x1C, 0x00, 0xFF, 0xFF, 0xFF, 0xFE, 0x00, 0x00, 0x1C, 0x30, 0xFF,  
    0xFF, 0xFF, 0xFE, 0x00, 0x00, 0x1C, 0x78, 0xE0, 0x07, 0x80, 0x3C, 0x00,  
    0x1F, 0xFF, 0xFC, 0xE0, 0x07, 0x80, 0x3C, 0x00, 0x0F, 0xFF, 0xFE, 0xE0,  
    0x07, 0x80, 0x3C, 0x00, 0x00, 0x1C, 0x00, 0xE0, 0x07, 0x80, 0x3C, 0x00,  
    0x00, 0x1C, 0x00, 0xE0, 0x07, 0x80, 0x3C, 0x00, 0x00, 0x1C, 0x00, 0xE0,  
    0x07, 0x80, 0x3C, 0x00, 0x00, 0x1C, 0x00, 0xFF, 0xFF, 0xFF, 0xFC, 0x00,  
}
```

---

```

0x00, 0x1C, 0x00, 0xE0, 0x07, 0x80, 0x3C, 0x00, 0x00, 0x1C, 0x00, 0xE0,
0x07, 0x80, 0x3C, 0x00, 0x00, 0x1C, 0x00, 0xE0, 0x07, 0x80, 0x3C, 0x00,
0x00, 0x1C, 0x00, 0xE0, 0x07, 0x80, 0x3C, 0x00, 0x00, 0x1C, 0x00, 0xE0,
0x07, 0x80, 0x3C, 0x00, 0x00, 0x1C, 0x00, 0xE0, 0x07, 0x80, 0x3C, 0x00,
0x00, 0x1C, 0x00, 0xFF, 0xFF, 0xFF, 0xFC, 0x00, 0x00, 0x1C, 0x00, 0xFF,
0xFF, 0xFF, 0xFC, 0x00, 0x00, 0x1C, 0x00, 0xE0, 0x07, 0x80, 0x3C, 0x00,
0x00, 0x1C, 0x00, 0xE0, 0x07, 0x80, 0x38, 0x00, 0x00, 0x1C, 0x00, 0xE0,
0x07, 0x81, 0xA0, 0x00, 0x00, 0x1C, 0x00, 0xE0, 0x06, 0x01, 0xE0, 0x00,
0x00, 0x1C, 0x00, 0xC0, 0x00, 0x01, 0xF0, 0x00, 0x00, 0x1C, 0x00, 0x00,
0x00, 0x01, 0xE0, 0xC0, 0x00, 0x1C, 0x00, 0x00, 0x00, 0x01, 0xE1, 0xE0,
0x00, 0x1C, 0x00, 0x00, 0x00, 0x01, 0xE3, 0xF0, 0x00, 0x1C, 0x7F, 0xFF,
0xFF, 0xFF, 0xFF, 0xF8, 0x00, 0x1C, 0x3F, 0xFF, 0xFF, 0xFF, 0xFF, 0xFC,
0x00, 0x1C, 0x00, 0x80, 0x00, 0x01, 0xE0, 0x00, 0x00, 0x1C, 0x00, 0x60,
0x00, 0x01, 0xE0, 0x00, 0x00, 0x1C, 0x00, 0x38, 0x00, 0x01, 0xE0, 0x00,
0x00, 0x1C, 0x00, 0x1C, 0x00, 0x01, 0xE0, 0x00, 0x00, 0x1C, 0x00, 0x0F,
0x00, 0x01, 0xE0, 0x00, 0x00, 0x1C, 0x00, 0x0F, 0x80, 0x01, 0xE0, 0x00,
0x00, 0x1C, 0x00, 0x07, 0x80, 0x01, 0xE0, 0x00, 0x00, 0x1C, 0x00, 0x07,
0x80, 0x01, 0xE0, 0x00, 0x00, 0x1C, 0x00, 0x03, 0x80, 0x01, 0xE0, 0x00,
0x00, 0x1C, 0x00, 0x03, 0x80, 0x01, 0xE0, 0x00, 0x00, 0x1C, 0x00, 0x01,
0x00, 0x01, 0xC0, 0x00, 0x00, 0x1C, 0x00, 0x00, 0x00, 0xFF, 0xC0, 0x00,
0x00, 0x1C, 0x00, 0x00, 0x00, 0x3F, 0xC0, 0x00, 0x00, 0x1C, 0x00, 0x00,
0x00, 0x1F, 0xC0, 0x00, 0x00, 0x18, 0x00, 0x00, 0x00, 0x07, 0x80, 0x00,
0x00, 0x20, 0x00, 0x00, 0x00, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
};/* "POSTEK char", 64 * 64 dot matrix */

```

```

PTK_OpenUSBPort(255);
//PTK_EnableFLASH();// Enable FLASH storage
PTK_BinGraphicsDel(_T("bo64X64"));
PTK_BinGraphicsDownload(_T("bo64X64"), 8, 64, font_bo);
//PTK_DisableFLASH();// Disable FLASH storage
PTK_CloseUSBPort();

```

### *PTK\_RecallBinGraphics*

**Description:** prints a binary image stored on the printer.

**Note:** The image names stored on the printer can be retrieved with PTK\_GetStorageList or



---

PTK\_BinGraphicsList.

Note: PTK\_PrintLabel must be called last to start printing.

#### Prototype:

```
int _stdcall PTK_RecallBinGraphics(unsigned int px, unsigned int py, LPTSTR binname);
```

#### Parameters:

px: set the X coordinate, in dots;

py: set the Y coordinate, in dots;

binname: the name under which the image is stored in the printer. Maximum length: 16 characters.

#### Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

#### Example:

```
unsigned char font_bo[] =
{
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x20, 0x00, 0x00, 0x06, 0x04, 0x00, 0x00, 0x00, 0x18, 0x00, 0x00,
    0x07, 0x87, 0x00, 0x00, 0x00, 0x1E, 0x00, 0x00, 0x07, 0xC3, 0xC0, 0x00,
    0x00, 0x1F, 0x00, 0x00, 0x07, 0x81, 0xF0, 0x00, 0x00, 0x1C, 0x00, 0x00,
    0x07, 0x80, 0xF0, 0x00, 0x00, 0x1C, 0x00, 0x00, 0x07, 0x80, 0x78, 0x00,
    0x00, 0x1C, 0x00, 0x00, 0x07, 0x80, 0x38, 0x00, 0x00, 0x1C, 0x00, 0x00,
    0x07, 0x80, 0x31, 0x80, 0x00, 0x1C, 0x00, 0x00, 0x07, 0x80, 0x03, 0xC0,
    0x00, 0x1C, 0x0F, 0xFF, 0xFF, 0xFF, 0xFF, 0xE0, 0x00, 0x1C, 0x07, 0xFF,
    0xFF, 0xFF, 0xFF, 0xF0, 0x00, 0x1C, 0x00, 0x00, 0x07, 0x80, 0x00, 0x00,
    0x00, 0x1C, 0x00, 0x00, 0x07, 0x80, 0x00, 0x00, 0x00, 0x1C, 0x00, 0x00,
    0x07, 0x80, 0x00, 0x00, 0x00, 0x1C, 0x00, 0xC0, 0x07, 0x80, 0x18, 0x00,
    0x00, 0x1C, 0x00, 0xFF, 0xFF, 0xFF, 0xFE, 0x00, 0x00, 0x1C, 0x30, 0xFF,
    0xFF, 0xFF, 0xFE, 0x00, 0x00, 0x1C, 0x78, 0xE0, 0x07, 0x80, 0x3C, 0x00,
    0x1F, 0xFF, 0xFC, 0xE0, 0x07, 0x80, 0x3C, 0x00, 0x0F, 0xFF, 0xFE, 0xE0,
    0x07, 0x80, 0x3C, 0x00, 0x00, 0x1C, 0x00, 0xE0, 0x07, 0x80, 0x3C, 0x00,
    0x00, 0x1C, 0x00, 0xE0, 0x07, 0x80, 0x3C, 0x00, 0x00, 0x1C, 0x00, 0xE0,
    0x07, 0x80, 0x3C, 0x00, 0x00, 0x1C, 0x00, 0xFF, 0xFF, 0xFF, 0xFC, 0x00,
    0x00, 0x1C, 0x00, 0xE0, 0x07, 0x80, 0x3C, 0x00, 0x00, 0x1C, 0x00, 0xE0,
    0x07, 0x80, 0x3C, 0x00, 0x00, 0x1C, 0x00, 0xE0, 0x07, 0x80, 0x3C, 0x00,
    0x00, 0x1C, 0x00, 0xE0, 0x07, 0x80, 0x3C, 0x00, 0x00, 0x1C, 0x00, 0xE0,
    0x07, 0x80, 0x3C, 0x00, 0x00, 0x1C, 0x00, 0xE0, 0x07, 0x80, 0x3C, 0x00,
    0x00, 0x1C, 0x00, 0xFF, 0xFF, 0xFF, 0xFC, 0x00, 0x00, 0x1C, 0x00, 0xFF,
    0xFF, 0xFF, 0xFC, 0x00, 0x00, 0x1C, 0x00, 0xE0, 0x07, 0x80, 0x3C, 0x00,
    0x00, 0x1C, 0x00, 0xE0, 0x07, 0x80, 0x38, 0x00, 0x00, 0x1C, 0x00, 0xE0,
    0x07, 0x81, 0xA0, 0x00, 0x00, 0x1C, 0x00, 0xE0, 0x06, 0x01, 0xE0, 0x00,
```

---

```

0x00, 0x1C, 0x00, 0xC0, 0x00, 0x01, 0xF0, 0x00, 0x00, 0x1C, 0x00, 0x00,
0x00, 0x01, 0xE0, 0xC0, 0x00, 0x1C, 0x00, 0x00, 0x00, 0x01, 0xE1, 0xE0,
0x00, 0x1C, 0x00, 0x00, 0x00, 0x01, 0xE3, 0xF0, 0x00, 0x1C, 0x7F, 0xFF,
0xFF, 0xFF, 0xFF, 0xF8, 0x00, 0x1C, 0x3F, 0xFF, 0xFF, 0xFF, 0xFF, 0xFC,
0x00, 0x1C, 0x00, 0x80, 0x00, 0x01, 0xE0, 0x00, 0x00, 0x1C, 0x00, 0x60,
0x00, 0x01, 0xE0, 0x00, 0x00, 0x1C, 0x00, 0x38, 0x00, 0x01, 0xE0, 0x00,
0x00, 0x1C, 0x00, 0x1C, 0x00, 0x01, 0xE0, 0x00, 0x00, 0x1C, 0x00, 0x0F,
0x00, 0x01, 0xE0, 0x00, 0x00, 0x1C, 0x00, 0x0F, 0x80, 0x01, 0xE0, 0x00,
0x00, 0x1C, 0x00, 0x07, 0x80, 0x01, 0xE0, 0x00, 0x00, 0x1C, 0x00, 0x07,
0x80, 0x01, 0xE0, 0x00, 0x00, 0x1C, 0x00, 0x03, 0x80, 0x01, 0xE0, 0x00,
0x00, 0x1C, 0x00, 0x03, 0x80, 0x01, 0xE0, 0x00, 0x00, 0x1C, 0x00, 0x01,
0x00, 0x01, 0xC0, 0x00, 0x00, 0x1C, 0x00, 0x00, 0x00, 0xFF, 0xC0, 0x00,
0x00, 0x1C, 0x00, 0x00, 0x00, 0x3F, 0xC0, 0x00, 0x00, 0x1C, 0x00, 0x00,
0x00, 0x1F, 0xC0, 0x00, 0x00, 0x18, 0x00, 0x00, 0x00, 0x07, 0x80, 0x00,
0x00, 0x20, 0x00, 0x00, 0x00, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
};/* "POSTEK char", 64 * 64 dot matrix */

```

```

PTK_OpenUSBPort(255);
PTK_BinGraphicsDel(_T("bo64X64"));
PTK_BinGraphicsDownload(_T("bo64X64"), 8, 64, font_bo);
PTK_RecallBinGraphics(50, 50, _T("bo64X64"));
PTK_RecallBinGraphics(50, 150, _T("bo64X64"));
PTK_RecallBinGraphics(150, 50, _T("bo64X64"));
PTK_RecallBinGraphics(150, 150, _T("bo64X64"));
PTK_PrintLabel(1, 1);
PTK_CloseUSBPort();

```

### *PTK\_DrawBinGraphics*

Description: directly defines and prints a binary image.

Note: PTK\_PrintLabel must be called last to start printing.

#### Prototype:

```
int _stdcall PTK_DrawBinGraphics(unsigned int px, unsigned int py, unsigned int pbyte, unsigned int
pH, unsigned char* Gdata);
```

#### Parameters:

px: set the X coordinate, in dots;  
py: set the Y coordinate, in dots;

pbyte: number of bytes per row of data. If the number of bits in a row is not a multiple of 8, the value is rounded up.

pH: image height, in dots.

Gdata: binary image data. A bit value of 1 prints content, 0 does not print.

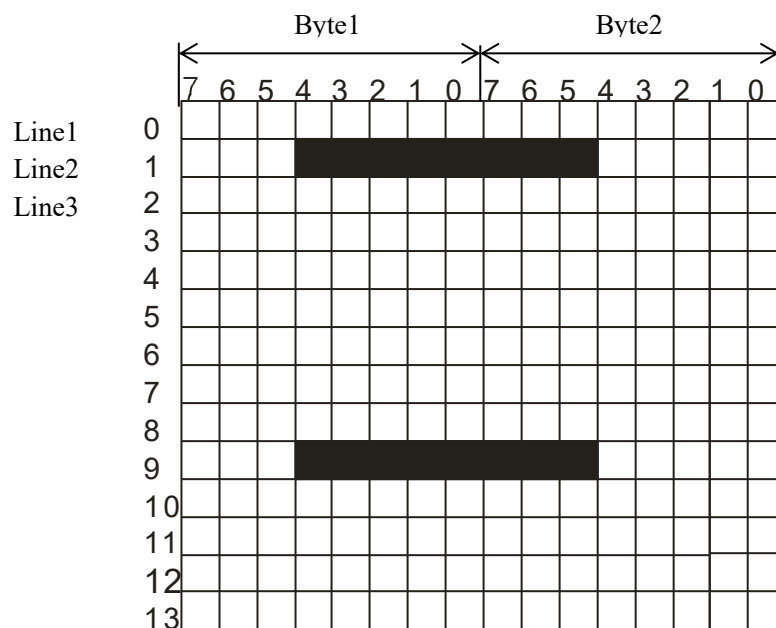
Note:

The dot-matrix layout uses row-by-row order, MSB first, negative encoding.

Example:

The data is transmitted in the order: Line1 Byte1 (0x00), Line1 Byte2 (0x00), Line2 Byte1 (0x1f), Line2 Byte2 (0xe0), Line3 Byte1 (0x00), Line3 Byte2 (0x00), ...

The dashed areas are non-image regions; their bit values are 0.



Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
unsigned char font_bo[] =
{
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x20, 0x00, 0x00, 0x06, 0x04, 0x00, 0x00, 0x00, 0x18, 0x00, 0x00,
    0x07, 0x87, 0x00, 0x00, 0x00, 0x1E, 0x00, 0x00, 0x07, 0xC3, 0xC0, 0x00,
    0x00, 0x1F, 0x00, 0x00, 0x07, 0x81, 0xF0, 0x00, 0x00, 0x1C, 0x00, 0x00,
    0x07, 0x80, 0xF0, 0x00, 0x00, 0x1C, 0x00, 0x00, 0x07, 0x80, 0x78, 0x00,
    0x00, 0x1C, 0x00, 0x00, 0x07, 0x80, 0x38, 0x00, 0x00, 0x1C, 0x00, 0x00,
    0x07, 0x80, 0x31, 0x80, 0x00, 0x1C, 0x00, 0x00, 0x07, 0x80, 0x03, 0xC0,

```

---

```

0x00, 0x1C, 0x0F, 0xFF, 0xFF, 0xFF, 0xFF, 0xE0, 0x00, 0x1C, 0x07, 0xFF,
0xFF, 0xFF, 0xFF, 0xF0, 0x00, 0x1C, 0x00, 0x00, 0x07, 0x80, 0x00, 0x00,
0x00, 0x1C, 0x00, 0x00, 0x07, 0x80, 0x00, 0x00, 0x00, 0x1C, 0x00, 0x00,
0x07, 0x80, 0x00, 0x00, 0x00, 0x1C, 0x00, 0xC0, 0x07, 0x80, 0x18, 0x00,
0x00, 0x1C, 0x00, 0xFF, 0xFF, 0xFF, 0xFE, 0x00, 0x00, 0x1C, 0x30, 0xFF,
0xFF, 0xFF, 0xFE, 0x00, 0x00, 0x1C, 0x78, 0xE0, 0x07, 0x80, 0x3C, 0x00,
0x1F, 0xFF, 0xFC, 0xE0, 0x07, 0x80, 0x3C, 0x00, 0x0F, 0xFF, 0xFE, 0xE0,
0x07, 0x80, 0x3C, 0x00, 0x00, 0x1C, 0x00, 0xE0, 0x07, 0x80, 0x3C, 0x00,
0x00, 0x1C, 0x00, 0xE0, 0x07, 0x80, 0x3C, 0x00, 0x00, 0x1C, 0x00, 0xE0,
0x07, 0x80, 0x3C, 0x00, 0x00, 0x1C, 0x00, 0xFF, 0xFF, 0xFF, 0xFC, 0x00,
0x00, 0x1C, 0x00, 0xE0, 0x07, 0x80, 0x3C, 0x00, 0x00, 0x1C, 0x00, 0xE0,
0x07, 0x80, 0x3C, 0x00, 0x00, 0x1C, 0x00, 0xE0, 0x07, 0x80, 0x3C, 0x00,
0x00, 0x1C, 0x00, 0xFF, 0xFF, 0xFF, 0xFC, 0x00, 0x00, 0x1C, 0x00, 0xFF,
0xFF, 0xFF, 0xFC, 0x00, 0x00, 0x1C, 0x00, 0xE0, 0x07, 0x80, 0x3C, 0x00,
0x00, 0x1C, 0x00, 0xE0, 0x07, 0x80, 0x38, 0x00, 0x00, 0x1C, 0x00, 0xE0,
0x07, 0x81, 0xA0, 0x00, 0x00, 0x1C, 0x00, 0xE0, 0x06, 0x01, 0xE0, 0x00,
0x00, 0x1C, 0x00, 0xC0, 0x00, 0x01, 0xF0, 0x00, 0x00, 0x1C, 0x00, 0x00,
0x00, 0x01, 0xE0, 0xC0, 0x00, 0x1C, 0x00, 0x00, 0x00, 0x01, 0xE1, 0xE0,
0x00, 0x1C, 0x00, 0x00, 0x00, 0x01, 0xE3, 0xF0, 0x00, 0x1C, 0x7F, 0xFF,
0xFF, 0xFF, 0xFF, 0xF8, 0x00, 0x1C, 0x3F, 0xFF, 0xFF, 0xFF, 0xFF, 0xFC,
0x00, 0x1C, 0x00, 0x80, 0x00, 0x01, 0xE0, 0x00, 0x00, 0x1C, 0x00, 0x60,
0x00, 0x01, 0xE0, 0x00, 0x00, 0x1C, 0x00, 0x38, 0x00, 0x01, 0xE0, 0x00,
0x00, 0x1C, 0x00, 0x1C, 0x00, 0x01, 0xE0, 0x00, 0x00, 0x1C, 0x00, 0x0F,
0x00, 0x01, 0xE0, 0x00, 0x00, 0x1C, 0x00, 0x0F, 0x80, 0x01, 0xE0, 0x00,
0x00, 0x1C, 0x00, 0x07, 0x80, 0x01, 0xE0, 0x00, 0x00, 0x1C, 0x00, 0x07,
0x80, 0x01, 0xE0, 0x00, 0x00, 0x1C, 0x00, 0x03, 0x80, 0x01, 0xE0, 0x00,
0x00, 0x1C, 0x00, 0x03, 0x80, 0x01, 0xE0, 0x00, 0x00, 0x1C, 0x00, 0x01,
0x00, 0x01, 0xC0, 0x00, 0x00, 0x1C, 0x00, 0x00, 0x00, 0xFF, 0xC0, 0x00,
0x00, 0x1C, 0x00, 0x00, 0x00, 0x3F, 0xC0, 0x00, 0x00, 0x1C, 0x00, 0x00,
0x00, 0x1F, 0xC0, 0x00, 0x00, 0x18, 0x00, 0x00, 0x00, 0x07, 0x80, 0x00,
0x00, 0x20, 0x00, 0x00, 0x00, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
};/* "POSTEK char", 64 * 64 dot matrix */

```

```

PTK_OpenUSBPort(255);
PTK_DrawBinGraphics(50, 50, 8, 64, font_bo);
PTK_PrintLabel(1, 1);
PTK_CloseUSBPort();

```

---

**Prints a line**

### *PTK\_DrawRectangle*

Description: draws an empty rectangle on the label.

Note: PTK\_PrintLabel must be called last to start printing.

Prototype:

```
int _stdcall PTK_DrawRectangle(unsigned int px, unsigned int py,  
    unsigned int thickness, unsigned int pEx,unsigned int pEy);
```

Parameters:

- px: X coordinate of the start point, in dots;
- py: Y coordinate of the start point, in dots;
- thickness: border thickness, in dots;
- pEx: X coordinate of the end point, in dots;
- pEy: Y coordinate of the end point, in dots.

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
PTK_OpenUSBPort(255);  
PTK_DrawRectangle(50, 50, 24, 200, 200);  
PTK_PrintLabel(1, 1); // Print one label  
PTK_CloseUSBPort();
```

### *PTK\_DrawLineXor*

Description: draws a straight line, applying an XOR operation where it intersects existing content.

Note: Can also be used to draw a filled rectangle.

Note: PTK\_PrintLabel must be called last to start printing.

Prototype:

```
int _stdcall PTK_DrawLineXor(unsigned int px, unsigned int py,unsigned int pL, unsigned int pH);
```

Parameters:

- px: starting X coordinate, in dots;
- py: starting Y coordinate, in dots;
- pL: horizontal length of the line, in dots;
- pH: vertical height of the line, in dots.

---

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

**Example:**

```
PTK_OpenUSBPort(255);
PTK_DrawLineXor(300, 300, 500, 12);
PTK_DrawLineXor(400, 200, 12, 500);
PTK_PrintLabel(1, 1); // Print one label
PTK_CloseUSBPort();
```

### *PTK\_DrawLineOr*

Description: draws a straight line, applying an OR operation where it intersects existing content.

Note: Can also be used to draw a filled rectangle.

Note: PTK\_PrintLabel must be called last to start printing.

**Prototype:**

```
int _stdcall PTK_DrawLineOr(unsigned int px, unsigned int py, unsigned int pL, unsigned int pH);
```

Parameters:

px: starting X coordinate, in dots;  
py: starting Y coordinate, in dots;  
pL: horizontal length of the line, in dots;  
pH: vertical height of the line, in dots.

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

**Example:**

```
PTK_OpenUSBPort(255);
PTK_DrawLineOr (300, 300, 500, 12);
PTK_DrawLineOr(400, 200, 12, 500);
PTK_PrintLabel(1, 1); // Print one label
PTK_CloseUSBPort();
```

---

### *PTK\_DrawDiagonal*

Description: draws a diagonal line, applying an OR operation where it intersects existing content.

Note: PTK\_PrintLabel must be called last to start printing.

#### Prototype:

```
int _stdcall PTK_DrawDiagonal(unsigned int px, unsigned int py, unsigned int thickness, unsigned int pEx, unsigned int pEy);
```

#### Parameters:

px: X coordinate of the start point, in dots;  
py: Y coordinate of the start point, in dots;  
thickness: border thickness, in dots;  
pEx: X coordinate of the end point, in dots;  
pEy: Y coordinate of the end point, in dots.

#### Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

#### Example:

```
PTK_OpenUSBPort(255);  
PTK_DrawDiagonal(50, 50, 12, 300, 300);  
PTK_DrawDiagonal(300, 50, 12, 50, 300);  
PTK_PrintLabel(1, 1); // Print one label  
PTK_CloseUSBPort();
```

### *PTK\_DrawWhiteLine*

Description: draws a white straight line.

Note: PTK\_PrintLabel must be called last to start printing.

#### Prototype:

```
int _stdcall PTK_DrawWhiteLine(unsigned int px, unsigned int py, unsigned int pL, unsigned int pH);
```

#### Parameters:

px: X coordinate of the start point, in dots;  
py: Y coordinate of the start point, in dots;  
pL: horizontal length of the line, in dots;  
pH: vertical height of the line, in dots.

---

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
PTK_OpenUSBPort(255);
PTK_DrawLineOr(50, 50, 500, 500);
PTK_DrawWhiteLine(100, 100, 300, 12);
PTK_DrawWhiteLine(100, 200, 300, 12);
PTK_DrawWhiteLine(100, 300, 300, 12);
PTK_PrintLabel(1, 1); // Print one label
PTK_CloseUSBPort();
```

**Prints a 2D barcode**

*PTK\_DrawBar2D\_QR*

Description: prints a QR code.

Note: PTK\_PrintLabel must be called last to start printing.

Prototype:

```
int _stdcall PTK_DrawBar2D_QR(unsigned int x, unsigned int y,
    unsigned int w, unsigned int v,
    unsigned int o, unsigned int r,
    unsigned int m, unsigned int g,
    unsigned int s, LPTSTR pstr);
```

Parameters:

x: X coordinate of the start point, in dots;

y: Y coordinate of the start point, in dots;

w: reserved parameter, please pass 0.

v: QR code version, which corresponds to the QR code's size. Version number ranges from 1 to 40.

Version 1 is a 21x21 matrix; each version increment increases the matrix size by 4 modules, so version 40 is a 177x177 matrix.

0: auto-detect (default).

1: 21\* 21

2: 25\* 25

.....

40: 177\* 177

o: rotation direction. Range: 0~3.



---

(0--0° , 1--90° , 2--180° , 3--270° )

r: scaling factor, in dots. Range: 1 - 99.  
(1 -- 1x; 2 -- 2x; 3 -- 3x; ...)

m: reserved parameter, please pass 0.

g: QR code error correction level. Range: 0 - 3.  
0 is 'L' level  
1 is 'M' level  
2 is 'Q1' level  
3 is 'H1' level

s: QR code mask pattern. Range: 0 - 8.  
0 - is mask pattern 000  
1 - is mask pattern 001  
2 - is mask pattern 010  
3 - is mask pattern 011  
4 - is mask pattern 100  
5 - is mask pattern 101  
6 - is mask pattern 110  
7 - is mask pattern 111  
8 - automatically selects the mask pattern

pstr: barcode content string.

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

#### Example:

```
PTK_OpenUSBPort(255);  
PTK_DrawBar2D_QR(300, 100, 0, 0, 0, 10, 0, 0, 8, _T("POSTEK Technology Development Co., Ltd.  
POSTEK2020"));  
PTK_PrintLabel(1, 1); // Print one label  
PTK_CloseUSBPort();
```

#### *PTK\_DrawBar2D\_QREx*

Description: prints a QR code as an image and saves it as an image on the printer to be printed. Supports legacy firmware.

Note: PTK\_PrintLabel must be called last to start printing.

#### Prototype:

```
int _stdcall PTK_DrawBar2D_QREx(unsigned int x, unsigned int y,  
    unsigned int o, unsigned int r,  
    unsigned int g, unsigned int v,  
    unsigned int s, LPTSTR binname,  
    LPTSTR pstr);
```

---

Parameters:

x: X coordinate of the start point, in dots;

y: Y coordinate of the start point, in dots;

o: rotation direction. Range: 0~3.  
(0--0°, 1--90°, 2--180°, 3--270° )

r: scaling factor, in dots. Range: 1 - 99.  
(1 -- 1x; 2 -- 2x; 3 -- 3x; ...)

g: QR code error correction level. Range: 0 - 3.  
0 is 'L' level  
1 is 'M' level  
2 is 'Q1' level  
3 is 'H1' level

v: QR code version, which corresponds to the QR code's size. Version number ranges from 1 to 40.

Version 1 is a 21x21 matrix; each version increment increases the matrix size by 4 modules, so version 40 is a 177x177 matrix.

0: auto-detect (default).

1: 21\* 21

2: 25\* 25

.....

40: 177\* 177

s: QR code mask pattern. Range: 0 - 8.

0 - is mask pattern 000

1 - is mask pattern 001

2 - is mask pattern 010

3 - is mask pattern 011

4 - is mask pattern 100

5 - is mask pattern 101

6 - is mask pattern 110

7 - is mask pattern 111

8 - automatically selects the mask pattern

binname: the name under which the image is stored in the printer. Maximum length: 16 characters.

Note: You can use this name with PTK\_RecallBinGraphics to reprint the 2D code; lost when power is removed. If FLASH storage is enabled, it remains valid after power-cycle.

pstr: barcode content string.

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
PTK_OpenUSBPort(255);
```

```
PTK_DrawBar2D_QREx(100, 100, 0, 10, 0, 0, 8, "QR1", _T("POSTEK Technology Development Co., Ltd.  
POSTEK2020"));
```

---

```
PTK_RecallBinGraphics(400, 100, _T("QR1"));
PTK_RecallBinGraphics(100, 400, _T("QR1"));
PTK_RecallBinGraphics(400, 400, _T("QR1"));
PTK_PrintLabel(1, 1); // Print one label
PTK_CloseUSBPort();
```

### *PTK\_DrawBar2D\_HANXIN*

Description: prints a Han Xin code.

Note: PTK\_PrintLabel must be called last to start printing.

Prototype:

```
int _stdcall PTK_DrawBar2D_HANXIN(unsigned int x, unsigned int y,
    unsigned int w, unsigned int v,
    unsigned int o, unsigned int r,
    unsigned int m, unsigned int g,
    unsigned int s, LPTSTR pstr);
```

Parameters:

- x: X coordinate of the start point, in dots;
- y: Y coordinate of the start point, in dots;
- w: reserved parameter, please pass 0;
- v: reserved parameter, please pass 0;
- o: rotation direction. Range: 0~3.  
(0—0°, 1—90°, 2—180°, 3—270° )
- r: scaling factor, in dots. Range: 0 - 30.  
(0 — 1x; 1 — 2x; 2 — 3x; ...)
- m: Han Xin code encoding mode. Range: 0 to 6.
  - 0 selects Numeric mode,
  - 1 selects TEXT mode,
  - 2 selects Binary mode,
  - 3 selects Common Chinese Region 1 mode encoding
  - 4 selects Common Chinese Region 2 mode encoding
  - 5 is GB 18030 double-byte region mode
  - 6 is GB 18030 four-byte mode encoding
- g: Han Xin code error correction level. Range: 0 to 3.
  - 0 is 'L1' level
  - 1 is 'L2' level
  - 2 is 'L3' level
  - 3 is 'L4' level
- s: Han Xin code mask pattern. Range: 0 to 3.
  - 0 is mask pattern 00
  - 1 is mask pattern 01
  - 2 is mask pattern 10

---

3 is mask pattern 11  
pstr: barcode content string.

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

**Example:**

```
PTK_OpenUSBPort(255);  
PTK_DrawBar2D_HANXIN(50, 50, 0, 0, 0, 8, 5, 3, 2, _T("http://www.postek.com.cn/"));  
PTK_PrintLabel(1, 1); // Print one label  
PTK_CloseUSBPort();
```

[\*PTK\\_DrawBar2D\\_Pdf417\*](#)

Description: prints a PDF417 code.

Note: PTK\_PrintLabel must be called last to start printing.

**Prototype:**

```
int _stdcall PTK_DrawBar2D_Pdf417(unsigned int x, unsigned int y,  
    unsigned int w, unsigned int v,  
    unsigned int s, unsigned int c,  
    unsigned int px, unsigned int py,  
    unsigned int r, unsigned int l,  
    unsigned int t, unsigned int o,  
    LPTSTR pstr);
```

Parameters:

x: X coordinate of the start point, in dots;  
y: Y coordinate of the start point, in dots;  
w: reserved parameter, please pass 0.  
v: reserved parameter, please pass 0.  
s: error correction level. Range: 0~8.  
c: reserved parameter, please pass 0.  
px: module width. Range: 2~9 dots.  
py: module height. Range: 4~99 dots.  
r: maximum number of rows. Range: 3~90.  
l: maximum number of columns. Range: 1~30.  
t: truncation flag,  
    0 - do not truncate, 1 - truncate.  
o: rotation direction. Range: 0~3.  
    (0--0°, 1--90°, 2--180°, 3--270° )  
pstr: barcode content string.

Return Value:

0 -> OK

---

For other return values, call PTK\_GetErrorInfo to parse them.

**Example:**

```
PTK_OpenUSBPort(255);
PTK_DrawBar2D_Pdf417(50, 50, 0, 0, 0, 0, 3, 7, 10, 2, 0, 0, _T("POSTEKINFO"));
PTK_PrintLabel(1, 1); // Print one label
PTK_CloseUSBPort();
```

*PTK\_DrawBar2D\_Pdf417Ex*

Description: prints a PDF417 barcode as an image.

Note: This function is supported on firmware that cannot call PTK\_DrawBar2D\_Pdf417 for printing.

Note: PTK\_PrintLabel must be called last to start printing.

**Prototype:**

```
int _stdcall PTK_DrawBar2D_Pdf417Ex(unsigned int x, unsigned int y,
    unsigned int w, unsigned int v,
    unsigned int s, unsigned int c,
    unsigned int px, unsigned int py,
    unsigned int r, unsigned int l,
    unsigned int t, unsigned int o,
    LPTSTR pstr);
```

**Parameters:**

x: X coordinate of the start point, in dots;  
y: Y coordinate of the start point, in dots;  
w: reserved parameter, please pass 0.  
v: reserved parameter, please pass 0.  
s: error correction level. Range: 0~8.  
c: reserved parameter, please pass 0.  
px: module width. Range: 2~9 dots.  
py: module height. Range: 4~99 dots.  
r: maximum number of rows. Range: 3~90.  
l: maximum number of columns. Range: 1~30.  
t: truncation flag,  
    0 - do not truncate, 1 - truncate.  
o: rotation direction. Range: 0~3.  
    (0--0°, 1--90°, 2--180°, 3--270° )  
pstr: barcode content string.

**Return Value:**

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

**Example:**

```
PTK_OpenUSBPort(255);
```

---

```
PTK_DrawBar2D_Pdf417Ex(50, 50, 0, 0, 0, 0, 3, 7, 10, 2, 0, 0, _T("POSTEKINFO"));
PTK_PrintLabel(1, 1); // Print one label
PTK_CloseUSBPort();
```

### *PTK\_DrawBar2D\_MaxiCode*

Description: prints a MaxiCode.

Note: PTK\_PrintLabel must be called last to start printing.

Prototype:

```
int _stdcall PTK_DrawBar2D_MaxiCode(unsigned int x, unsigned int y,
    unsigned int m, unsigned int u, LPTSTR pstr);
```

Parameters:

x: X coordinate of the start point, in dots;  
y: Y coordinate of the start point, in dots;  
m: Mode [2 - 4];  
u: whether the format is UPS. Value: 0 or 1.  
pstr: barcode content string.

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
PTK_OpenUSBPort(255);
PTK_DrawBar2D_MaxiCode(50, 50, 4, 0, _T("1Z000A7&dajc_iaj-3=+~#$5"));
PTK_PrintLabel(1, 1); // Print one label
PTK_CloseUSBPort();
```

### *PTK\_DrawBar2D\_DATAMATRIX*

Description: prints a Data Matrix code.

Note: PTK\_PrintLabel must be called last to start printing.

Prototype:

```
int _stdcall PTK_DrawBar2D_DATAMATRIX(unsigned int x, unsigned int y,
    unsigned int w, unsigned int v,
    unsigned int o, unsigned int m, LPTSTR pstr);
```

Parameters:

x: X coordinate of the start point, in dots;  
y: Y coordinate of the start point, in dots;  
w: reserved parameter, please pass 0.  
v: reserved parameter, please pass 0.  
o: rotation direction. Range: 0~3.

---

(0--0° , 1--90° , 2--180° , 3--270° )

m: scaling factor, in dots.

pstr: barcode content string.

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
PTK_OpenUSBPort(255);  
PTK_DrawBar2D_DATAMATRIX(50, 50, 0, 0, 0, 20, _T("POSTEK2020"));  
PTK_PrintLabel(1, 1); // Print one label  
PTK_CloseUSBPort();
```

## Prints a 1D barcode

### *PTK\_DrawBarcode*

Description: prints various types of 1D barcodes, where the printed content is a constant string.

Note: PTK\_PrintLabel must be called last to start printing.

Prototype:

```
int _stdcall PTK_DrawBarcode(unsigned int px, unsigned int py,  
    unsigned int pdirec, LPTSTR pCode, unsigned int NarrowWidth,  
    unsigned int pHorizontal, unsigned int pVertical, TCHAR ptext,  
    LPTSTR pstr);
```

Parameters:

px: set the X coordinate, in dots.

py: set the Y coordinate, in dots.

pdirec: select the print direction of the barcode. 0 - no rotation; 1 - rotate 90 deg; 2 - rotate 180 deg; 3 - rotate 270 deg.

pCode: select the type of barcode to print. (Different barcode types have character or length restrictions; refer to the specific standard.)

P4 value	Barcode type
0	Code 128 UCC (shipping container code)
1	Code 128 AUTO
1A	Code 128 subset A
1B	Code 128 subset B
1C	Code 128 subset C
1E	UCC/EAN

---

2	Interleaved 2 of 5
2C	Interleaved 2 of 5 with check sum digit
2D	Interleaved 2 of 5 with human readable check digit
2G	German Postcode
2M	Matrix 2 of 5
2U	UPC Interleaved 2 of 5
3	Code 3 of 9
3C	Code 3 of 9 with check sum digit
3E	Extended Code 3 of 9
3F	Extended Code 3 of 9 with check sum digit
9	Code93
E30	EAN-13
E32	EAN-13 2 digit add-on
E35	EAN-13 5 digit add-on
E80	EAN-8
E82	EAN-8 2 digit add-on
E-85	EAN-8 5 digit add-on
K	Codabar
P	Postnet
UA0	UPC-A
UA2	UPC-A 2 digit add-on
UA5	UPC-A 5 digit add-on
UE0	UPC-E
UE2	UPC-E 2 digit add-on
UE5	UPC-E 5 digit add-on

NarrowWidth: set the width of the narrow element in the barcode, in dots.

pHorizontal: set the width of the wide element in the barcode, in dots.

pVertical: set the barcode height, in dots.

pText: 'N' (ASCII value 78) means do not print human-readable text below the barcode.

'B': print the human-readable text below the barcode, left aligned.

When the barcode type is set to Code 128 AUTO:

'C': print the human-readable text below the barcode, center aligned.

'R': print the human-readable text below the barcode, right aligned.

**Note:** For 200 dpi printers, you must set `tph=2` in `CDFPSK.ini` to use the center/right alignment feature.

pstr: a string of 1-100 characters.

Return Value:

0 -> OK

For other return values, call `PTK_GetErrorInfo` to parse them.

**Example:**

```
PTK_OpenUSBPort(255);
PTK_DrawBarcode(100, 50, 0, _T("1"), 3, 0, 100, 'N', _T("POSTEK2020"));
PTK_DrawBarcode(100, 200, 0, _T("1"), 3, 0, 100, 'B', _T("POSTEK2020"));
```



---

```

PTK_DrawBarcode(100, 350, 0, _T("1"), 3, 0, 100, 'C', _T("POSTEK2020")); // Currently only
Code128 auto supports center/right alignment
PTK_DrawBarcode(100, 500, 0, _T("1"), 3, 0, 100, 'R', _T("POSTEK2020"));
PTK_PrintLabel(1, 1); // Print one label
PTK_CloseUSBPort();

```

### *PTK\_DrawBarcodeEx*

**Description:** prints various types of 1D barcodes, where the printed content can be a constant string, a serial number or a variable.

**Note:** PTK\_PrintLabel must be called last to start printing.

**Prototype:**

```

int _stdcall PTK_DrawBarcodeEx(unsigned int px, unsigned int py,
    unsigned int pdirec, LPTSTR pCode,
    unsigned int NarrowWidth,
    unsigned int pHorizontal,
    unsigned int pVertical,
    TCHAR ptext, LPTSTR pstr, BOOL Variable);

```

**Parameters:**

- px: set the X coordinate, in dots.
- py: set the Y coordinate, in dots.
- pdirec: select the print direction of the barcode. 0 – no rotation; 1 – rotate 90 deg; 2 – rotate 180 deg; 3 – rotate 270 deg.
- pCode: select the type of barcode to print. (Different barcode types have character or length restrictions; refer to the specific standard.)

P4 value	Barcode type
0	Code 128 UCC (shipping container code)
1	Code 128 AUTO
1A	Code 128 subset A
1B	Code 128 subset B
1C	Code 128 subset C
1E	UCC/EAN
2	Interleaved 2 of 5
2C	Interleaved 2 of 5 with check sum digit
2D	Interleaved 2 of 5 with human readable check digit
2G	German Postcode
2M	Matrix 2 of 5
2U	UPC Interleaved 2 of 5
3	Code 3 of 9
3C	Code 3 of 9 with check sum digit
3E	Extended Code 3 of 9
3F	Extended Code 3 of 9 with check sum digit
9	Code93
E30	EAN-13
E32	EAN-13 2 digit add-on

E35	EAN-13 5 digit add-on
E80	EAN-8
E82	EAN-8 2 digit add-on
E-85	EAN-8 5 digit add-on
K	Codabar
P	Postnet
UA0	UPC-A
UA2	UPC-A 2 digit add-on
UA5	UPC-A 5 digit add-on
UE0	UPC-E
UE2	UPC-E 2 digit add-on
UE5	UPC-E 5 digit add-on

NarrowWidth: set the width of the narrow element in the barcode, in dots.

pHorizontal: set the width of the wide element in the barcode, in dots.

pVertical: set the barcode height, in dots.

ptext: 'N' (ASCII value 78) means do not print human-readable text below the barcode.

'B': print the human-readable text below the barcode, left aligned.

When the barcode type is set to Code 128 AUTO:

'C': print the human-readable text below the barcode, center aligned.

'R': print the human-readable text below the barcode, right aligned.

**Note: For 200 dpi printers, you must set tph=2 in CDFPSK.ini to use the center/right alignment feature.**

pstr: a string of 1-100 characters. Users may freely combine "DATA", Cn and Vn into a composite string,

"DATA": a constant string. The double quote (") must be used as the start and end markers, e.g. "POSTEK Printer".

Cn: serial number value. This serial number must have been previously defined.

Vn: variable string. This variable string must have been previously defined.

Example: "text1"Cn"text2"Vn.

R: indicates that the content to print is the current UHF RFID tag TID (64-bit) data. (Note: this feature is only supported on RFID printers.)

Variable: true indicates that the current string contains a variable operation; \" must be added to enclose the constant strings to be printed.

For example: PTKDrawBarcodeEx (50,30,0,"1A",1,1,10,'N',"123456\",true);

false indicates that the current string does not contain a variable operation and that \" need not be added.

For example, PTK\_DrawBarcode(100, 100, 0, \_T("1"), 3, 0, 100, 'N', \_T("POSTEK2020"));  
and PTK\_DrawBarcodeEx(100, 100, 0, \_T("1"), 3, 0, 100, 'N', \_T("POSTEK2020"),  
FALSE); have the same effect;

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

---

```
PTK_OpenUSBPort(255);
PTK_DefineCounter(0, 6, 'N', _T("+1"), _T("\nEnter\Code:")); // Define serial number C0
PTK_DrawBarcodeEx(100, 100, 0, _T("1"), 3, 0, 100, 'C', _T("\Test Serial:\C0"), TRUE);
PTK_DrawBarcodeEx(100, 300, 0, _T("1"), 3, 0, 100, 'C', _T("Test Serial:Text"), FALSE);
PTK_Download();
PTK_DownloadInitVar(_T("2020")); // Initialize C0
PTK_PrintLabel(3, 1); // Print three labels
PTK_CloseUSBPort();
```

## Print form and related functions

### *PTK\_GetStorageList*

Description: retrieves the names of the forms, fonts or images stored in FLASH.

Note: Only items in FLASH can be retrieved; items in RAM cannot.

Note: Only USB reads are supported.

#### Prototype:

```
int _stdcall PTK_GetStorageList(LPTSTR listBuff, DWORD listBuffSize, int TempType);
```

#### Parameters:

listBuff: buffer that stores the name list.

listBuffSize: size of the listBuff buffer.

TempType: the type of name list to retrieve.

0: form.

1: font.

2: image.

#### Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

#### Example:

```
TCHAR buff[1024] = { 0 };
PTK_OpenUSBPort(255);
PTK_GetStorageList(buff, sizeof(buff), 0);
MessageBox(buff);
PTK_CloseUSBPort();
```

### *PTK\_FormList*

Description: prints a list of the names of the forms stored in the printer's RAM or FLASH memory.

Note: This function does not require PTK\_PrintLabel to be called in order to print.

---

**Prototype:**

```
int _stdcall PTK_FormList(void);
```

**Parameters:**

None

**Return Value:**

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

**Example:**

```
PTK_OpenUSBPort(255);  
PTK_FormList();  
PTK_CloseUSBPort();
```

### *PTK\_FormDel*

**Description:** deletes a form stored on the printer.

**Prototype:**

```
int _stdcall PTK_FormDel(LPTSTR pid);
```

**Parameters:**

pid: the name of the form stored on the printer. Maximum 16 bytes.

**Return Value:**

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

**Example:**

```
PTK_OpenUSBPort(255);  
PTK_FormDel(_T("F1"));  
PTK_CloseUSBPort();
```

### *PTK\_FormDownload*

**Description:** tells the printer to start storing a form.

**Note:** Use together with PTK\_FormEnd.

**Prototype:**

```
int _stdcall PTK_FormDownload(LPTSTR pid);
```

**Parameters:**

pid: name under which to store the form on the printer. Maximum 16 bytes.

---

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

**Example:**

```
PTK_OpenUSBPort(255);  
/* Store a form */  
//PTK_EnableFLASH();// Enable FLASH storage  
PTK_FormDel(_T("F1"));// Delete any form with the same name to avoid name collisions  
PTK_FormDownload(_T("F1"));// Instruct the printer to start storing the form content  
PTK_DrawTextEx(100, 100, 0, 6, 2, 2, 'N', _T("1234567"), FALSE);// Print one line of text  
PTK_FormEnd();// Tell the printer that form storage is complete  
//PTK_DisableFLASH();// Disable FLASH storage  
PTK_CloseUSBPort();
```

### *PTK\_FormEnd*

**Description:** tells the printer that form storage is complete.

**Note:** Use together with PTK\_FormDownload.

**Prototype:**

```
int _stdcall PTK_FormEnd(void);
```

Parameters:

None

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

**Example:**

```
PTK_OpenUSBPort(255);  
/* Store a form */  
//PTK_EnableFLASH();// Enable FLASH storage  
PTK_FormDel(_T("F1"));// Delete any form with the same name to avoid name collisions  
PTK_FormDownload(_T("F1"));// Instruct the printer to start storing the form content  
PTK_DrawTextEx(100, 100, 0, 6, 2, 2, 'N', _T("1234567"), FALSE);// Print one line of text  
PTK_FormEnd();// Tell the printer that form storage is complete  
//PTK_DisableFLASH();// Disable FLASH storage  
PTK_CloseUSBPort();
```

### *PTK\_ExecForm*

**Description:** runs a form. Equivalent to executing every API call in the form once.

**Note:**

- 
- 1, The form must already be stored on the printer; you can use `PTK_GetStorageList` or `PTK_FormList` to inspect the form names.
  - 2, If `PTK_PrintLabelAuto` is not called inside the form to perform the print, `PTK_PrintLabel` must also be called in order to start printing the labels.

**Prototype:**

```
int _stdcall PTK_ExecForm(LPTSTR pid);
```

**Parameters:**

pid: the name of the form stored on the printer. Maximum 16 bytes.

**Return Value:**

0 -> OK

For other return values, call `PTK_GetErrorInfo` to parse them.

**Example:**

```
PTK_OpenUSBPort(255);
/* Store a form */
//PTK_EnableFLASH();// Enable FLASH storage
PTK_FormDel(_T("F1"));// Delete any form with the same name to avoid name collisions
PTK_FormDownload(_T("F1"));// Instruct the printer to start storing the form content
PTK_DrawTextEx(100, 100, 0, 6, 2, 2, 'N', _T("1234567"), FALSE);// Print one line of text
PTK_FormEnd();// Tell the printer that form storage is complete
//PTK_DisableFLASH();// Disable FLASH storage
PTK_ExecForm(_T("F1"));
PTK_PrintLabel(1, 1);
PTK_CloseUSBPort();
```

*[PTK\\_DefineVariable](#)*

**Description:** defines a variable on the printer.

**Prototype:**

```
int _stdcall PTK_DefineVariable(unsigned int id, unsigned int maxNum, TCHAR ptext, LPTSTR hintMsg);
```

**Parameters:**

id: variable ID. Range: 00-99;

maxNum: maximum number of characters. Range: 1-99;

ptext: alignment. L - left aligned; R - right aligned; C - center; N - no alignment.

hintMsg: the prompt text, which will be displayed on the KDU or the printer's LCD.

**Return Value:**

0 -> OK

For other return values, call `PTK_GetErrorInfo` to parse them.

---

**Example:**

```
TCHAR v0[16] = _T("1234");
PTK_OpenUSBPort(255);
PTK_DefineVariable(0, 4, 'L', _T(""));
PTK_DrawTextEx(100, 100, 0, 6, 2, 2, 'N', _T("\Test variable\V0"), TRUE);
PTK_Download();
PTK_DownloadInitVar(v0); // Assign a value to a printer variable
PTK_PrintLabel(1, 1);
PTK_CloseUSBPort();
```

### *PTK\_DefineCounter*

Description: defines a serial number on the printer.

**Prototype:**

```
int _stdcall PTK_DefineCounter(unsigned int id, unsigned int maxNum, TCHAR ptext, LPTSTR prule, LPTSTR hintMsg);
```

**Parameters:**

id: serial-number ID. Range: 0-9;

maxNum: maximum number of digits in the serial number. Range: 1-40;

ptext: alignment. L - left aligned; R - right aligned; C - center; N - no alignment.

prule: rule by which the serial number changes. It consists of "+" or "-", followed by a number, followed by a change flag (D - decimal, B - binary, O - octal, H - hexadecimal):

- " +1" = increment by 1 each time, in decimal by default, e.g. 1234, 1235, 1236, ...;
- " +3D" = increment by 3 each time, in decimal (same as above);
- " -1B" = decrement by 1 each time, in binary, e.g. 1111, 1110, 1101, ...;
- " -4O" = decrement by 4 each time, in octal, e.g. 1234, 1230, 1224, ...;
- " -6H" = decrement by 6 each time, in hexadecimal, e.g. 1234, 122E, 1228, ...;

hintMsg: the prompt text, which will be displayed on the KDU or the printer's LCD.

**Return Value:**

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

**Example:**

```
PTK_OpenUSBPort(255);
PTK_DefineCounter(0, 6, 'L', _T("+3"), _T(""));
PTK_DrawTextEx(100, 100, 0, 6, 2, 2, 'N', _T("\Test serial number\CO"), TRUE);
PTK_Download();
PTK_DownloadInitVar(_T("0001")); // Assign the printer's initial serial number value
PTK_PrintLabel(3, 1);
PTK_CloseUSBPort();
```

---

### *PTK\_Download*

Description: tells the printer to start assigning initial values to variables or serial numbers.

#### Prototype:

```
int _stdcall PTK_Download(void);
```

#### Parameters:

None

#### Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

#### Example:

```
PTK_OpenUSBPort(255);
PTK_DefineCounter(0, 6, 'L', _T("+3"), _T(""));
PTK_DrawTextEx(100, 100, 0, 6, 2, 2, 'N', _T("\Test serial number\CO"), TRUE);
PTK_Download();
PTK_DownloadInitVar(_T("0001")); // Assign the printer's initial serial number value
PTK_PrintLabel(3, 1);
PTK_CloseUSBPort();
```

### *PTK\_DownloadInitVar*

Description: initializes variables or serial numbers in the order in which they were defined.

#### Prototype:

```
int _stdcall PTK_DownloadInitVar(LPTSTR pstr);
```

#### Parameters:

pstr: string containing the initial value to set.

#### Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

#### Example:

```
PTK_OpenUSBPort(255);
PTK_DefineCounter(0, 6, 'L', _T("+3"), _T(""));
PTK_DrawTextEx(100, 100, 0, 6, 2, 2, 'N', _T("\Test serial number\CO"), TRUE);
PTK_Download();
PTK_DownloadInitVar(_T("0001")); // Assign the printer's initial serial number value
PTK_PrintLabel(3, 1);
PTK_CloseUSBPort();
```



---

### *PTK\_PrintLabelAuto*

Description: instructs the printer to start printing labels; used inside a form, in combination with serial numbers or variables.

Note: This function must be used together with PTK\_Download. The printer will only start printing once the form has defined a serial number or variable.

#### Prototype:

```
int _stdcall PTK_PrintLabelAuto(unsigned int number, unsigned int cpnumber);
```

#### Parameters:

number: number of labels to print. Range: 1-65535;

cpnumber: number of copies of each label. Range: 1-65535;

#### Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

#### Example:

```
PTK_OpenUSBPort(255);
/* Store a form */
PTK_FormDel(_T("F1"));
PTK_FormDownload(_T("F1"));
PTK_DefineVariable(0, 6, 'N', _T(""));
PTK_DefineCounter(0, 6, 'N', _T("+1"), _T(""));
PTK_DrawTextEx(100, 100, 0, 6, 2, 2, 'N', _T("V0\"Test variable\""), TRUE);
PTK_DrawTextEx(100, 200, 0, 6, 2, 2, 'N', _T("C0\"Test serial number\""), TRUE);
PTK_PrintLabelAuto(3, 1); // Used inside a form
PTK_FormEnd();
/* Run a form */
PTK_ExecForm(_T("F1"));
PTK_Download();
PTK_DownloadInitVar(_T("1234"));
PTK_DownloadInitVar(_T("1234"));
PTK_CloseUSBPort();
```

### *PTK\_FormPrinting*

Description: prints a form.

#### Prototype:

```
int _stdcall PTK_FormPrinting(LPTSTR FormName, LPTSTR FormVariable, unsigned int Quantity, unsigned int Copies);
```

#### Parameters:

---

FormName: form name, no more than 16 bytes;  
FormVariable: form variables. Variables are separated by "\r\n". If a variable itself contains "\r\n", use the format "\\r\\n";  
number: number of labels to print. Range: 1-65535;  
cpnumber: number of copies of each label. Range: 1-65535;  
Return Value:  
0 -> OK  
For other return values, call PTK\_GetErrorInfo to parse them.

**Example:**

```
PTK_OpenUSBPort(255);  
PTK_FormPrinting(_T("Test"), _T("1\r\n2\r\n3\r\n4\r\n5\r\n6\r\n7\r\n"), 1, 1);  
PTK_CloseUSBPort();
```

## RFID tag read/write functions

### *PTK\_RFIDCalibrate*

Description: RFID and HF tag probing calibration.

**Prototype:**

```
int _stdcall PTK_RFIDCalibrate(void);
```

**Parameters:**

None

**Return Value:**

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

**Example:**

```
PTK_OpenUSBPort(255);  
PTK_RFIDCalibrate();  
PTK_CloseUSBPort();
```

### *PTK\_RWRFIDLabel*

Description: writes RFID tag data.

**Prototype:**

```
int _stdcall PTK_RWRFIDLabel(unsigned int nRWMode, unsigned int nWForm, unsigned int nStartBlock,  
unsigned int nWDataNum, unsigned int nWArea, LPTSTR pstr);
```

Note: PTK\_PrintLabel must be called last to start printing.

---

Parameters:

nRWMode: RFID operation mode. 0 - reserved (no function for now); 1 - write RFID;

nWForm: RFID write format. 0 - HEX (hexadecimal); 1 - ASCII;

nStartBlock: starting block for the write operation.

*Note: When writing to the EPC area, the units are words (2 bytes);*

nWDataNum: number of bytes to write.

nWArea: write area. 0 - Reserved area; 1 - EPC; 3 - USER;

pstr: a constant string. (Format is constrained by parameter P2.)

Note: When writing in HEX format, nWDataNum is the byte count of the data being written, not the byte count of the string.

Example: the HEX string "313233343536" corresponds to the bytes 0x31, 0x32, 0x33, 0x34, 0x35, 0x36 (data length of 6 bytes).

*The length of the data to be written must be expressed in words (2 bytes). The valid data length must be an integer multiple of 2 bytes.*

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
PTK_OpenUSBPort(255);  
//PTK_RWRFIDLabel(1, 0, 2, 6, 1, "313233343536");// Write in HEX format  
PTK_RWRFIDLabel(1, 1, 2, 6, 1, "123456");// Write in ASCII format, same effect as above  
PTK_PrintLabel(1, 1);  
PTK_CloseUSBPort();
```

*PTK\_RWRFIDLabelEx*

Description: writes RFID tag data (not cleared).

Prototype::

```
int __stdcall PTK_RWRFIDLabelEx(unsigned int nRWMode, unsigned int nWForm, unsigned int nStartBlock,  
unsigned int nWDataNum, unsigned int nWArea, LPTSTR pstr);
```

Note: PTK\_PrintLabel must be called last to start printing.

Parameters:

nRWMode: RFID operation mode. 0 - reserved (no function for now); 1 - write RFID;

nWForm: RFID write format. 0 - HEX (hexadecimal); 1 - ASCII;

nStartBlock: starting block for the write operation.

*Note: When writing to the EPC area, the units are words (2 bytes);*

nWDataNum: number of bytes to write.

nWArea: write area. 0 - Reserved area; 1 - EPC; 3 - USER;

pstr: a constant string. (Format is constrained by parameter P2.)

Note: When writing in HEX format, nWDataNum is the byte count of the data being

---

written, not the byte count of the string.

Example: the HEX string "313233343536" corresponds to the bytes 0x31, 0x32, 0x33, 0x34, 0x35, 0x36 (data length of 6 bytes).

The length of the data to be written must be expressed in words (2 bytes). The valid data length must be an integer multiple of 2 bytes.

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
PTK_OpenUSBPort(255);  
//PTK_RWRFIDLabel(1, 0, 2, 6, 1, "313233343536");// Write in HEX format  
PTK_RWRFIDLabelEx(1, 1, 2, 6, 1, "123456");// Write in ASCII format, same effect as above  
PTK_PrintLabel(1, 1);  
PTK_CloseUSBPort();
```

### *PTK\_SetRFLabelPWAndLockRFLabel*

Description: sets the RFID tag password and locks the RFID tag.

Prototype::

```
int _stdcall PTK_SetRFLabelPWAndLockRFLabel(unsigned int nOperationMode, unsigned int OperationnArea,  
LPTSTR pstr);
```

Parameters:

nOperationMode: operation mode. 0 - unlock; 1 - lock; 2 - permanent unlock; 3 - permanent lock; 4 - write password.

OperationnArea: operation area. 0 - Kill password area; 1 - Access password area; 2 - EPC; 3 - TID; 4 - USER.

pstr: a constant string. (Format limited to 8 HEX characters.)

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
PTK_OpenUSBPort(255);  
PTK_SetRFLabelPWAndLockRFLabel(1, 1, "73BE115B");  
PTK_CloseUSBPort();
```

### *PTK\_EncodeRFIDPC*

Description: encodes the RFID PC value or the Chinese national-standard encoding header.

---

Prototype::

```
int _stdcall PTK_EncoderRFIDPC(char* PCValue);
```

Parameters:

PCValue: the PC value of the RFID, or the encoding header for the Chinese national standard. The data is in hexadecimal format.

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
PTK_OpenUSBPort(255);
PTK_RWRFIDLabel(1, 1, 2, 6, 1, "123456");// Write in ASCII format,
PTK_EncoderRFIDPC("A040");
PTK_CloseUSBPort();
```

### *PTK\_SetRFID*

Description: sets the RFID tag print parameters.

Note: Effective only for a single print job. To use it, call it before every print. Not saved to FLASH.

Prototype::

```
int _stdcall PTK_SetRFID(unsigned int nReservationParameters, unsigned int nReadWriteLocation,
unsigned int ReadWriteArea, unsigned int nMaxErrNum, unsigned int nErrProcessingMethod);
```

Parameters:

nReservationParameters: reserved parameter. Pass 0 by default.

nReadWriteLocation: RFID read/write position. Range: 0-999. Default: 0. Units: mm.

ReadWriteArea: reserved parameter. Default value is 0.

nMaxErrNum: read/write error retry count. Range: 0~9. Default: 1.

nErrProcessingMethod: reserved parameter. Default value is 0.

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
PTK_OpenUSBPort(255);
PTK_SetRFID(0, 0, 0, 3, 0);
PTK_RWRFIDLabel(1, 1, 2, 6, 1, "123456");
PTK_PrintLabel(1, 1);
PTK_CloseUSBPort();
```

---

### *PTK\_ReadRFIDLabelData*

Description: reads RFID tag data.

Note: Supports reading via serial port, USB and network.

Prototype::

```
int _stdcall PTK_ReadRFIDLabelData(unsigned int nDataBlock, unsigned int nRFPower, unsigned int bFeed,
LPTSTR data, DWORD dataSize);
```

Parameters:

nDataBlock: select the data area. 0: TID, 1: EPC, 2: TID+EPC, 3: USER.

nRFPower: reserved. Please pass 0.

bFeed:       whether to feed one label forward after reading;  
              TRUE: feed one label forward is enabled,  
              FALSE: feed one label forward is disabled.

data: buffer that stores the RFID tag data.

dataSize: size of the buff buffer.

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
TCHAR data [1024] = { 0 };
PTK_OpenUSBPort(255);
PTK_ReadRFIDLabelData(1, 23, 1, data, sizeof(data));
MessageBox(data);
PTK_CloseUSBPort();
```

Example returned data:

TID :

E200341401260100016630C7

EPC :

12345692

TID+EPC:

TID:E200341401260100016630C7, EPC:12345692

USER:

45000000001200000058000000800024

Error code (ERROR + error area + error code):

ERROR+TID+EPC0003

P.S. Error codes: 0003: print Void 0 indicates TID could not be read

0004:print Void 1 indicates a write failure was read

0005:print Void 2 indicates a duplicate TID was read

0006:print Void 3 indicates that multiple new labels were re-inventoried

---

### *PTK\_ReadRFIDLabelDataEx*

Description: reads RFID tag data.

Note: Supports reading via serial port, USB and network.

Prototype::

```
int _stdcall PTK_ReadRFIDLabelData(unsigned int nDataBlock, unsigned int nRFPower, unsigned int bFeed,
LPTSTR AccessCode, LPTSTR data, DWORD dataSize);
```

Parameters:

nDataBlock: select the data area to read.

0 – TID

1 – EPC

2 – TID+EPC

3 – USER

4 – TID+USER

5 – RESERVED

6 – TID+RESERVED

nRFPower: reserved. Please pass 0.

bFeed: whether to feed one label forward after reading;

TRUE: feed one label forward is enabled,

FALSE: feed one label forward is disabled.

AccessCode: access password. (Format limited to 4 HEX bytes.) The default access password is 00000000.

data: buffer that stores the RFID tag data.

dataSize: size of the buff buffer.

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
TCHAR data [1024] = { 0 };
PTK_OpenUSBPort(255);
PTK_ReadRFIDLabelData(1, 23, 1, data, sizeof(data));
MessageBox(data);
PTK_CloseUSBPort();
```

Example returned data:

TID :

E200341401260100016630C7

EPC :

12345692

TID+EPC:

TID:E200341401260100016630C7, EPC:12345692

USER:

---

45000000001200000058000000800024

Error code (ERROR + error area + error code):

ERROR+TID+EPC0003

P.S. Error codes: 0003: print Void 0 indicates TID could not be read

0004:print Void 1 indicates a write failure was read

0005:print Void 2 indicates a duplicate TID was read

0006:print Void 3 indicates that multiple new labels were re-inventoried

### *PTK\_RFIDEndPrintLabel*

Description: prints one label, then returns the RFID tag data that was printed.

Note: This function includes the behavior of PTK\_PrintLabel; you do not need to call PTK\_PrintLabel separately to start printing.

Prototype::

```
int _stdcall PTK_RFIDEndPrintLabel(unsigned int block, LPTSTR data, DWORD dataSize);
```

Parameters:

block: select the data area. 0: TID, 1: EPC, 2: TID+EPC, 3: USER.

data: buffer used to store the retrieved RFID tag data.

dataSize: size of the data buffer.

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
TCHAR data[1024] = { 0 };
TCHAR status[1024] = { 0 };
PTK_OpenUSBPort(255);
PTK_DrawText(20, 20, 0, 6, 2, 2, 'N', _T("313233343536")); // Print text
PTK_RWRFIDLabel(1, 0, 2, 6, 1, _T("313233343536")); // Write RFID EPC data
PTK_RFIDEndPrintLabel(1, data, sizeof(data)); // Print one label
MessageBox(data);
PTK_CloseUSBPort();
```

Example returned data:

TID :

E200341401260100016630C7

EPC :

12345692

TID+EPC:

TID:E200341401260100016630C7, EPC:12345692

USER:



---

45000000001200000058000000800024

Error code (ERROR + error area + error code):

ERROR+TID+EPC0003

P.S. Error codes: 0003: print Void 0 indicates TID could not be read

0004:print Void 1 indicates a write failure was read

0005:print Void 2 indicates a duplicate TID was read

0006:print Void 3 indicates that multiple new labels were re-inventoried

### *PTK\_RFIDEndPrintLabelFeedBack*

Description: prints one label, then returns the RFID tag data that was printed along with the printer status.

Note: This function includes the behavior of PTK\_PrintLabel; you do not need to call PTK\_PrintLabel separately to start printing.

Note: This function is only supported on firmware V7.61 and later.

Prototype::

```
int _stdcall PTK_RFIDEndPrintLabelFeedBack(unsigned int block, LPTSTR data, DWORD dataSize, LPTSTR printerStatus, DWORD statusSize);
```

Parameters:

block: select the data area. 0: TID, 1: EPC, 2: TID+EPC, 3: USER.

data: buffer used to store the retrieved RFID tag data.

dataSize: size of the data buffer.

printerStatus: the printer's current status. The return value is formatted as W1XXXX, where XXXX is the printer status code.

status: size of the printerStatus buffer.

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
TCHAR data[1024] = { 0 };
TCHAR status[1024] = { 0 };
PTK_OpenUSBPort(255);
PTK_DrawText(20, 20, 0, 6, 2, 2, 'N', _T("313233343536")); // Print text
PTK_RWRFIDLabel(1, 0, 2, 6, 1, _T("313233343536")); // Write RFID EPC data
PTK_RFIDEndPrintLabelFeedBack(1, data, sizeof(data), status, sizeof(status));
MessageBox(data);
MessageBox(status);
PTK_CloseUSBPort();
```

Example returned data:

---

TID :  
E200341401260100016630C7  
EPC :  
12345692  
TID+EPC:  
TID:E200341401260100016630C7, EPC:12345692  
USER:  
45000000001200000058000000800024  
Error code (ERROR + error area + error code):  
ERROR+TID+EPC0003  
P.S. Error codes: 0003: print Void 0 indicates TID could not be read  
0004:print Void 1 indicates a write failure was read  
0005:print Void 2 indicates a duplicate TID was read  
0006:print Void 3 indicates that multiple new labels were re-inventoried

### *PTK\_ReadRFIDSetting*

Description: returns data settings after printing a UHF RFID label.

Note: This function is only supported on firmware V10.0 and later.

Prototype::

```
int _stdcall PTK_ReadRFIDSetting(unsigned int nRMode, unsigned int nStartBlock, unsigned int nRBlockLength, LPTSTR AccessCode);
```

Parameters:

nRMode: select the data area to read.

- 0 – TID
- 1 – EPC
- 2 – TID+EPC
- 3 – USER
- 4 – TID+USER
- 5 – RESERVED
- 6 – TID+RESERVED

Note: when reading the TID (nRMode=0), pass 0 for both nStartBlock and nRBlockLength.

nStartBlock: starting address to read from, in words.

nRBlockLength: length to read, in bytes.

AccessCode: access password. (Format limited to 4 HEX bytes.) The default access password is 00000000.

Description of the returned data format:

(1) Format of the success report:

ZONE1,x1,y1+DATA1\*ZONE2,x2,y2+DATA2\*.....ZONE<sub>n</sub>,x<sub>n</sub>,y<sub>n</sub>+DATA<sub>n</sub>

(2) Format of the failure report:

ZONE1,x1,y1+DATA1\*ZONE2,x2,y2+DATA2\*.....ERROR+ZONE<sub>m</sub>,x<sub>m</sub>,y<sub>m</sub>+ERRORTYPE

ZONE<sub>n</sub> represents the data area of the n-th read command. [TID; EPC; TID+TIDDATA+EPC (TIDDATA is the

---

tag's TID data); USER; TID+TIDDATA+USER; RESERVED; TID+TIDDATA+RESERVED]

xn represents the starting-block parameter of the n-th read command. When nRMode is 0 (TID) this field is absent;

yn represents the read-block-count parameter of the n-th read command. When nRMode is 0 (TID) this field is absent;

DATAn represents the data (hexadecimal) read by the n-th read command;

**ERRORTYPE** represents the error code. It can take the following categories:

0003 ---> RFID tag could not be scanned;

0004 ---> RFID tag write failed;

0005 ---> read the TID of an already-written tag but could not read the TID of a new tag;

0006 ---> read TIDs from at least two new tags;

0009 ---> TID type mismatch;

0010 ---> RFID feedback error;

0203 ---> RFID tag could not be scanned; trying to print again;

0204 ---> RFID tag write failed; trying to print again;

0205 ---> read the TID of an already-written tag but could not read the TID of a new tag; trying to print again;

0206 ---> read TIDs from at least two new tags; trying to print again;

0209 ---> TID type mismatch; trying to print again;

0210 ---> RFID feedback error; trying to print again;

Report retrieval failed. The LCD shows: RFID FEEDBACK ERROR.

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
TCHAR data[1024] = { 0 };
```

```
TCHAR status[1024] = { 0 };
```

```
PTK_OpenUSBPort(255);
```

```
PTK_DrawText(20, 20, 0, 6, 2, 2, 'N', _T("313233343536")); // Print text
```

```
PTK_RWRFDLabel(1, 0, 2, 6, 1, _T("313233343536")); // Write RFID EPC data
```

```
PTK_ReadRFIDSetting(2, 0, 5, _T("00000000"));
```

```
PTK_PrintAndCallback(data, status);
```

```
MessageBox(data);
```

```
MessageBox(status);
```

```
PTK_CloseUSBPort();
```

### *PTK\_PrintAndCallback*

Description: prints a UHF or HF RFID label and reports back the data.

Note: This function includes the behavior of PTK\_PrintLabel; you do not need to call PTK\_PrintLabel separately to start printing.

Note: This function is only supported on firmware V10.0 and later.

---

#### Prototype:

```
int _stdcall PTK_PrintAndCallback(LPTSTR data, LPTSTR printerStatus);
```

#### Parameters:

**data:** buffer used to store the retrieved RFID tag data.

**printerStatus:** the printer's current status. The return value is formatted as W1XXXX, where XXXX is the printer status code.

Note: For descriptions of the reported data, see PTK\_ReadRFIDSetting for UHF RFID and PTK\_ReadHFRFIDSetting for HF RFID.

#### Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

#### Example:

```
TCHAR data[1024] = { 0 };
TCHAR status[1024] = { 0 };
PTK_OpenUSBPort(255);
PTK_DrawText(20, 20, 0, 6, 2, 2, 'N', _T("313233343536")); // Print text
PTK_RWRFIDLabel(1, 0, 2, 6, 1, _T("313233343536")); // Write RFID EPC data
PTK_ReadRFIDSetting(2, 0, 5);
PTK_PrintAndCallback(data, status);
MessageBox(data);
MessageBox(status);
PTK_CloseUSBPort();
```

#### *PTK\_SetReadRFIDForwardSpeed*

Description: sets the speed at which the label advances to the optimal read/write position when reading RFID data.

Note: Only supported on firmware V7.60 and later.

#### Prototype::

```
int _stdcall PTK_SetReadRFIDForwardSpeed(unsigned int speed);
```

#### Parameters:

**speed:** speed at which the label advances to the optimal read/write position. Units: ips.

#### Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

#### Example:

```
PTK_OpenUSBPort(255);
```

---

```
PTK_SetReadRFIDForwardSpeed(4);  
PTK_CloseUSBPort();
```

### *PTK\_SetReadRFIDBackSpeed*

Description: sets the speed at which the label is rolled back to the print line when reading RFID data.

Note: Only supported on firmware V7.60 and later.

Prototype::

```
int _stdcall PTK_SetReadRFIDBackSpeed(unsigned int speed);
```

Parameters:

speed: speed at which the label is rolled back to the print line. Units: ips.

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
PTK_OpenUSBPort(255);  
PTK_SetReadRFIDBackSpeed(4);  
PTK_CloseUSBPort();
```

## HF tag read/write functions

### *PTK\_RWHFLabel*

Description: writes HF tag data.

Prototype::

```
int _stdcall PTK_RWHFLabel(TCHAR nRWMode, unsigned int nStartBlock, unsigned int nBlockNum, LPTSTR pstr,  
BOOL Variable);
```

Parameters:

nRWMode: RFID operation mode. Reserved parameter; default is 'W'.

nStartBlock: starting block of the operation (first block number).

(Note: for the 15693 protocol, the range runs from 0 to the maximum supported block, with a block size of 4 bytes. For the 14443A protocol there are two restrictions: 1) the manufacturer block (sector 0, block 0) cannot be written; 2) control blocks (in the first 32 sectors: 4\*n-1; from sector 32 onward: 128+16\*n-1, where n = 1/2/3/4 ...) cannot be used as the starting block, so this command always skips control blocks when writing data; the block size is 16 bytes. For NTAG, the range runs from 4 (index 0) to the maximum supported block, with a block size of 4 bytes.)

nBlockNum: number of blocks to operate on.

pstr: a string of 1-100 characters. Users may freely combine "DATA", Cn and Vn into a composite string,

"DATA": a constant string. The double quote (") must be used as the start and end

---

markers, e.g. "POSTEK Printer".

Cn: serial number value. This serial number must have been previously defined.

Vn: variable string. This variable string must have been previously defined.

Example: "text1"Cn"text2"Vn.

Variable: TRUE indicates that the current string contains a variable operation; \" must be added to enclose the constant strings to be printed.

FALSE indicates that the current string does not contain a variable operation and that \" need not be added.

For example: PTK\_RWHFLabel('W', 5, 1, \_T("1234"), FALSE);

has the same effect as PTK\_RWHFLabel('W', 5, 1, \_T("\1234\""), TRUE);

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Note: If the write fails, the label is printed with a VOID marker and the screen displays an RFID read/write error.

**Example:**

**Normal print:**

```
PTK_OpenUSBPort(255);
PTK_RWHFLabel('W', 5, 1, _T("1234"), FALSE);
PTK_PrintLabel(1, 1);
PTK_CloseUSBPort();
```

**Serial number printing:**

```
PTK_OpenUSBPort(255);
PTK_ClearBuffer();
PTK_DefineCounter(0, 6, 'N', _T("+1"), _T("")); // Define serial number CO
PTK_RWHFLabel('W', 5, 1, "CO", TRUE);
PTK_Download();
PTK_DownloadInitVar(_T("1111")); // Initialize CO
PTK_PrintLabel(3, 1); // Start printing labels
PTK_CloseUSBPort();
```

### *PTK\_SetHFRFID*

Description: sets HF tag information.

Prototype::

```
int _stdcall PTK_SetHFRFID(TCHAR pWForm, int nProtocolType, int nMaxErrNumd);
```

Parameters:

WForm: read/write data type.

---

A: ASCII;  
H: hexadecimal mode;  
Default is ASCII format.

nProtocol: protocol type.

0: label type identified during the probing calibration;

1: ISO 15693 protocol;

2: ISO 14443 protocol;

Defaults to the value stored in FLASH; configured via the printer's screen or the

PTK\_SetAllPrinterInfo function.

nMaxErrNumd: number of retry attempts after a read/write failure (not counting the first attempt). Once the maximum number of attempts has been reached, a "Void" marker is printed; if the device has a display, an error message is shown.

Default: 1.

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
PTK_OpenUSBPort(255);  
PTK_SetHFRFID('A', 1, 3);  
PTK_RWHFLabel('W', 5, 1, _T("1234"), FALSE);  
PTK_PrintLabel(1, 1);  
PTK_CloseUSBPort();
```

### *PTK\_ReadHFLabelData*

Description: reads HF tag data.

Note: Supports reading via serial port, USB and network.

Prototype::

```
int _stdcall PTK_ReadHFLabelData(unsigned int nStartBlock, unsigned int nBlockNum, TCHAR pFeed, LPTSTR data, DWORD dataSize);
```

Parameters:

nStartBlock: starting block to read from the tag (first block number).

nBlockNum: number of blocks to read from the tag (nBlockNum blocks starting from nStartBlock).

pFeed: whether to feed one label forward after reading the data;

Y: feed one label after reading the information.

N: do not perform a feed action after reading the information.

data: buffer that stores the HF tag data.

dataSize: size of the data buffer.

---

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
PTK_OpenUSBPort(255);
PTK_ReadHFLabelData(5, 1, 'Y', buff, sizeof(buff));
MessageBox(buff);
PTK_CloseUSBPort();
```

### *PTK\_ReadHFLabeUID*

Description: reads the HF tag UID.

Note: Supports reading via serial port, USB and network.

Prototype::

```
int _stdcall PTK_ReadHFLabeUID(TCHAR pFeed, LPTSTR data, DWORD dataSize);
```

Parameters:

pFeed: whether to feed one label forward after reading the data;

Y: feed one label after reading the information.

N: do not perform a feed action after reading the information.

data: buffer that stores the HF tag UID.

dataSize: size of the data buffer.

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
PTK_OpenUSBPort(255);
PTK_ReadHFLabeUID('Y', buff, sizeof(buff));
MessageBox(buff);
PTK_CloseUSBPort();
```

### *PTK\_ReadHFRFIDSetting*

Description: returns data settings after printing an HF RFID label.

Note: This function is only supported on firmware V10.0 and later.

Prototype::



---

```
int _stdcall PTK_ReadHFRFIDSetting(unsigned int nRMode, unsigned int nStartBlock, unsigned int nRBlockLength);
```

Parameters:

nRMode: select the data area. 0: UID, 1: BLOCK, 2: UID+BLOCK.

Note: when reading the UID (nRMode=0), pass 0 for both nStartBlock and nRBlockLength.

nStartBlock: starting block to read from.

Note: For the 15693 protocol, the range runs from 0 to the maximum supported block (each block is 4 bytes). For the 14443A protocol there is one restriction: the control block (in the first 32 sectors:  $4*n-1$ ; from sector 32 onward:  $128+16*n-1$ , where  $n = 1/2/3/4 \dots$ ) cannot be used as the starting block. Each block is 16 bytes, and the data read out does not include the control block content. For NTAG, the range runs from 0 to the maximum supported block (each block is 4 bytes).

nRBlockLength: length of the block to read. Range: 1-1024.

Description of the returned data format:

(1) Format of the success report:

ZONE1,x1,y1+DATA1\*ZONE2,x2,y2+DATA2\*.....ZONE<sub>n</sub>,xn,yn+DATA<sub>n</sub>

(2) Format of the failure report:

ZONE1,x1,y1+DATA1\*ZONE2,x2,y2+DATA2\*.....ERROR+ZONE<sub>m</sub>,xm,ym+ERRORTYPE

ZONE<sub>n</sub> represents the data area of the n-th read command. [UID; BLOCK; UID+BLOCK]

x<sub>n</sub> represents the starting-block parameter of the n-th read command. When nRMode is 0 (UID) this field is absent;  
y<sub>n</sub> represents the read-block-count parameter of the n-th read command. When nRMode is 0 (UID) this field is absent;

DATA<sub>n</sub> represents the data (hexadecimal) read by the n-th read command;

ERRORTYPE represents the error code. It can take the following categories:

0003 ---> RFID tag could not be scanned;

0004 ---> RFID tag write failed;

0005 ---> read the TID of an already-written tag but could not read the UID of a new tag;

0006 ---> read TIDs from at least two new tags;

0009 ---> UID type mismatch;

0010 ---> RFID feedback error;

0203 ---> RFID tag could not be scanned; trying to print again;

0204 ---> RFID tag write failed; trying to print again;

0205 ---> read the UID of an already-written tag but could not read the UID of a new tag; trying to print again;

0206 ---> read UIDs from at least two new tags; trying to print again;

0209 ---> UID type mismatch; trying to print again;

0210 ---> RFID feedback error; trying to print again;

Report retrieval failed. The LCD shows: RFID FEEDBACK ERROR.

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

---

**Example:**

```
TCHAR data[1024] = { 0 };
TCHAR status[1024] = { 0 };
PTK_OpenUSBPort(255);
PTK_DrawText(20, 20, 0, 6, 2, 2, 'N', _T("313233343536")); // Print text
PTK_RWHFLabel('W', 5, 1, _T("1234"), 0);
PTK_ReadHFRFIDSetting(2, 0, 5);
PTK_PrintAndCallback(data, status);
MessageBox(data);
MessageBox(status);
PTK_CloseUSBPort();
```

### *PTK\_ReadHFTagDataPrintAuto*

Description: during printing, reads the data from a specified block of the HF tag first, then prints it onto the label.

**Note:** This method must be used together with PTK\_DrawTextEx.

**Prototype::**

```
int _stdcall PTK_ReadHFTagDataPrintAuto(unsigned int nStartBlock, unsigned int nBlockNum);
```

**Parameters:**

nStartBlock: starting block to read from the tag (first block number).

nBlockNum: number of blocks to read (nBlockNum blocks starting from nStartBlock).

**Return Value:**

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

**Example:**

```
PTK_OpenUSBPort(255);
PTK_ClearBuffer();
PTK_ReadHFTagDataPrintAuto(5, 1);
PTK_DrawTextEx(50, 30, 0, 2, 1, 1, _T('N'), _T("R"), TRUE);
PTK_PrintLabel(1, 1);
PTK_CloseUSBPort();
```

### *PTK\_ReadHFTagUIDPrintAuto*

Description: during printing, reads the HF tag UID first, then prints it onto the label.

**Note:** This method must be used together with PTK\_DrawTextEx.

**Prototype::**

```
int _stdcall PTK_ReadHFTagUIDPrintAuto(void);
```

---

Parameters:

None

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Example:

```
PTK_OpenUSBPort(255);
PTK_ClearBuffer();
PTK_ReadHFTagUIDPrintAuto();
PTK_DrawTextEx(50, 30, 0, 2, 1, 1, _T('N'), _T("R"), TRUE);
PTK_PrintLabel(1, 1);
PTK_CloseUSBPort();
```

### *PTK\_SetHFAFI*

Description: sets the AFI value of the HF tag.

Prototype::

```
int _stdcall PTK_SetHFAFI(int nAFIValue)
```

Parameters:

nAFIValue: the AFI value to set.

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Note: If the write fails, the label is printed with a VOID marker and the screen displays an RFID read/write error.

Example:

```
PTK_OpenUSBPort(255);
PTK_ClearBuffer();
PTK_SetHFAFI(25);
PTK_PrintLabel(1, 1);
PTK_CloseUSBPort();
```

### *PTK\_SetHFDSFID*

Description: sets the DSFID value of the HF tag.

Prototype::

```
int _stdcall PTK_SetHFDSFID(int nDSFIDValue)
```

Parameters:

nDSFIDValue: the DSFID value to set.

---

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Note: If the write fails, the label is printed with a VOID marker and the screen displays an RFID read/write error.

**Example:**

```
PTK_OpenUSBPort(255);
PTK_ClearBuffer();
PTK_SetHFDSFID(15);
PTK_PrintLabel(1, 1);
PTK_CloseUSBPort();
```

### *PTK\_SetHFEAS*

Description: sets the EAS value of the HF tag.

Prototype::

```
int _stdcall PTK_SetHFEAS(TCHAR EAS)
```

Parameters:

EAS: the EAS value to set. Value: 'E' or 'R'.

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Note: If the write fails, the label is printed with a VOID marker and the screen displays an RFID read/write error.

**Example:**

```
PTK_OpenUSBPort(255);
PTK_ClearBuffer();
PTK_SetHFEAS('E');
PTK_PrintLabel(1, 1);
PTK_CloseUSBPort();
```

### *PTK\_HFDecrypt*

Description: decrypts an HF tag.

**Note:** The tag is an HF 13.56 MHz tag conforming to the ISO 14443A protocol.

Prototype::

```
int _stdcall PTK_HFDecrypt(int key, unsigned int nStartBlock, unsigned int nBlockNum, char* VerifyPassword)
```

Parameters:

---

key: select keyA or keyB for decryption.

1 - keyA

2 - keyB

nStartBlock: starting block for verification.

nBlockNum: number of blocks to verify.

password: decryption password (hexadecimal). Default value: FFFFFFFFFF.

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Note: If the write fails, the label is printed with a VOID marker and the screen displays an RFID read/write error.

Example:

```
PTK_OpenUSBPort(255);  
PTK_ClearBuffer();  
PTK_HFDecrypt(1, 1, 4, "222222222222");  
PTK_PrintLabel(1, 1);  
PTK_CloseUSBPort();
```

### *PTK\_LockHFLabel*

Description: locks an HF tag.

**Note: The tag is an HF 13.56 MHz tag conforming to the ISO 14443A protocol.**

Prototype::

```
int_stdcall PTK_LockHFLabel(int nStartBlock, int nBlockNum, char* keyA, char* keyB, char* nControlByte)
```

Parameters:

nStartBlock: starting block to lock.

(Note: for the 14443A protocol there is one restriction: blocks at index 3\*n, where n = 0/1/2/3/..., cannot be used as the starting block.)

nBlockNum: number of blocks to lock.

keyA: the keyA password used for locking.

keyB: the keyB password used for locking.

nControlByte: control word, in hexadecimal. When NULL, the default value FF078069 is used.

Return Value:

0 -> OK

For other return values, call PTK\_GetErrorInfo to parse them.

Note: If the write fails, the label is printed with a VOID marker and the screen displays an RFID read/write error.

Example:

---

### 1. Encode HF RFID data and lock it

The default first-time password is FFFFFFFFFF, so use this password to decrypt.

```
PTK_OpenUSBPort(255);
PTK_ClearBuffer();
PTK_HFDecrypt(1, 1, 1, "FFFFFFFFFFFF");
PTK_RWHFLabel('W', 5, 1, "1111111166666666", FALSE);
PTK_LockHFLabel(1, 1, _T("FFFFFFFFFFFF"), _T("333333333333"), "FF078069");
PTK_PrintLabel(1, 1);
PTK_CloseUSBPort();
```

### 2. Unlock the locked HF RFID and re-encode it.

#### Using the HF RFID tag locked in step 1 as an example

After an HF RFID tag is locked, the keyA password is 222222222222. Use this password to decrypt; tag data can only be written successfully when the decryption password is correct. Either keyA or keyB can be used for decryption; keyA is normally sufficient.

```
PTK_OpenUSBPort(255);
PTK_ClearBuffer();
PTK_HFDecrypt(1, 1, 1, "222222222222");
PTK_RWHFLabel('W', 5, 1, _T("2222222266666666"), FALSE);
PTK_LockHFLabel(1, 1, _T("FFFFFFFFFFFF"), _T("333333333333"), _T("FF078069"));
PTK_PrintLabel(1, 1);
PTK_CloseUSBPort();
```

### *PTK\_LockHFIdentifier*

Description: locks the AFI / DSFID of a 15693 tag.

Prototype::

```
int _stdcall PTK_LockHFIdentifier(TCHAR Identifier)
```

Parameters:

Identifier: lock identifier

L - lock AFI, U - lock DSFID

Return Value:

0 -> OK

#### Example:

```
PTK_OpenUSBPort(255);
PTK_ClearBuffer();
PTK_RWHFLabel('W', 5, 1, "1111111166666666", FALSE);
PTK_LockHFIdentifier('L');
PTK_PrintLabel(1, 1);
PTK_CloseUSBPort();
```

---

### *PTK\_LockHFBlock*

Description: locks blocks on a 15693 / NTAG tag.

Prototype:

```
int _stdcall PTK_LockHFBlock(unsigned int nStartBlock, unsigned int nBlockNum)
```

Parameters:

nStartBlock: starting block to lock.

nBlockNum: number of blocks to lock.

Return Value:

0 -> OK

Example:

```
PTK_OpenUSBPort(255);  
PTK_ClearBuffer();  
PTK_LockHFBlock(1, 5);  
PTK_RWHFLabel('W', 5, 1, "1111111166666666", FALSE);  
PTK_PrintLabel(1, 1);  
PTK_CloseUSBPort();
```

### *PTK\_SetHFKey*

Description: sets the key.

Prototype:

```
int _stdcall PTK_SetHFKey((int lockType, char* keyA, char* keyB, char* keyFx)
```

Parameters:

lockType: lock setting. 1 - lock after the key has been modified; 0 - do not lock.

keyA: KeyA value.

keyB: keyB value.

keyFx: keyFx value.

Return Value:

0 -> OK

Example:

```
PTK_OpenUSBPort(255);  
PTK_ClearBuffer();  
PTK_SetHFKey(1, "00112233", "44556677", "88990000");  
PTK_RWHFLabel('W', 5, 1, "1111111166666666", FALSE);  
PTK_PrintLabel(1, 1);  
PTK_CloseUSBPort();
```

---

### *PTK\_SetHFCRCCCommand*

Description: verifies, modifies and/or locks the CRC password.

Prototype:

```
int_stdcall PTK_SetHFCRCCCommand((int lockType, char* oldCRCCCommand, char* newCRCCCommand)
```

Parameters:

lockType: lock setting.

1 - after successful verification, lock the CRC password once it has been modified; 0 - do not lock the CRC password.

oldCRCCCommand: old CRC password (hexadecimal), used for verification.

newCRCCCommand: new CRC password (hexadecimal), used for verification.

Return Value:

0 -> OK

Example:

```
PTK_OpenUSBPort(255);
PTK_ClearBuffer();
PTK_SetHFCRCCCommand(0, "11111111", "66666666");
PTK_RWHFLabel('W', 5, 1, "1111111166666666", FALSE);
PTK_PrintLabel(1, 1);
PTK_CloseUSBPort();
```

### *PTK\_SetHFPrivateCommand*

Description: verifies and modifies the private-mode password.

Prototype:

```
int_stdcall PTK_SetHFPrivateCommand((int lockType, char* oldPrivateCommand, char* newPrivateCommand)
```

Parameters:

lockType: lock setting.

1 - after successful verification, lock the private-mode password once it has been modified; 0 - do not lock the private-mode password.

oldPrivateCommand: old private-mode password (hexadecimal), used for verification.

newPrivateCommand: new private-mode password (hexadecimal), used for verification.

Return Value:

0 -> OK

Example:

```
PTK_OpenUSBPort(255);
PTK_ClearBuffer();
PTK_SetHFPrivateCommand(0, "11111111", "66666666");
PTK_RWHFLabel('W', 5, 1, "1111111166666666", FALSE);
```



---

```
PTK_PrintLabel(1, 1);  
PTK_CloseUSBPort();
```

### *PTK\_LockHFUser*

Description: locks the user area.

Prototype:

```
int _stdcall PTK_LockHFUser((int lockType, int nStartBlock, int nBlockNum)
```

Parameters:

lockType: lock setting.

1 - enable locking; 0 - do not enable locking.

nStartBlock: starting block to lock.

nBlockNum: number of blocks to lock.

Return Value:

0 -> OK

Example:

```
PTK_OpenUSBPort(255);  
PTK_ClearBuffer();  
PTK_LockHFUser(0, 1, 2);  
PTK_RWHFLabel('W', 5, 1, "1111111166666666", FALSE);  
PTK_PrintLabel(1, 1);  
PTK_CloseUSBPort();
```

### *PTK\_SetHFCFG10*

Description: sets the CFG Set 0x10 function parameter.

Prototype:

```
int _stdcall PTK_SetHFCFG10((char* CFG_Set_0x10)
```

Parameters:

CFG\_Set\_0x10: CFG Set 0x10 function parameter, in hexadecimal format.

Return Value:

0 -> OK

Example:

```
PTK_OpenUSBPort(255);  
PTK_ClearBuffer();  
PTK_SetHFCFG10("1111111166666666");  
PTK_RWHFLabel('W', 5, 1, "1111111166666666", FALSE);  
PTK_PrintLabel(1, 1);  
PTK_CloseUSBPort();
```

---

### *PTK\_SetHFCFG80*

Description: sets the CFG Set 0x80 function parameter.

Prototype:

```
int _stdcall PTK_SetHFCFG80((char* CFG_Set_0x80)
```

Parameters:

CFG\_Set\_0x80: CFG Set 0x80 function parameter, in hexadecimal format.

Return Value:

0 -> OK

Example:

```
PTK_OpenUSBPort(255);  
PTK_ClearBuffer();  
PTK_SetHFCFG80("1111111666666666");  
PTK_RWHFLabel('W', 5, 1, "1111111666666666", FALSE);  
PTK_PrintLabel(1, 1);  
PTK_CloseUSBPort();
```

---

## Legacy interface (kept for backward compatibility)

Note: The following function interfaces are maintenance-only and are no longer updated.

### OpenPort

#### Description:

The **OpenPort** function opens a communication port.

**Before using any other function in this library, OpenPort must be called successfully.**

#### Prototype:

```
int OpenPort(int PortFlag);
```

#### Parameters:

xx: communication port identifier;

0: prints to the file PBuffi.txt (file is created in the executable's directory);

1: opens LPT1.

2: opens LPT2.

3: opens LPT3.

4: opens COM1.

5: opens COM2.

6: opens COM3.

7: opens USB001.

8: opens USB002.

9: opens USB003.

.....

255: when there is one and only one POSTEK printer, opens that POSTEK printer by default.

#### Return Value:

0 -> OK;

For other return values, see the section "CDFPSK.dll error return code descriptions".

#### Example:

```
OpenPort(1); // Opens the LPT1 port.
```

### ClosePort

#### Description:

The **ClosePort** function closes a communication port that was opened with **OpenPort**.

---

After the user finishes operating the printer, it is recommended to call ClosePort to close the communication port;  
Otherwise the user's program will continue to hold the opened communication port until the program exits.

Prototype:

Void ClosePort(void);

Parameters: none

Return Value: none

Example:

ClosePort();

SetPCComPort

**Description:**

**The SetPCComPort function sets the baud rate of the serial port on the PC.**

**This function is only effective when communicating via a serial port.**

**Note:** must match the serial port baud rate selected on the printer. (If the printer has DIP switches, configure pins 7 and 8 as described in the user manual; if there are no DIP switches, configure it via POSTEK Utility 3.0 or contact customer service for instructions.)

Prototype:

int SetPCComPort(DWORD BaudRate, BOOL HandShake);

Parameters:

BaudRate: serial port baud rate to set. Valid values:  
9600, 19200, 38400, 57600;

HandShake: whether to use hardware handshaking;

TRUE: hardware handshaking is enabled,

FALSE: hardware handshaking is disabled.

Return Value:

0 -> OK;

For other return values, see the section "CDFPSK.dll error return code descriptions".

Example: SetPCComPort ( 9600, TRUE);

---

## PTK\_Reset

### Description:

#### The PTK\_Reset function resets the printer.

This command is equivalent to powering the printer off and on again; the system re-initializes itself.

This command has no effect in the following cases:

- While a soft font or PCX image is being downloaded, or while the printer is in DUMP state;

- Use this command inside a FORM;

- While the printer is executing a print job;

In addition, after this function is executed the printer's initialization may take more than 2 seconds, during which the printer does not accept any control commands.

### Prototype:

```
int PTK_Reset(void);
```

Parameters: none

Return Value:

- 0 -> OK;

For other return values, see the section "CDFPSK.dll error return code descriptions".

Example:

```
PTK_Reset( );
```

## PTK\_DrawTextTrueTypeW

### Description:

PTK\_DrawTextTrueTypeW prints a single line of TrueType font text and allows the text width and height to be fine-tuned.

Prototype:

```
int PTK_DrawTextTrueTypeW(int x, int y,  
                           unsigned int FHeight, unsigned int FWidth,  
                           LPCTSTR FType, unsigned int Fspin,  
                           unsigned int FWeight, BOOL FItalic,  
                           BOOL FUnline, BOOL FStrikeOut,  
                           LPTSTR id_name, LPCTSTR data);
```

Parameters:

- x: set the X coordinate, in dots;

- y: set the X coordinate, in dots;

---

FHeight: font height, in dots;

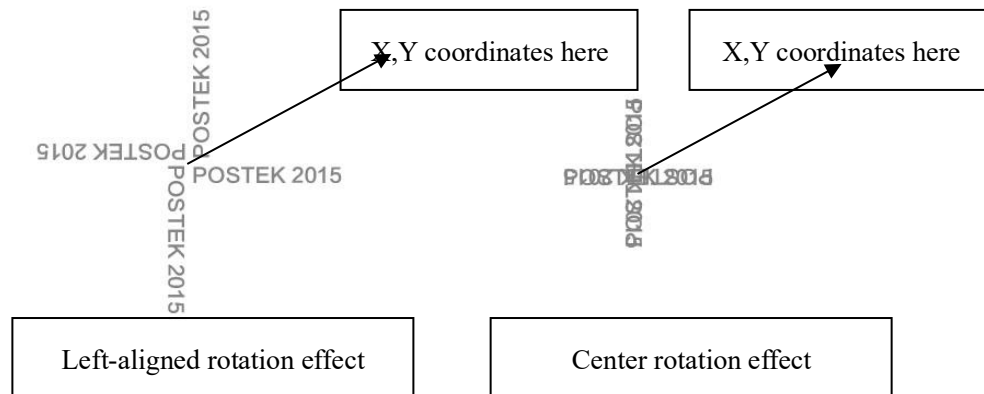
FWidth: font width, in dots;

**\* To print a font at normal proportions, set FWidth to 0;**

FType: font name;

Fspin: text rotation angle. 1 -> left-aligned 0 deg, 2 -> left-aligned 90 deg, 3 -> left-aligned 180 deg, 4 -> left-aligned 270 deg.

5 -> centered 0 deg, 6 -> centered 90 deg, 7 -> centered 180 deg, 8 -> centered 270 deg



Fweight: font weight.

0 and 400 -> 400 Normal,

100 -> Very Thin, 200 -> Extra Thin,

300 -> Thin , 500 -> Medium,

600 -> Semi Bold , 700 -> Bold ,

800 -> Extra Bold , 900 -> Black.

Fitalic: italic. 0 -> FALSE, 1 -> TRUE;

Funderline: underline text. 0 -> FALSE, 1 -> TRUE;

FstrikeOut: strikethrough text. 0 -> FALSE, 1 -> TRUE;

id\_name: identifier. A single line of TrueType text is converted into PCX-format data and stored in the printer under id\_name; before power is removed, this line of text can be printed any number of times by calling PTK\_DrawPcxGraphics() with id\_name. (When the data parameter or any other parameter differs, be sure to use a different id\_name value.)

data: string content.

Return value: 0 -> OK;

For other return values, see the section "CDFPSK.dll error return code descriptions".

Example: print 3 mm tall Chinese characters:

For a 203 DPI printer, set FHeight to  $3 / 0.125 = 24$  dots;

---

For a 300 DPI printer, set FHeight to  $3 / 0.08 = 38$  dots (rounded).

`PTK_PcxGraphicsDel("A1")` or `PTK_PcxGraphicsDel("*");` // Recommended to call before printing TrueType fonts

`PTK_DrawTextTrueTypeW (30,35,24,0,"SimSun",4,400,0,0,0,"A1","Top Secret");`

## PTK\_PcxGraphicsDownload

### Description:

**The PTK\_PcxGraphicsDownload function stores a PCX-format image into the printer.**

### Prototype:

`int PTK_PcxGraphicsDownload (LPTSTR pcxname, LPTSTR pcxpath);`

### Parameters:

`pcxname`: user-defined name for the image. Maximum length: 16 characters. After the image has been stored on the printer, this name must be passed to `PTK_DrawPcxGraphics()` to retrieve and print the image.

`pcxpath`: path of the PCX image file on the PC.

Return Value: 0 → OK;

For other return values, see the section "CDFPSK.dll error return code descriptions".

### Example:

`PTK_PcxGraphicsDownload ("PCXA", "c:\\test1111.pcx");`

## PTK\_PrintPCX

### Description:

**The PTK\_PrintPCX function prints a PCX-format image.**

**This function combines PTK\_PcxGraphicsDownload() and PTK\_DrawPcxGraphics() into a single call.**

### Prototype:

`int PTK_PrintPCX (unsigned int px, unsigned int py, LPTSTR filename);`

### Parameters:

`px`: set the X coordinate, in dots;

`py`: set the Y coordinate, in dots;

`filename`: PCX image file name, optionally including a file path.

Format example: "XXXXXXXX.XXX" or "X:\\XXX\\XXX.PCX".

---

Return Value: 0     -> OK;

For other return values, see the section "CDFPSK.dll error return code descriptions".

Example:

```
PTK_PrintPCX (10, 100, "c:\\phone.pcx");
```

## PTK\_SoftFontList

### Description:

The **PTK\_SoftFontList** function prints a list of the soft fonts stored in the printer's RAM or FLASH memory.

Prototype:

```
int PTK_SoftFontList (void);
```

Parameters: none

Return Value: 0     -> OK;

For other return values, see the section "CDFPSK.dll error return code descriptions".

Example:

```
PTK_SoftFontList ();
```

## PTK\_SoftFontDel

### Description:

The **PTK\_SoftFontDel** function deletes one or all soft fonts stored in the printer's RAM or FLASH memory.

Prototype:

```
int PTK_SoftFontDel (TCHAR pid);
```

Parameters:

pid: soft font ID. Range: A-Z or \*;

If pid = '\*', the printer deletes all soft fonts stored in RAM or FLASH memory.

Return Value: 0     -> OK;

For other return values, see the section "CDFPSK.dll error return code descriptions".

Example:



---

```
PTK_SoftFontDel ('A');
```

## PTK\_SetFontGap

### Description:

The PTK\_SetFontGap function adjusts the character spacing of printed text.

### Prototype:

```
int PTK_SetFontGap(int gap);
```

### Parameters:

gap: character spacing adjustment, in dots. Range: -99 to 99.

**Note: The printer's built-in fonts (including downloaded fonts) have a default character spacing. The g command can be used to adjust the character spacing, where actual spacing = default spacing + adjustment. This command only takes effect on models that support character-spacing adjustment.**

**Tip: The unit conversions for the various resolutions are as follows:**

203DPI: 1mm = 8 dot;

300DPI: 1mm = 11.8dot;

600DPI: 1mm = 23.6 dot;

For example, to set the character spacing to 1 mm, set it to 8 at 203 DPI, 12 at 300 DPI, and 24 at 600 DPI.

### Return Value:

0 -> OK;

For other return values, see the section "CDFPSK.dll error return code descriptions".

### Example:

Example 1:

```
int return = PTK_SetFontGap (10);
```

---

## PTK\_GetUSBID

### Description:

**The PTK\_GetUSBID function retrieves the printer's USB ID.**

### Prototype:

```
int PTK_GetUSBID (LPTSTR USBDeviceSerial);
```

### Parameters:

USBDeviceSerial: stores the retrieved USB ID.

### Return Value:

0      -> OK;

For other return values, see the section "CDFPSK.dll error return code descriptions".

***Note: The USB ID can only be retrieved over a USB port.***

### Example:

```
TCHAR  USBIDString[10] = {_T('0')};  
PTK_GetUSBID (USBIDString);
```

## PTK\_DisableBackFeed

### Description:

**The PTK\_DisableBackFeed function disables the print backfeed feature.**

### Prototype:

```
int PTK_DisableBackFeed(void);
```

### Parameters: none

Return Value: 0      -> OK;

For other return values, see the section "CDFPSK.dll error return code descriptions".

### Example:

```
PTK_DisableBackFeed ( );
```

---

## PTK\_EnableBackFeed

### Description:

**The PTK\_EnableBackFeed function enables the print backfeed feature.**

### Prototype:

```
int PTK_EnableBackFeed (unsigned int distance);
```

### Parameters:

distance: rollback distance, in dots.

Return Value: 0 -> OK;

For other return values, see the section "CDFPSK.dll error return code descriptions".

### Example:

```
PTK_EnableBackFeed (140);
```

## PTK\_SetPrinterState

### Description:

**The PTK\_SetPrinterState function sets the printer's operating state.**

### Prototype:

```
int PTK_SetPrinterState (char* state);
```

Parameters: state takes one of the following characters:

D: set the printer to direct thermal (thermal transfer) mode;

P: configure the printer for continuous media feeding (default);

L: configure the printer to pause after printing each label and wait for user confirmation before printing the next label;

(Confirmation methods: 1. press the "FEED" key; 2. with a peeler installed, when the user takes the label

the next label is printed automatically.)

CN: cutting state. Only valid when the cut frequency is set to 0, and is reset at the end of each print job.

N is the print quantity. Range: 1~30000.

**When N is omitted, one cut is performed after each label; when N is specified, the label is cut after every N labels are printed.**

N: configure the printer for a peel-off installation.

### Note:

1. The cutter and the peeler cannot be installed at the same time;

- 
2. If the printer state is configured incorrectly, the READY indicator on the printer's front panel will flash. See the troubleshooting section of the printer user manual.

Return Value: 0 → OK;

For other return values, see the section "CDFPSK.dll error return code descriptions".

Example:

```
PTK_SetPrinterState ("C10");
```

## PTK\_DisableErrorReport

### Description:

**The PTK\_DisableErrorReport function disables error reporting.**

Prototype:

```
int PTK_DisableErrorReport(void);
```

Parameters: none

Return Value:

0 → OK;

For other return values, see the section "CDFPSK.dll error return code descriptions".

Example: PTK\_DisableErrorReport( );

## PTK\_EnableErrorReport

### Description:

**The PTK\_EnableErrorReport function enables error reporting.**

The printer's reply data is returned to the computer via the RS-232 serial port.

If an error occurs during printing, the printer first sends a NACK (15H) character back to the computer, followed by the error code.

If no error occurs, the printer sends an ACK (06H) character after receiving the P command.

Error code	Description
0x00	No Error
0x01	Object Exceeded Label Border
0x02	Bar Code Data Length Error
0x03	Insufficient Memory to Store Data
0x04	Memory Configuration Error
0x05	RS-232 Interface Error
0x06	Paper or Ribbon Empty
0x07	Duplicate Name: Form, Graphic or Soft Font

0x08	Name Not Found: Form, Graphic or Soft Font
0x09	Not in Data Entry Mode
0x0a	Print Head Up (Open)
0x0b	Pause Mode or Paused in Peel mode
0x0c	Does not fit in area specified
0x0d	Data length to long
0x0c	PDF-417 coded data to large to fit in bar code
0x0d	
0x0e	

Prototype:

```
int PTK_ EnableErrorReport (void );
```

Parameters: none

Return Value:

0 -> OK;

For other return values, see the section "CDFPSK.dll error return code descriptions".

Example:

```
PTK_ EnableErrorReport ( );
```

## PTK\_FeedBack

### Description:

**The PTK\_FeedBack function asks the printer to immediately return an error report.**

Users can use this command to immediately determine the printer's current error status. The printer returns 4 bytes to the host:

0xXX XX 0x0d 0x0a :Error/Status code <CR><LF>

Error/Status code	Description
00	No error
01	Syntax error
82	Ribbon detection error.
83	Label detection error.
86	Cutter detection error.
87	Print head is open.
88	Paused
99	Other errors

Prototype:

---

```
int PTK_FeedBack (void);
```

Parameters: none

Return Value:

0      -> OK;

For other return values, see the section "CDFPSK.dll error return code descriptions".

Example:

```
PTK_FeedBack ( );
```

## PTK\_ErrorReport

### Description:

The PTK\_ErrorReport function sends an error-query command to the printer and receives and parses the printer's current error code from the specified serial port.

**Before calling this function, if the program already has a send port open, ClosePort() must be called first to close it.**

Users can use this command to immediately determine the printer's current error status. The printer returns 4 bytes to the host:

0xXX XX 0x0d 0x0a

:Error/Status code <CR><LF>

Error/Status code	Description	PTK_ErrorReport() return value	ErrorCode
00	No error	0	"00"
01	Syntax error	1	"01"
82	Ribbon detection error.	82	"82"
83	Label detection error.	83	"83"
86	Cutter detection error.	86	"86"
87	Print head is open.	87	"87"
88	Paused	88	"88"
99	Other errors	99	"99"

Prototype:

```
int PTK_ErrorReport (int wPort, int rPort, DWORD BaudRate, BOOL HandShake, int TimeOut);
```

Parameters:

wPort: port to which the data will be sent;

0:      prints to the file PBuffi.txt (file is created in the executable's directory); (Do NOT use this port!)

1:      opens LPT1;

2:      opens LPT2;

---

3: opens LPT3;  
4: opens COM1;  
5: opens COM2;  
6: opens COM3.

rPort: port through which the printer's current error status code is received;

1: opens COM1 as the receive port;  
2: opens COM2 as the receive port;  
3: opens COM3 as the receive port;

BaudRate: serial port baud rate to set. Valid values:  
9600, 19200, 38400, 57600;

HandShake: whether to use hardware handshaking;

TRUE: hardware handshaking is enabled,

FALSE: hardware handshaking is disabled.

TimeOut: serial-port receive timeout. Units: 100 ms.

Return Value:

>0 -> OK;

For other return values, see the section "CDFPSK.dll error return code descriptions".

Example:

1. Send the command via LPT1, and read the current error status code via COM1:

OpenPort(1);

....

ClosePort();

PTK\_ErrorReport (1, 1, 9600, TRUE, 10);

.....

2. Send the command via COM1, and read the current error status code via COM1:

OpenPort(4);

....

ClosePort();

PTK\_ErrorReport (4, 1, 9600, TRUE, 10 );

.....

.....

PTK\_ErrorReportUSB

**The PTK\_ErrorReportUSB function sends an error-query command to the printer and receives and parses the printer's current error code from the specified USB port.**

**Before calling this function, if the program already has a send port open, ClosePort() must be called first to close it.**

Error/Status code	Description	PTK_ErrorReport() return value	ErrorCode
00	No error	0	"00"
01	Syntax error	1	"01"
82	Ribbon detection error.	82	"82"
83	Label detection error.	83	"83"
86	Cutter detection error.	86	"86"
87	Print head is open.	87	"87"
88	Paused	88	"88"
99	Other errors	99	"99"

Prototype:

```
int PTK_ErrorReportUSB(int USBport);
```

Parameters:

USBport: port to which the data will be sent;

1: opens USB001.

2: opens USB002.

3: opens USB003.

.....

255: when there is one and only one POSTEK printer, opens that POSTEK printer by

default.

Return Value:

Returns one of the printer status codes listed above.

Error/Status code
00
01
82
83
86
87
88
99

For other return values, see the section "CDFPSK.dll error return code descriptions".

Example: PTK\_ErrorReportUSB(255);

```
OpenPort(255);
```

```
....
```

```
ClosePort();
```



---

## PTK\_ErrorReportNet

The **PTK\_ErrorReportNet** function sends an error-query command to the printer over the specified TCP/IP port and receives and parses the printer's current error code.

Error/Status code	Description	PTK_ErrorReport() return value	ErrorCode
00	No error	0	"00"
01	Syntax error	1	"01"
82	Ribbon detection error.	82	"82"
83	Label detection error.	83	"83"
86	Cutter detection error.	86	"86"
87	Print head is open.	87	"87"
88	Paused	88	"88"
99	Other errors	99	"99"

Prototype:

```
int PTK_ErrorReportNet(LPTSTR PrintIPAddress, unsigned int PrinterNetPort);
```

Parameters:

PrintIPAddress: printer IP address.

PrinterNetPort: printer network port.

Return Value:

Returns one of the printer status codes listed above.

Error/Status code
00
01
82
83
86
87
88
99

For other return values, see the section "CDFPSK.dll error return code descriptions".

Example: int nPrintStatus;

```
nPrintStatus = PTK_ReadSerialNumberNet ("199.9.10.230",9100);
```

---

## PTK\_ReadPrintConifUSB

### Description:

The PTK\_ReadPrintConifUSB function reports printer information back to the host over the USB port (see parameter 2 for the contents of the report).

### Prototype:

```
int PTK_ReadPrintConifUSB(unsigned int usbPort,unsigned int nInfoType,LPTSTR strData)
```

### Parameters:

usbPort: USB port number to send the report through;

nInfoType: type of the report information; see the table below.

p2	Description of the report format
1	Printer model, software version, firmware identifier, printer resolution,
2	Print method (0 direct thermal / 1 thermal transfer), media sensing mode (0 see-through / 1 bottom reflective / 2 top reflective), ribbon detector (1 enabled / 0 disabled), tear-off mode (1 on / 0 off), cut mode (1 on / 0 off), print darkness (1~20).
3	IP ADDRESS, SUBNET MASK, GATEWAY, BASE RAW PORT, MAC ADDRESS
4	Print line to sensor distance, cutter to print line distance, tear bar to print line distance, distance from the start of one label to the start of the next.
5	RFID (1 = enabled / 0 = disabled), RFID POWER, RFID probing offset distance (in mm), current RFID tag height (in dots).

strData: buffer that receives the retrieved printer information.

### Return Value:

0 -> OK;

For other return values, see the section "CDFPSK.dll error return code descriptions".

Example: LPTSTR strData;

```
PTK_ReadPrintConifUSB(255,1, strData);
```

## PTK\_ReadPrintConifNet

### Description:

The PTK\_ReadPrintConifNet function reports printer information back to the host over the network port (see parameter 2 for the contents of the report).

---

Prototype:

```
int PTK_ReadPrintConifNet(LPTSTR PrintIPAddress, unsigned int PrinterNetPort,  
                          unsigned int nInfoType,LPTSTR strData)
```

Parameters:

PrintIPAddress: printer IP address.

PrinterNetPort: printer network port.

nInfoType: type of the report information; see the table below.

p2	Description of the report format
1	Printer model, software version, firmware identifier, printer resolution,
2	Print method (0 direct thermal / 1 thermal transfer), media sensing mode (0 see-through / 1 bottom reflective / 2 top reflective), ribbon detector (1 enabled / 0 disabled), tear-off mode (1 on / 0 off), cut mode (1 on / 0 off), print darkness (1~20).
3	IP ADDRESS, SUBNET MASK, GATEWAY, BASE RAW PORT, MAC ADDRESS
4	Print line to sensor distance, cutter to print line distance, tear bar to print line distance, distance from the start of one label to the start of the next.
5	RFID (1 = enabled / 0 = disabled), RFID POWER, RFID probing offset distance (in mm), current RFID tag height (in dots).

**Note:** When using the network reporting feature, by default the host that receives the report is the host IP address of the current connection. To change the reporting IP, use this together with NF, as shown in Example 1.

Return Value:

0      -> OK;

For other return values, see the section "CDFPSK.dll error return code descriptions".

Example: LPTSTR strData;

```
PTK_ReadPrintConifNet("199.9.10.196",1, strData);
```

## PTK\_ReadRFTagData

### Description:

The PTK\_ReadRFTagData function sends a read-RFID-tag-data command to the printer and receives the currently read RFID tag information from the specified serial port.

**Before calling this function, if the program already has a send port open, ClosePort() must be called first to close it.**

---

**Prototype:**

int PTK\_ReadRFTagData(unsigned int wPort, unsigned int rPort, DWORD BaudRate, BOOL HandShake, unsigned int TimeOut, unsigned int nDataBlock, unsigned int nRFPower, BOOL bFeed, LPTSTR strRFData);

**Parameters:**

wPort: port to which the data will be sent;

0: prints to the file PBuffi.txt (file is created in the executable's directory); (Do NOT use this port!)

1: opens LPT1;

2: opens LPT2;

3: opens LPT3;

4: opens COM1;

5: opens COM2;

6: opens COM3.

rPort: port through which the printer's current error status code is received;

1: opens COM1 as the receive port;

2: opens COM2 as the receive port;

3: opens COM3 as the receive port;

BaudRate: serial port baud rate to set. Valid values:

9600, 19200, 38400, 57600;

HandShake: whether to use hardware handshaking;

TRUE: hardware handshaking is enabled,

FALSE: hardware handshaking is disabled.

TimeOut: serial-port receive timeout. Units: ms.

nDataBlock: select the data area. 0: TID, 1: EPC, 2: TID+EPC.

nRFPower: read power. Range: 0~30 dBm. When set to 0, the system default of 23 dBm is used;

bFeed: whether to feed one label forward after reading;

TRUE: feed one label forward is enabled,

FALSE: feed one label forward is disabled.

strRFData: buffer used to store the retrieved RFID tag data.

**Return Value:**

0 -> OK;

For other return values, see the section "CDFPSK.dll error return code descriptions".

**Example:**

1. Send the print command via LPT1, and read the current RFID tag EPC information via COM1:

LPTSTR strRFData;

OpenPort(1);

....

---

```
ClosePort();
PTK_ReadRFTagData (1, 1, 38400, TRUE, 300,1,0,TRUE,strRFData);
```

```
.....
```

2. Send the command via COM1, and read the current RFID tag EPC information via COM1:

```
LPTSTR strRFData;
```

```
OpenPort(4);
```

```
....
```

```
ClosePort();
```

```
PTK_ReadRFTagData (4, 1, 38400, TRUE, 300,1,0, TRUE,strRFData);
```

```
.....
```

## PTK\_ReadRFTagDataUSB

### Description:

The PTK\_ReadRFTagDataUSB function sends a read-RFID-tag-data command to the printer over the specified USB port and receives the RFID tag information that the printer has currently read.

**Before calling this function, if the program already has a send port open, ClosePort() must be called first to close it.**

### Prototype:

```
int PTK_ReadRFTagDataUSB(unsigned int usbPort, unsigned int nDataBlock, unsigned int nRFPower,
                          BOOL bFeed, LPTSTR strRFData);
```

### Parameters:

usbPort: port to which the data will be sent;

1: opens USB001.

2: opens USB002.

3: opens USB003.

.....

255: when there is one and only one POSTEK printer, opens that POSTEK printer by

default.

nDataBlock: select the data area. 0: TID, 1: EPC, 2: TID+EPC.

nRFPower: read power. Range: 0~30 dBm. When set to 0, the system default of 23 dBm is used;

bFeed: whether to feed one label forward after reading;

TRUE: feed one label forward is enabled,

FALSE: feed one label forward is disabled.

strRFData: buffer used to store the retrieved RFID tag data.

### Return Value:

---

0     -> OK;

For other return values, see the section "CDFPSK.dll error return code descriptions".

Example:

```
LPTSTR strRFData;  
OpenPort(255);  
....  
ClosePort();  
PTK_ReadRFTagDataUSB(255, 1,0,TRUE,strRFData);
```

PTK\_ReadRFTagDataNet

### Description:

The PTK\_ReadRFTagDataNet function sends a read-RFID-tag-data command to the printer over the specified TCP/IP port and receives the RFID tag information that the printer has currently read.

### Prototype:

```
int PTK_ReadRFTagDataNet(LPTSTR IPAddress, unsigned int Port, unsigned int nFeedbackPort,  
                        unsigned int nRFPower, BOOL bFeed, LPTSTR strRFData);
```

Parameters:

IPAddress: printer IP address.  
Port:       printer network port.  
nDataBlock: select the data area. 0: TID, 1: EPC, 2: TID+EPC.  
nRFPower: read power. Range: 0~30 dBm. When set to 0, the system default of 23 dBm is used;  
bFeed:      whether to feed one label forward after reading;  
            TRUE: feed one label forward is enabled,  
            FALSE: feed one label forward is disabled.  
strRFData:  buffer used to store the retrieved RFID tag data.

Return Value:

0     -> OK;

For other return values, see the section "CDFPSK.dll error return code descriptions".

Example:

```
LPTSTR strRFData;  
PTK_ReadRFTagDataNet("199.9.10.230",9100,1, 0,TRUE,strRFData);
```

---

## PTK\_ReadHFTagDataUSB

### Description

The PTK\_ReadHFTagDataUSB function sends a read-HF-RFID-tag-data command to the printer over the specified USB port and receives the RFID tag information that the printer has currently read.

Prototype:

```
int PTK_ReadHFTagDataUSB(unsigned int usbPort,unsigned int nStartBlock,  
                          unsigned int nBlockNum,TCHAR pFeed,LPTSTR strRFData);
```

### Parameters

usbPort: port to which the data will be sent;

1: opens USB001.

2: opens USB002.

3: opens USB003.

.....

255: when there is one and only one POSTEK printer, opens that POSTEK printer by default.

nStartBlock: starting block to read from the tag (first block number).

nBlockNum: number of blocks to read from the tag (nBlockNum blocks starting from nStartBlock).

bFeed: whether to feed one label forward after reading the data;

Y: feed one label after reading the information.

N: do not perform a feed action after reading the information.

strRFData: buffer used to store the retrieved RFID tag data.

Return Value: 0 -> OK;

For other return values, see the section "CDFPSK.dll error return code descriptions".

Example:

```
LPTSTR strRFData;
```

```
PTK_ReadHFTagDataUSB(255,5,1,_T('Y'),strRFData);
```

Example description: read 1 block starting from block 5 (i.e. block 5) and return the data over the USB port; advance one label after returning.

```
LPTSTR strRFData;
```

```
PTK_ReadHFTagDataUSB(255,5,5,_T('N'),strRFData);
```

Example description: read 5 blocks starting from block 5 (blocks 5, 6, 7, 8, 9) and return the data over the USB port; do not advance a label after returning.

## PTK\_ReadHFTagUID

### Description

---

The PTK\_ReadHFTagUID function receives, from the specified serial port, the UID of the HF RFID tag that the printer has currently read.

Prototype:

```
int PTK_ReadHFTagUID(unsigned int wPort, unsigned int rPort, DWORD BaudRate,  
                     BOOL HandShake, unsigned int TimeOut, TCHAR pFeed, LPTSTR strRFData);
```

Parameters

wPort: port to which the data will be sent;

0: prints to the file PBuff.txt (file is created in the executable's directory); (Do NOT use this port!)

1: opens LPT1;

2: opens LPT2;

3: opens LPT3;

4: opens COM1;

5: opens COM2;

6: opens COM3.

rPort: port through which the printer's current error status code is received;

1: opens COM1 as the receive port;

2: opens COM2 as the receive port;

3: opens COM3 as the receive port;

BaudRate: serial port baud rate to set. Valid values:

9600, 19200, 38400, 57600;

HandShake: whether to use hardware handshaking;

TRUE: hardware handshaking is enabled,

FALSE: hardware handshaking is disabled.

TimeOut: serial-port receive timeout. Units: ms.

bFeed: whether to feed one label forward after reading the data;

Y: feed one label after reading the information.

N: do not perform a feed action after reading the information.

strRFData: buffer used to store the retrieved RFID tag data.

Return Value: 0 -> OK;

For other return values, see the section "CDFPSK.dll error return code descriptions".

Example:

1. Send the print command via LPT1, and read the current HF RFID tag UID via COM1:

LPTSTR strRFData;

PTK\_ReadHFTagUID (1, 1, 38400, TRUE, 300, \_T('Y') , strRFData);

Example description: read the current HF RFID tag's UID and return it over the serial port; advance one label



---

after returning.

.....

2. Send the command via COM1, and read the current HF RFID tag data via COM1:

LPTSTR strRFData;

PTK\_ReadHFTagUID (4, 1, 38400, TRUE, 300, \_T('N') , strRFData);

Example description: read the current HF RFID tag's UID and return it over the serial port; do not advance a label after returning.

## PTK\_ReadHFTagUIDUSB

### Description

The PTK\_ReadHFTagUIDUSB function sends a read-HF-RFID-tag-data command to the printer over the specified USB port and receives the UID of the RFID tag that the printer has currently read.

Prototype:

int PTK\_ReadHFTagUIDUSB(unsigned int usbPort, TCHAR pFeed, LPTSTR strRFData);

Parameters:

usbPort: port to which the data will be sent;

1: opens USB001.

2: opens USB002.

3: opens USB003.

.....

255: when there is one and only one POSTEK printer, opens that POSTEK printer by default.

bFeed: whether to feed one label forward after reading the data;

Y: feed one label after reading the information.

N: do not perform a feed action after reading the information.

strRFData: buffer used to store the retrieved RFID tag data.

Return value: 0 -> OK;

For other return values, see the section "CDFPSK.dll error return code descriptions".

Example:

LPTSTR strRFData;

PTK\_ReadHFTagUIDUSB (255, \_T('Y'),strRFData);

Example description: read the current HF RFID tag's UID and return it over the USB port; advance one label after returning.

LPTSTR strRFData;

PTK\_ReadHFTagUIDUSB (255, \_T('N'),strRFData);

Example description: read the current HF RFID tag's UID and return it over the USB port; do not advance a label

---

after returning.

## PTK\_GetPrinterDPI

### Description

Retrieves the printer resolution (dpi).

Prototype:

```
int _stdcall PTK_GetPrinterDPI(int* dpi);
```

Parameters:

`int* dpi` pointer that receives the printer's DPI.

Example: the resolution of a 200-dot printer is 203 dpi, so 1 mm = 8 dots.

The resolution of a 300-dot printer is 300 dpi, so 1 mm = 12 dots.

The resolution of a 600-dot printer is 600 dpi, so 1 mm = 24 dots.

Return value:

0 -> OK;

For other return values, see the section "CDFPSK.dll error return code descriptions".

Example:

```
int dpi = 0;
```

```
OpenPort(255);
```

```
PTK_GetPrinterDPI(&dpi);
```

```
ClosePort();
```

## PTK\_SetTearAndTphOffset

### Description

The PTK\_SetTearAndTphOffset function sets the printer's tear-off offset and top-of-page (TPH) offset.

**Note: This method writes the relevant settings into the printer's FLASH parameter area. The operation takes about 10 seconds, and frequent FLASH erase/write cycles shorten the printer's lifetime, so this function should only be called when necessary.**

Prototype:

```
int _stdcall PTK_SetTearAndTphOffset(float tear_offset, float tph_offset)
```

Parameters:

`tear_offset`: tear-off offset, in mm.

`tph_offset`: top-of-page (TPH) offset, in mm.

Return value:

---

0     -> OK;

For other return values, see the table below and the section "CDFPSK.dll error return code descriptions".

Example:

```
int dpi = 0;
OpenPort(255);
PTK_SetTearAndTphOffset (-2.5,3);// Set the printer's tear-off offset to -2.5 mm and the top-of-page offset
to 3 mm
ClosePort();
```

## Appendix

Table – Printer status code descriptions

Error/Status code	Description
000	No error
001	Syntax error
004	Print head is heating up (only on industrial-class printers).
082	Ribbon detection error.
083	Label detection error.
086	Cutter detection error.
087	Print head is open.
088	Paused
108	RF command (setting the RFID write method and data area) failed; invalid parameter supplied.
109	Failed to write data to RFID tag; retry count exhausted.
110	Data write failed, but the retry count has not been exhausted.
111	RFID tag calibration failed.
112	RI command (setting the RFID read method and data area) failed; invalid parameter supplied.
116	Failed to read RFID tag data.

Table – 0X-series printer status code descriptions

31	30	29	28	27	26	25	24
0: system is idle. 1: system is busy.							
23	22	21	20	19	18	17	16
15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0

			0: no label above the peel-off sensor. 1: a label is above the peel-off sensor.	0: print head is down. 1: print head is raised.	Printer status 0: Ready. 1: Printing. 2: Paused. 3: Deleting. 4: Error.
--	--	--	--	--	--

Table – Country / region frequency bands

Region Name	Country or Region	Serial Interface Region Code
NA	North America	1
NA2	North America	13
NA3	North America	14
IN	India	4
JP	Japan	5
PRC	China	6
EU3	Europe	8
KR2	Korea	9
AU	Australia	11
NZ	New Zealand	12
MY	Malaysia	16
ID	Indonesia	17
PH	Philippines	18
TW	Taiwan	19
MO	Macao	20
RU	Russia	21
SG	Singapore	22
VN	Vietnam	25
TH	Thailand	26
AR	Argentina	27
HK	Hong Kong	28
BD	Bangladesh	29

#### CDFPSK.dll error return code descriptions

Call the PTK\_GetErrorInfo function to parse it.