able

The Experts in Small Printer Solutions

Ap1400
Thermal Printer

Programmer Guide

Able Systems Limited Denton Drive, Northwich, Cheshire CW9 7TU England
Tel: +44 (0) 1606 48621 Fax: +44 (0) 1606 44903 Web: www.able-systems.com

1. Table of Contents

1. Table of Contents 2

2. Introduction 3

Notes on Printer Firmware Revisions (Including Flash) 3
Copyright Notice and Disclaimer 3

3. Modes of Operation 4

Paper Feed Button 4
Printing Method 4
Built-In Fonts 5
User Font 5
Multiple Code Page Fonts 5
Dot Addressable Graphics 5
Serial Interface 6
Data Buffer and Flow Control 6
USB Interface 7
Paper Out, Paper Low and Head Up Sensors 7

4. Control Code Tables 8

Command Description Format 8
Command Codes Received 9
Graphics Commands 10
Formatting Commands 11
Formatting Commands (Continued) 12
Initialise and Request Status Commands 13
Initialise and Request Status Commands (Continued) 14
Barcode Commands 15
Mark Sense Operation 15
Reserved Commands 15
Codes Transmitted 16

5. Settings and Configuration 17

Setting Configuration Information 17
Saving Configuration Information 19
Querying Configuration Information 20

6. Command Summaries 21

Summary of Print Format Commands and Volatile Settings 21
Summary of Real Time Status Information 21
Summary Of Non-Volatile Settings 22

7. Setting up USB functionality 23

Installing LIBUSB. 23
Installing USBPrintDll 24
Manual install of the USBPrintDll 24
Testing the LIBUSB install. 25

8. The USBPrintDLL interface. 26

2. Introduction

This Programmers Guide provides a description of the software commands supported by the Ap1400 series panel mounting printers. This includes the:

- Ap1400 : Panel Mounting Printer (4.5V to 8.5V DC input)
- Ap1400V : Panel Mounting Printer (9V to 36V DC input)

An Installers' Guide, which contains connection data and details of operation, is available from Able Systems and should be read in conjunction with this document.

Throughout this Programmers Guide the term "Printer" should be taken to refer to any of the printers in the Ap1400 range.

If individual products differ in some important aspect, then this is noted.

A wide range of software commands are supported, allowing control of printing format (e.g. width, height and spacing of text, underlining, text orientation etc.), as well as selection of modes of operation (e.g. Serial Comms settings, Paper Out, Paper Low and Head Up indications and actions etc).

Many of the software commands are emulations of the EPSON TM-Series ESCPOS codes. Since the implementation of ESCPOS varies from one EPSON printer to another, there is no universal standard. The command set for these products has primarily been based on that of the EPSON TM-T Series of thermal printers. Please refer to the acknowledgement and disclaimer.

Notes on Printer Firmware Revisions (Including Flash)

Able Systems reserves the right to modify and improve the firmware in its products at any time. Whilst every effort is made to ensure backward compatibility, no guarantee in this respect is given or implied.

These products include a flash re-programmable microcontroller. This allows firmware upgrades under customer control. A Flash Programming Utility is available to aid the user in re-flashing printers.

Also available is a Font Editor Utility, which allows the user to design and use bespoke fonts as required. User created or modified fonts may be flashed into the printer using the same Flash Programming Utility.

Refer to the factory for more information on these features.

Some host-selectable features may be retained during power loss by saving them to non-volatile memory (FLASH). The user must ensure that any changes to the printer's internal parameters are saved to flash memory, either manually or by timed auto-save. Refer to section 5.3 SAVING CONFIGURATION INFORMATION for more details.

Copyright Notice and Disclaimer

Able Systems does not assume responsibility for interchangeable functionality of other parties' command sets.

3. Modes of Operation

Please refer to the Installers' Guide for an overview of the modes of operation, including idle mode and spool mode (where applicable).

Some of the host-selectable features may be retained during power off by storage in non-volatile memory (FLASH), but the others are lost. This is clearly shown in the individual command descriptions below. The user must ensure that any changes to the printer's internal parameters are saved to non-volatile memory (FLASH), either manually or by timed auto-save.

Operational status is indicated by factory-programmable colour combinations on the front-panel LED. These can indicate input voltage, buffer mode, paper status and so on. Refer to the User Guide for details.

Paper Feed Button

No external power switch is fitted, so additional functions have been assigned to the paper feed button.

- A single press and release of the button:
- in idle or spooling mode, advances paper, also prints any partial line data from the buffer;

- "Double-clicking" the button:
- in idle mode, prints a demo/test message including the firmware version, encoded calibration data, and optionally the full character set and sample bar codes;
- in spooling mode, prints any stored data and enters idle mode.

Double-clicking means pressing and releasing twice in quick succession, in a similar manner to a PC mouse.

Some functions of the paper feed button can be invoked or disabled by the host. The button is also disabled for 0.25 seconds after each data byte is received, to prevent the user from prematurely printing partial lines (Data from the host will generally be continuous).

Printing Method

Each printer mechanism has 384 horizontally-arranged elements, printing a single dot line across the full width of the paper at once. The paper is advanced past the print head by a rubber platen driven by a stepper motor. Typical print speed is 60 mm/s at an input voltage of 7 Volts, the maximum is 80mm/s at an input voltage of 8.5 Volts. Each printed dot is nominally 0.125 mm square. The paper may only be fed through the printer in the forward direction.

It is not possible to print partial lines in isolation: if such a line remains in the buffer, it will not be printed until flushed out by: a line terminator; some following data; a programmable timeout; or by the user pressing the paper feed button, if enabled. Once a partial line has been flushed out, it may not be extended. Any following data will be printed on the next line.

Character lines may be printed in single or double width, single or double height, and underlined. These three text mode attributes may be combined at will. If single and double height characters are mixed in a line, the bottom of the characters will be aligned.

Built-In Fonts

The built-in font characters are formed according to the Font Mode:

Font Mode	Chars/Line	Character Height	Row Height
0	32	24 dots (3.00 mm)	30 dots (3.75 mm)
1	42	24 dots (3.00 mm)	30 dots (3.75 mm)
2	24	24 dots (3.00 mm)	30 dots (3.75 mm)
3	32	24 dots (3.00 mm)	24 dots (3.00 mm)
4	48	16 dots (2.00 mm)	19 dots (2.38 mm)

Font Mode 0 is the default factory setting. Character heights refer to the maximum spans of capital letters and descenders. Row heights refer to the interval from the top of one character row to the top of the next, including inter-row spacing. The row height may be adjusted if required.

In all of the built-in Font Modes, the character set is the standard IBM® character set (also known as Code Page 437) which includes graphics box-drawing characters. These box-drawing characters are extended to link up in both axes where appropriate. This character set has been modified to include the Euro symbol ('€') at position 80H (128 Decimal), in place of the usual capital C with cedilla ('Ç'). Also, certain special character substitutions are possible for particular applications (e.g.:- switching the '£' and '#' characters, or including the Nordic 'ø' & 'Ø' characters in place of the '¢' & '¥' characters). See the section on SETTING INTERNAL CONFIGURATION INFORMATION for details.

The Font Mode is normally stored in non-volatile memory (FLASH). This means that the printer can be pre-configured to operate in 32, 48, 42 or 24 Char/Line modes as applicable.

If required, Font Mode changes may be disabled. See the section on SETTING INTERNAL CONFIGURATION INFORMATION for details.

User Font

This mode gives access to a single User Font programmed into the printer's non-volatile memory using the Flash Programming Utility. The format of the characters in the User Font is defined within the font itself.

When a User Font is present, the Built-In Fonts, and any character substitutions, are disabled. If the User Font is subsequently removed, then the Built-In Fonts are re-enabled automatically.

Multiple Code Page Fonts

This kind of User Font consists of multiple code pages resident at ASCII 0x80 to 0xFF. A simple command can be issued to switch between these pages, providing access to multilingual character sets.

Dot Addressable Graphics

Several protocols for dot-addressable graphics data are supported:

- Eight (8) dot high graphics are aligned with the top of text characters. In these modes the dot patterns sent by the host may be doubled-up, tripled-up or quadrupled-up, both horizontally and vertically before printing. The dot size is nominally 0.25, 0.375 or 0.5 mm square.
- Twenty-four (24) dot high graphics are also aligned with the top of text characters, and are printed dot-for-bit as sent by the host. The nominal dot size is 0.125 mm square. Successive blocks can be vertically contiguous only if Font Mode 3 is selected.
- Single dot high graphics are also aligned with the top of text characters. Successive horizontal dot lines can be vertically contiguous if no other printing is performed between them.

Graphics may be combined with text printing as required. Large areas of solid dots are not recommended, as they may cause overheating and shorten the head life: try shading.

Serial Interface

The default serial interface format for the Ap1400 series printers is 9600 baud, 8 data bits, 1 stop bit and no parity. Other formats can be programmed into the printer at the factory or in the field, from the host. A setup program, suitable for use with a PC, is available from Able Systems to simplify this process.

Serial data is expected on Rx in RS-232C format with -12V meaning 'mark' or logical '1', and +12V meaning ‘space’ or logical '0', with reference to the common ground. The serial data output line, Tx, transmits XON/XOFF and status information to the host at the same baud rate and format as the serial data input. The hardware busy line, Busy, is true (nominally -12V) when busy.

Some host equipment use a constant space condition (+12V) to indicate a reset condition or wait state. Some battery powered host equipment present the same output signal when they go to sleep. By default the printer will interpret this condition as a repetitive receive error, and will print multiple ‘?’ characters to indicate the fault. If required, the printer may be set to ignore this condition.

Data Buffer and Flow Control

The printer has a nominal 10k byte buffer which enables data to be received while previous lines are being printed. The state of the data buffer is transmitted to the host as follows:

XOFF	(13HEX) sent when buffer fills to	3/4 full;	
XON	(11HEX) sent when buffer empties to	1/4 full;	and also after a controller reset.

The hardware busy line is set when 256 bytes of space remains; and incoming data are no longer passed to the buffer (but may be processed) when 128 bytes remain. The hardware busy line goes ready again when 384 bytes become free. Note that the buffer can become filled with non-printable codes, in which case the controller will go busy.

The printer always transmits an XON character when it is powered on and is ready to receive data. It is not necessary to select hardware or software handshaking. Both are active at all times.

USB Interface

The Ap1400 printer has a full speed USB client interface allowing the unit to be connected to a Windows based host via USB allowing simple configuration and access. There are two ways of printing from the Ap1400;

1) Spool printing. This is the way a 'standard' desktop printer works and will enable you to print in the normal way from applications such as Word, Excel or any other application that supports spool printing. Spool printing requires the installation of windows drivers and allows all applications access to the printer. To use the Ap1400 as a spool printer you will need to install the following:
   - Ap1400 USB driver

2) API interface. The API interface, takes exclusive control of the printer and allows either text or graphical printing. In addition, the full printer command set is available to the user allowing much tighter tailoring of the printed output to suit this form of printer & application combination. To use the API Interface you will need to install the following:
   - LIBUSB
   - USBPrintDLL

Please refer to section 7 for details on how to install the correct driver for your application.

Paper Out, Paper Low and Head Up Sensors

The Ap1400 series printers typically use an FTP-628MCL103 print head which has a paper out sensor and optionally a head up sensor. The reflective optical Paper Out sensor within the mechanism detects an out-of-paper condition, and/or senses black marks to register with pre-printed forms. The optional head up sensor is a mechanical switch that detects when the lid is opened.

The printer is set as standard to enter Spool mode when the sensors become active. Spool mode can be automatically exited, when the sensors become inactive again. This behaviour may be modified. See the section on SETTING INTERNAL CONFIGURATION INFORMATION for details.

The state of the sensors is reported in the STATUS BYTE.

4. Control Code Tables

General Notes: All codes from 00 to 1F which are not listed below are ignored.

An ESC, GS etc code followed by an unrecognised command is ignored, but any following parameters are interpreted as normal characters. Any ESC, GS etc sequence which is described below but which has an illegal parameter is abandoned at that point. The controller will attempt to interpret subsequent characters as normal characters.
There is no need to follow ESC or GS control code strings with a line terminator. If one is added it will be interpreted as such.

Most control codes are executed when they fall through the data buffer at printing time. Some ("real time") codes are interpreted immediately on receipt, so that the printer can respond even if the buffer is full.

The settings defined in some commands, identified by N, may be stored in non-volatile memory [FLASH]. All settings are implemented as soon as they are interpreted, except for communication settings. Newly set values for the Baud rate, parity, data and stop bit parameters are not implemented until the printer has been cycled.

Command Description Format

In the explanations that follow, commonly recognised labels have been used (e.g. LF, CR, FF, ESC); command parameters are given as equivalent ASCII characters (e.g. "J"); or as hexadecimal [hex] values (e.g. 4AH); or decimal numbers (e.g. 74 or n) which are single byte values. Dots (...) mean more parameters follow.

The command string descriptions are punctuated by commas [,] for clarity, though most versions of BASIC will use semi-colons [;]. However, these separators will not be transmitted by the host.

As an example, the command to select the underlined printing mode can be expressed in a number of ways, all of which are equivalent:

ESC, "!" ,n	(as in the command description)
27 , 33 ,128	(expressed as decimal character values)
1BH,21H,80H	(expressed as hex values)

n is a number, such that if expressed in binary, setting its most significant bit (bit 7) will select the underline mode.

This expression could be written into a BASIC program as:

```
10 PRINT#1,CHR$(27);"!Ç";                          ...using ASCII characters, or as:
10 PRINT#1,CHR$(27);CHR$(33);CHR$(128);            ...using decimal values, or as:
10 PRINT#1,CHR$(&H1B);CHR$(&H21);CHR$(&H80);       ...which is equivalent, but using hex.
```

Command Codes Received

The commands detailed below are listed in ascending ASCII code order.

To assist programmers in converting existing host software, the following codes are used:

✓	Fully supported as in EPSON ESCPOS
✓+	Altered functionality compared with ESCPOS
@	Able special code, no ESCPOS equivalent exists
N	Setting may be retained in non-volatile [FLASH] memory

ESC,"L"	**1BH,4CH**	**Enter spooling mode**	**✓+**
Data and control codes received after this command will be stored in the buffer until either a command (FF) or GS,L command to exit spooling mode is received, or the paper feed button is double-clicked by the operator (double-click operation can be disabled by the ESC,c,5.. command).			

ESC,"c","5",..	**1BH,63H,35H,n**	**Enable/Disable button functions**	**✓+N**
n is encoded so: bit 0 clear (0) Disable paper feed button bit 0 set (1) Enable paper feed button (default condition) bit 1 clear (0) Disable double-click demo mode bit 1 set (1) Enable double-click demo mode (default condition) Other bits are ignored; the double-click demo mode only functions if the paper feed button is enabled.			

ESC,"X",..	**1BH,58H,m..**	**eXtended Commands (special to Able)**	**@ N**
See the section SETTING INTERNAL CONFIGURATION INFORMATION below for format.			

ESC,"!",..	**1BH,21H,n**	**Select printing mode**	**✓+N**
The third byte n is interpreted as follows: bit 0 } Select Font Mode 0 to 4 bit 1 } (Mode 0 is factory setting) bit 2 } (Invalid Font Mode Values are Ignored) bit 4 Select(1)/Cancel(0) Double height mode bit 5 Select(1)/Cancel(0) Double width mode bit 7 Select(1)/Cancel(0) Underlined mode Bits 3 and 6 are ignored. Each of these attributes remains in force until actively cancelled. The Font Mode may be stored in non-volatile memory (FLASH) and retained when the power is cycled. Some older host equipment may require the Font Mode Select bits to be disabled; See the ESC,"X",09H,n command for details of how to do this. Font Modes (0 to 4) cannot be mixed within a single print line: a new line results when the Font Mode is changed.			

ESC,"R", …	**1BH,52H,n**	**Select Extended Page**	**@ N**
Font Format 2 and 3 are user font images containing multiple code pages. This command is used to select one of the code pages in the image.			

Graphics Commands

ESC,"*",..	1BH,2AH,m,n1,n2,d1...dk	Dot-Addressable (Bit) Graphics	✓+

Several modes of Dot Addressable Graphics are possible:
In all modes dot patterns are coded as 1= dot, 0= space, patterns are arranged in dot-columns, and the MSBit of each byte is printed at the top. The number of dot columns is given by (n1+ 256*n2).

When the third byte, m=0 or 2 (00H or 02H): **Doubled-up 8 dot graphics** [k = n1+ 256*n2]
Each 8-dot vertical column is encoded as a single byte. Each dot is doubled in both axes to 0.25 mm square. Successive rows of 8-dot graphics cannot be printed contiguously in the vertical direction unless using a 16 dot high User Font.

When the third byte, m=3 (03H): **Tripled-up 8 dot graphics** [k = n1+ 256*n2]
Each 8-dot vertical column is encoded as a single byte. Each dot is tripled in both axes to 0.375 mm square. Successive rows of 8-dot graphics can be printed contiguously in the vertical direction only in Font Mode 3 or a User Font which is 24 dots high.

When the third byte, m=4 (04H): **Quadrupled-up 8 dot graphics** [k = n1+ 256*n2]
Each 8-dot vertical column is encoded as a single byte. Each dot is quadrupled in both axes to 0.5 mm square. Successive rows of 8-dot graphics can be printed contiguously in the vertical direction only in Font Modes 0, 1 or 2 or a User Font which is 32 dots high.

When the third byte, m=32 (20H): **Plain 24 dot graphics** [k = 3 x (n1+ 256*n2)]
Each 24-dot vertical column is encoded as three successive bytes. Bytes d1, d4, d7 etc are at the top of the 24-dot vertical columns. The printed graphic pattern is therefore represented by the byte numbers as follows:

d1 d4 d7 ... d(k-2)
d2 d5 d8 ... d(k-1)
d3 d6 d9 ... dk

Each dot is the mechanism dot size of 0.125 mm square. Significant features should be coded using at least 2 dots together. Successive rows of 24-dot graphics can be printed contiguously in the vertical direction only in Font Mode 3 or a User Font which is 24 dots high.

When the third byte, m=8 (08H): **Single Dot Line graphics** [k = n1+ 256*n2]
Data bytes are printed horizontally in order from left to right across the page, with the Most Significant Bit to the left. Each byte encodes 8 horizontal dots. Each dot is the mechanism dot size of 0.125 mm square. Successive horizontal dot rows can be printed contiguously in any Font Mode only if no other printing occurs between them.

If the print mechanism is allowed to stop between successive contiguous graphics patterns, slight discontinuities may occur due to mechanical hysteresis. Consider using spool mode to prevent this.

Formatting Commands

ESC,"-"..	1BH,2DH,n	Turn underline on/off	✓+
If n=0: Underlining is turned off; otherwise is turned on.			

ESC,"2"	1BH,32H	Set default Row Height	✓
Set default row height. ie:-30 (0.125mm) dots in Font Mode 0,1 and 2; 24 dots in Font Mode 3; and 16 dots in Font Mode 4.			

ESC,"3",..	1BH,33H,n	Set Row Height	✓
n is the row height. Allowed range: 16 to 99. Row height refers to the interval from the top of one character row to the top of the next, including inter-row spacing. This setting is retained until cleared, or the Font Mode is changed or the printer is powered off.			

ESC,"D",..	1BH,44H,d1..dk,00	Set horizontal tab positions	✓+
Up to 6 positions may be recorded, and replace the default settings. If more are given, the first 6 are accepted and normal processing resumes from that point. If less than 6 are given then the command should be terminated with a NUL(00H) character. Note that tab settings take variable character spacing into account. Setting is retained until cleared or the printer is powered off.			

ESC,"J",..	1BH,4AH,n	Print and feed extra paper	✓+
n is the count of notional 1/20th print lines to be fed. This command terminates the current line; n is divided by 20 (remainder discarded) and the quotient used as a count of additional blank single-height character lines.			

ESC,"$",..	1BH,24H,n1,n2	Set absolute print position	✓+
Position = (n1 + 256*n2) dots from start of line. If printing is currently to the left of this position, blank space is inserted up to it. If it is to the right, then over-printing can occur, allowing special character combinations to be formed. Any overflow is cut off at end of line.			

ESC,"\",..	1BH,5CH,n1,n2	Set relative position	✓
Position = (n1+ 256*n2) dots from current position in text lines. Blank space is inserted up to the selected dot position. Any overflow is cut off at end of line.			

ESC,"d",..	1BH,64H,n	Print and feed n lines	✓
Terminates the current line and feeds n blank print lines			

ESC,"{",..	1BH,7B,n	Set/Cancel inverted character printing	✓
n is encoded so: bit 0 cleared (0) Normal (upright) text bit 0 set (1) Inverted text Normal and inverted text cannot be mixed in a single line. This setting is temporary (the default setting can be redefined by the ESC,X command and stored in non-volatile memory [FLASH]).			

Formatting Commands (Continued)

HTAB	09H	Horizontal Tab	✓
Default positions: 8,16,24,32,40; 6 positions available; programmed using ESC,D command. Ignored if off the end of a line. The first HTAB does not move from a HTAB position to the next: e.g. 1234567890123456 123456 T (1 Tab before "T") 1234567T (1 Tab before "T") 1234567 T (2 Tabs before "T") 12345678 T (1 Tab before "T")			

LF	0AH	Line Feed	✓
Works in an either/or way with CR: CR/LF pairs are treated as a single line terminator. Line terminators immediately following full lines are ignored.			

FF	0CH	(Real time) Form Feed	✓+
Exit spooling mode and print buffer contents. Ignored if the printer is not already in spooling mode or an error condition exists.			

CR	0DH	Carriage Return	@
Line terminator: works in an either/or way with LF. CR/LF pairs are treated as a single line terminator			

ESC," ",..	1BH,20H,n	Set right-of-character spacing	✓
n is the number of additional (note!) dot spaces placed to the right of each character printed. Default= 0; maximum value= 31. Setting is retained until cleared or the printer is powered off.			

Initialise and Request Status Commands

ESC,"@"	1BH,40H	Initialise printer	✓
Clears print parameters to power-on default, ie normal width and height, no underline, no extra space, and default tabs. Does not affect the inverted mode. Not real-time, executed in data sequence (unlike CAN).			

CAN	18H	(Real time) Abort printing and initialise	@
Same result as ESC,'@' but executed immediately when received. A line in progress is allowed to finish, but printing then continues from data received after the CAN code. Ignored if embedded in an ESC, GS or graphics sequence.			

GS,ENQ	1DH,05H	(Real time) request for printer status	✓+
This command is executed and the STATUS byte is transmitted immediately on receipt, even if the printer is busy. See the section CODES TRANSMITTED for format.			

GS,"I",..	1DH,49H,n	Request to transmit information	✓+
See the section QUERYING INTERNAL CONFIGURATION INFORMATION below for format.			

GS,"L"	1DH,4CH	Exit Spool Mode and Transmit confirmation data	@

If spool mode is entered by ESC, L command and this command is used to exit from spool mode, confirmation of the number of bytes sent whilst in spool mode will be returned. The sequence on exiting spool mode is;
(i) Transmit initial confirmation string with data packet information
(ii) Exit Spool Mode
(iii) Print all spooled data from buffer
(iv) Transmit final confirmation string

When the **GS,'L'** command is received a 4 byte confirmation string is transmitted by the printer as follows:

STX ByteCountLo ByteCountHi CheckDigit
[02h] [06h] [00h] [4Fh]

ByteCount is transmitted as two consecutive bytes such that (ByteCountLo + (ByteCountHi x 256)) gives the actual ByteCount value. In this case, there are 6 data bytes between the ESC,'L' and GS,'L' commands, and the result of a cumulative XOR of each of these 6 bytes yields 4Fh as the CheckDigit.

The printer then exits from Spool mode and processes the data which has been held in the buffer in the normal way. Once the data buffer is completely empty and the mechanism is no longer running (as normally reported by bits 1 and 2 of the STATUS byte), then the packet of data is judged to have been processed completely. At this point a further confirmation string is transmitted as follows:

ETX ByteCountLo ByteCountHi CheckDigit
[03h] [06h] [00h] [4Fh]

Note that the first byte is STX for the first string and ETX for the second string.
The final 3 bytes are identical in both strings.

Initialise and Request Status Commands (Continued)

GS,"a",..	1DH,61H,n	Enable/Disable automatic status	✓+N
Defines the conditions under which the STATUS byte is transmitted without explicit request from the host. A bit set in 'n' causes the STATUS to be sent whenever the corresponding bit in the STATUS byte changes state (see CODES TRANSMITTED for format). A value of n=00H disables automatic reporting, and is the default condition.			

ESC,"u",..	1BH,75H,n	Transmit peripheral device status	✓+
Transmit STATUS byte when decoded. n is discarded; this command is exactly the same as ESC,v This command is in the buffer when the response is sent, so the buffer will not be reported as empty. See the section CODES TRANSMITTED below for format.			

ESC,"v"	1BH,76H	Transmit printer status	✓+
Transmit STATUS byte when decoded. This command is in the buffer when the response is sent, so the buffer will not be reported as empty. See the section CODES for format.			

Barcode Commands

GS,"H",n	1DH,48H,n	Select automatic text in barcode	✓

n is encoded so:

bit 0 cleared	(0)	No barcode text above barcode symbol (default)
bit 0 set	(1)	Print barcode text above barcode symbol
bit 1 cleared	(0)	No barcode text below barcode symbol (default)
bit 1 set	(1)	Print barcode text below barcode symbol

This setting is retained and used for all subsequent barcodes, but is cleared to the default when the printer is powered off.

GS,"h",n	1DH,68H,n	Select height of barcode	✓+

Height of barcode = (n x 0.125mm) n is valid in the range 1 <= n <= 150.
Default value 100; a zero value is ignored; a value of 150 is used in place of values greater than 150. This setting is retained and used for all subsequent barcodes, but is cleared to the default when the printer is powered off.

GS,"k",..	1DH,6BH,m,d1..dk,t	Print barcode using data provided	✓

N.B. This command must always be terminated with value “t” shown below.
The number and type of bytes of data varies with barcode type "m".

Valid m values:		Terminator (t)
0: UPC-A	Numeric only ASCII data: supply 11 digits	00H
1: UPC-E	Numeric only ASCII data: supply 6 digits	00H
2: EAN-13	Numeric only ASCII data: supply 12 digits	00H
3: EAN-8	Numeric only ASCII data: supply 7 digits	00H
4: Code 39	Alphanumeric ASCII data: variable length (Max 22)	00H
5: Int2 of 5	Numeric only ASCII data: variable length (Max 23)	00H
6: Code 128A	Alphanumeric ASCII data 0x00 to 0x5F: variable length (Max 14)	FFH
7: Code 128B	Alphanumeric ASCII data 0x20 to 0x7F: variable length (Max 14)	FFH
8: Code 128C	Numeric only ASCII data 30H to 39H: variable length (Max 14 bytes)	FFH
9: Code 93	Alphanumeric ASCII data 0x00 to 0x7F: variable length (Max 16)	FFH

Note that the user should verify that a given barcode will fit on the paper, especially when using the variable length barcodes. Barcodes may not be mixed with normal text.

GS,"w",n	1DH,77H,n	Select width of barcode	✓

Width of barcode element (narrow bar) = (n x 0.125mm)
n is valid in the range 2 <= n <= 4. Other values are ignored; Default value is 3.
This setting is retained and used for all subsequent barcodes, but is cleared to the default when the printer is powered off. Note that the user should verify that a given barcode will fit on the paper, especially when using the variable length barcodes.

Mark Sense Operation

Please refer to the factory for more detailed information before using the Mark Sense feature.

Reserved Commands

In general, commands which are not implemented and described above should not be sent to the printer, as the outcome may not be as expected. However, some additional EPSON ESCPOS commands are recognised, and an attempt is made to decode them, so that following commands will remain in synchronism and correctly interpreted. Please refer to Able Systems in case of difficulty.

Codes Transmitted

The printer automatically transmits XON and XOFF bytes when the buffer status changes, as follows:

XON	**11H**	**Start transmission**	
Meaning: The buffer is ready to receive data. Transmitted after a reset, or when the data buffer empties to only 1/4 full.			

XOFF	**13H**	**Stop transmission**	
Meaning: The buffer is not ready to receive data. Transmitted when the data buffer becomes 3/4 full.			

In addition the printer can transmit a special STATUS byte, either on request, or optionally when individual status conditions change state:
The commands ESC,"u",n and ESC,"v" request the STATUS byte be transmitted at the time that the command is interpreted. (N.B. The buffer is never reported as empty when these commands are used, as it contains at least this command at the time of transmission).
The command GS,ENQ requests the STATUS byte be transmitted immediately on receipt of the command, even if the buffer contains large amounts of data yet to be interpreted and/or printed.
The GS,"a",n command specifies individual bits of the STATUS byte which should be monitored, and result in the STATUS byte being automatically transmitted if a change of state is detected.

[STATUS]		**Status Report**	
The STATUS byte is encoded bitwise: *bit 0 Head Up Sensor Active? [0= head OK 1= head up] bit 1 Mechanism running [0= stopped 1= running] bit 2 Data buffer completely empty? [0= not empty 1= empty] bit 3 Paper Out Sensor Active? [0= paper OK 1= paper out] bit 4 Reserved [ ] bit 5 Spooling mode? [0= normal 1= spooling] bit 6 Error [0= no error 1= error present] bit 7 Always Set [Always 1]			

* Note Head Up sensor optional, status only when fitted

The command GS,"I",n is also a request for information to be transmitted at the time that the command is interpreted (i.e. not in 'real-time'). For details of the valid values for n, and the resultant transmitted information see the QUERYING INTERNAL CONFIGURATION INFORMATION section below.

If an error condition exists, then the usual single STATUS byte is followed by a second error identification byte. This second byte is only sent if bit6 of the STATUS byte is set. The values of the error ID byte are as follows:

Value	Error type
80H	Mechanism Voltage (Vmech) above upper limit
7FH	Mechanism Voltage (Vmech) below lower limit
40H	Mechanism Head Temperature above upper limit

Other values are either not defined or represent internal controller hardware errors.

5. Settings and Configuration

The printer maintains a large number of internal settings and configuration information which is not derived from ESCPOS control codes, and is unique to Able Systems printers.
In general, the ESC,"X",m,... command is used to set these values, and the GS,"I",m command is used to query them. The following table details all valid combinations of these commands:
(N.B. See the section SAVING CONFIGURATION INFORMATION).

m	ESC,"X",m,...	
4	eg "9600,N,8,1",CR	Set Baud
9	n1	Set Internal Defaults
15		
18	n1..n18	Set LED patterns
19	n1	Set MT102FLAGS
20	n1,n2	Set MarkFeed / EjectFeed
23	n1	Set AUXFLAGS
33	n1	Set MaxDotsAtOnce
42	n1	Set EjectOffset
48		Save all settings to Flash
52	n1,n2	Set Auto-Save Period
66	n1	Set print darkness
110		Produce test print

m	GS,"I",m	
3	n1,n2	Report Firmware Version Number
4	eg " 9600,N,8,1",CR	Report Baud Setting
6	eg "SerialNo",CR	Report Unit Serial Number
9	n1,n2,n3	Report Internal Defaults
15	n1,n2,n3	Report Voltage/Temp Data
18	n1..n18	Report LED patterns
19	n1	Report MT102FLAGS
20	n1,n2	Report MarkFeed / EjectFeed
23	n1	Report AUXFLAGS
33	n1	Report MaxDotsAtOnce
42	n1	Report EjectOffset
52	n1,n2	Report Auto-Save Period
66	n1	Report print darkness

Setting Configuration Information

N.B. This section contains details of how to modify some of the basic operating functions of the printer. Do NOT attempt to modify any parameter unless you fully understand the potential consequences.

The command ESC,"X",m,... may be used to set various internal configuration values according to the value of m as defined below.

m=4 Select Baud rate, parity, word length and number of stop bits. (Default: “9600,N,8,1”)
Command parameters must have format: e.g."19200,E,8,2", so:

4 or 5 number characters,	[the required Baud rate]
1 comma character	
1 "N/n/E/e/O/o" letter character,	[the required Parity]
1 comma character	
1 "7/8" number character,	[the required Data bits]
1 comma character	
1 "1/2" number character	[the required Stop bits]

e.g. To select 19200,E,8,2 the following 14 byte command should be sent:
ESC,"X",04H,"19200,E,8,2"
or as hex codes: 1BH,58H,04H,31H,39H,32H,30H,30H,2CH,45H,2CH,37H,2CH,32H

The Ap1400 accepts the following RS232 baud rates;
1200, 2400, 4800, 9600, 38400, 57600, 115200

It accepts 8 data bits followed by either 1 or 2 stop bits and can use Odd, Even or No parity.

The setting must saved to non-volatile memory (FLASH), and does not come into effect until after the next printer reset.

m=9 Set Internal Default Values.

The command ESC,"X",09H,n sets the internal default values according to individual bits in the value of n. In each case, a bit set(1) means the following statement is true:

Bit 0 Default to inverted print mode
Bit 1 Disable FontMode bits of ESC,"!",n command
Bit 2 Suppress "?" printing on RX error
Bit 3 [Unassigned]
Bit 4 Enable Split dots printing (See MaxDotsAtOnce [m=33] for details)
Bit 5 [Unassigned]
Bit 6 [Unassigned]
Bit 7 [Unassigned]

Note: the setting of bit 0 does not come into effect until after the next printer reset.

m=18 Set LED patterns.
The standard pattern set displayed by the printer has been worked out with a great deal of care to provide the maximum useful information while remaining clear and unambiguous. However they may be modified if required. Please refer to Able for details.

m=19 Set MT102FLAGS value.
The command ESC,"X",13H,n sets the internal values according to individual bits in the value of n. In each case, a bit set(1) means the following statement is true:

Bit 0 [Must be set to enable paper out optical sensor]
Bit 1 Enable Mark Sense Operation
Bit 2 Sense Mark at Black->White or White->Black Edge
Bit 3 [Unassigned]
Bit 4 [Unassigned]
Bit 5 Led Pattern Dependent on Paper Out Sensor
Bit 6 [Must be set to enable head up sensor] (when fitted - optional)
Bit 7 Led Pattern Dependent on Head Up Sensor (when fitted - optional)

The standard setting for this value is: E1H for the Ap1400.

m=20 Set MarkFeed and EjectFeed values.
The command ESC,"X",14H,n1,n2 sets the values of MarkFeed and EjectFeed. These parameters are used in Mark Sense operation. See MARK SENSE OPERATION section for details.

m=23 Set AUXFLAGS value.
The command ESC,"X",17H,n sets the internal values according to individual bits in the value of n. In each case, a bit set(1) means the following statement is true:

Bit 0 Select Short Demo Print
Bit 1 Select swapping '#' (23H) and '£' (9CH) chars
Bit 2 Select switching in Nordic 'ø' & 'Ø' chars in place of '¢' (9BH) & '¥' (9DH)
Bit 3 Select original CP437 character 'Ç' in place of new standard '€' (80H)
Bit 4 Select Busy to go active when Paper Out condition seen
Bit 5 [Unassigned]
Bit 6 [Unassigned]
Bit 7 [Unassigned]

m=33 Set MaxDotsAtOnce value.
The command ESC,"X",21H,n sets the value of MaxDotsAtOnce. This value controls the operation of the printing process to limit the maximum instantaneous current that may be drawn from the power supply.
Valid values are in the range 01H to 30H. This value equates to the number of dots that may be energised divided by eight. 01H represents the lowest current draw (i.e. only 8 dots may be energised at a time), and 30H represents maximum peak current draw (i.e. all dots are allowed to be energised simultaneously).
The printer automatically slows the paper feeding to allow all the dots to be printed. Therefore, reducing the value of MaxDotsAtOnce also reduces the overall print speed.

The default value for the Ap1400 is 08H. Refer to the factory if more information is required.

N.B. This value is disabled, and no splitting occurs, if bit4 of Internal Defaults (m=9) is clear.

m=42 Set EjectOffset value.
The command ESC,"X",2AH,n sets the value of EjectOffset. This parameter is used in Mark Sense operation. See MARK SENSE OPERATION section for details.

m=48 Save all settings to Flash.
The command ESC,"X",30H forces an immediate save of all the configuration settings to flash. This takes less than a second, but the printer goes busy during the process.

m=52 Set Auto-Save-to-Flash Period.
The command ESC,"X",34H,n1,n2 has the same format as the (m=11) command. By default the Auto-Save-to-Flash function is disabled (See below for more details).

m=66 Set Print Darkness.
The command ESC,"X",42H,n sets the value of print darkness. This value controls the operation of the printing process to suit different paper types.
Valid values are in the range 55H to 90H. Increasing the value causes the printed output to become darker. 55H suits standard thermal paper. 90H suits Linerless paper.

Saving Configuration Information

All configuration information must be saved to Flash if it is to be non-volatile. Ideally, this should be performed manually by sending the command ESC,"X",30H (see above).

Alternatively, the printer maintains a special timer which will perform an automatic save of the configuration information after a pre-set period of inactivity. This automatic timed save is normally disabled, but may be enabled by sending the ESC,"X",34H,n1,n2 command.

Note that during the saving procedure the printer effectively performs a full system reset. Therefore, it is advised that the save command be issued in isolation (i.e. not embedded in a stream of printable data).

Querying Configuration Information

In general, for each ESC,"X",m,.. command, the corresponding GS,"I",m command may be used to query the current value. In some cases the information reported is different from the value(s) set.

m=3 Report Firmware Version.
In response to the command GS,"I",03H the printer transmits 2 packed BCD bytes. The first byte contains the major and minor version, and the second byte contains the revision number. eg:- 12H and 34H means Version 1.2.34

m=4 Report Baud rate, parity, word length and number of stop bits.
In response to the command GS,"I",04H the printer transmits a string of ASCII characters in the same format as used in the ESC,"X",04H,... command. eg:- "19200,E,7,2"

m=6 Report Unit Serial Number.
In response to the command GS,"I",06H the printer transmits a string of up to 10 ASCII characters terminated by a CR character. eg:- "123456",CR
This is the Unit Serial Number as set by the factory during printer test/setup.

m=9 Report Internal Defaults.
In response to the command GS,"I",09H the printer transmits 3 bytes. The first byte contains the current values of the flags set using the ESC,"X",09H,n command. The following 2 bytes are for Able diagnostics only.

m=15 Report Real-Time Voltage and Temperature Values.
In response to the command GS,"I",0FH the printer transmits 2 bytes.
n1 = Supply Voltage x 10 [eg:- 43H means 6.7V]
n2 = Printhead temperature in ºC [eg:- 14H means 20ºC]

m=18 Report LED patterns.
Please refer to Able for details if required.

m=19 Report MT102FLAGS value.
m=20 Report MarkFeed and EjectFeed values.
m=23 Report AUXFLAGS value.
m=33 Report MaxDotsAtOnce value.
m=42 Report EjectOffset value
m=52 Report Auto-Save Period value
m=66 Report Print Darkness

In response to each of the above commands (in the form GS,"I",m), the printer transmits the current values of the parameters set using the corresponding ESC,"X",m,… command.

6. Command Summaries

Summary of Print Format Commands and Volatile Settings

Function	Command
Clear Print Settings and Buffer (Select single width, single height, no underlining, no extra inter-character space, and default HTAB positions)	ESC,”@” CAN (real-time)
Font Mode, Double Height, Double Width, Underlining	ESC,"!",n
Underlining only	ESC,"-",n
Inverted Print Mode	ESC,"{",n
Print positioning	HTAB ESC,"$",n1,n2 ESC,"\",n1,n2
Line termination (and Paper Feed)	LF CR ESC,"J",n ESC,"d",n
Dot Addressable (Bit) Graphics	ESC,"*",..
Entering and exiting Spooling Mode	ESC,"L" FF
Extra inter-character and inter-line spacing	ESC,” “,n ESC,”2” ESC,”3”,n
Set HTAB positions	ESC,"D",..
Request Printer STATUS BYTE	ESC,"u",n ESC,"v" GS,ENQ (real-time)
Select Barcode Size and Format	GS,"H",n GS,"h",n GS,"w",n
Print Barcode	GS,"k",..

Summary of Real Time Status Information

Information	How reported		
Mechanism Status	GS,ENQ	STATUS BYTE bit 1 (Set = Mech is Active)	+
Data Buffer Status	GS,ENQ	STATUS BYTE bit 2 (Set = Buffer is Empty)	+
Paper Out Status	GS,ENQ	STATUS BYTE bit 3 (Set = Paper is Out)	+
Spool Mode Status	GS,ENQ	STATUS BYTE bit 5 (Set = In Spool Mode)	+
Real Time Input Voltage	GS,”I”,15	Also In Self Test	+

Summary Of Non-Volatile Settings

Setting	How set	How reported	
Firmware Version No.		GS,”I”,3	Also In Self Test
Unit Serial No.		GS,”I”,6	Also In Self Test
Save all configuration to Flash	ESC,"X",48		
Auto-Save config to Flash (Period)	ESC,"X",52..	GS,”I”,52	
Baud Rate, Parity etc	ESC,”X”,4..	GS,”I”,4	Also In Self Test
LED patterns & LED Flash Period	ESC,”X”,18..	GS,”I”,18	
Default to Inverted Print	ESC,”X”,9..	GS,”I”,9	Also Coded In Self Test
Short Demo Message	ESC,”X”,23..		Coded In Self Test
Suppress "?" on Rx Error	ESC,”X”,9..	GS,”I”,9	Also Coded In Self Test
Split dots Printing (enable)	ESC,”X”,9..	GS,”I”,9	Also Coded In Self Test
Split dots Printing (MaxDotsAtOnce)	ESC,”X”,33..	GS,”I”,33	Also Coded In Self Test
Disable Feed Switch	ESC,”c5”..		Coded In Self Test
Disable Demo Mode	ESC,”c5”..		Coded In Self Test
Swap '#' (23H) <-> '£' (9CH) chars	ESC,"X",23,n	GS,"I",23	
Use 'ø' & 'Ø' <-> '¢' (9BH) & '¥' (9DH)	ESC,"X",23,n	GS,"I",23	
Use 'Ç' <-> '€' (80H)	ESC,"X",23,n	GS,"I",23	
Enable Black Mark Operation	ESC,”X”,19..	GS,”I”,19	Also Coded In Self Test
Black Mark parameters (detail)	ESC,”X”,20..	GS,”I”,20	
Black Mark parameters (more detail)	ESC,”X”,42..	GS,”I”,42	
Paper Out -> LED Pattern	ESC,”X”,19..	GS,”I”,19	Also Coded In Self Test
Head Up -> LED Pattern	ESC,”X”,19..	GS,”I”,19	Also Coded In Self Test
Select Busy on Paper Out condition	ESC,"X",23,n	GS,"I",23	
Font Mode	ESC,"!",n		Also Coded In Self Test
Disable FontMode changes	ESC,”X”,9..	GS,”I”,9	Also Coded In Self Test
AUTO_STATUS settings	GS,”a”..		

A simple Windows setup program is available from Able on request allowing a printer to be configured via a suitable RS232 (or alternatively USB to RS232 converter) cable.

Please refer to Able Systems or visit our website at **www.able-systems.com** for more information.

7. Setting up USB functionality

In order to take advantage of the USB functionality of the Ap1400, drivers must be installed on the Windows based machine you wish to use the printer with. First you must install LIBUSB, and then you should install the USBPrintDLL interface in order to take advantage of Able Systems' pre-configured commands.

Installing LIBUSB.

There are two ways to obtain the files to install LIBUSB

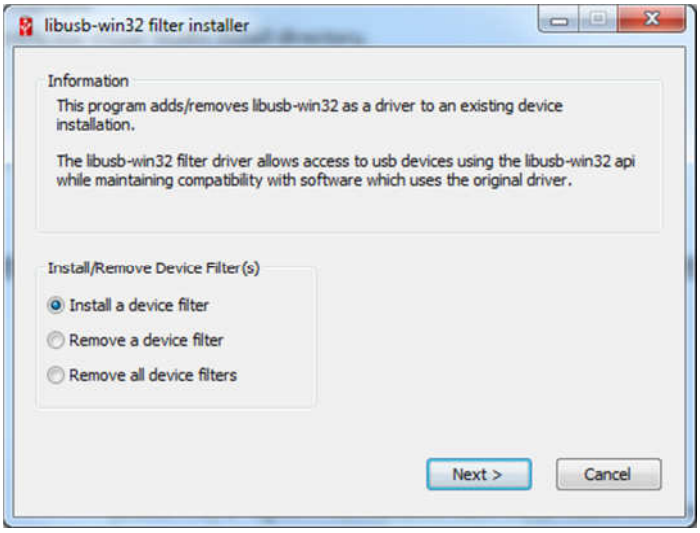
1) Contact Able Systems with details of your company, application and operating system at support@able-systems.com
2) Download LIBUSB from http://sourceforge.net/projects/libusb-win32/

When using files provided by Able Systems;
LIBUSB for Windows is supplied in a zip file containing all of the files needed to install the library on Win32 (32bit) and x64 (64bit) architecture machines. Contained within the zip file is a document detailing the install process. Please read and follow these instructions to install LIBUSB.

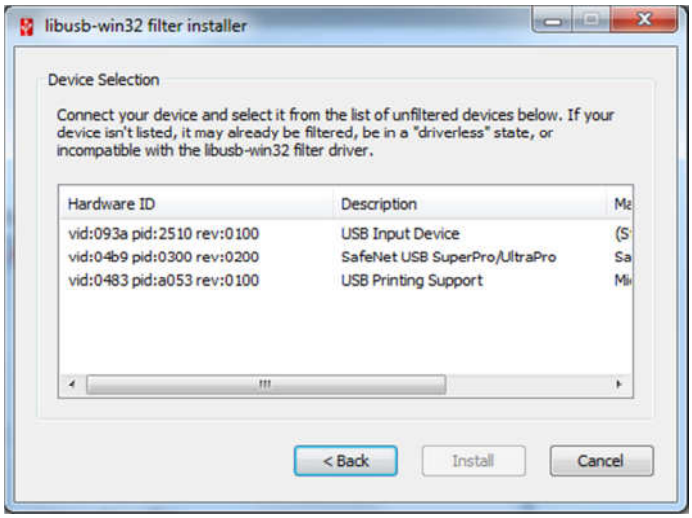
When using files obtained from http://sourceforge.net/projects/libusb-win32/;
Download the latest LIBUSB from the website. Extract 'libusb-win32-bin-1.2.6.0.zip' and navigate to the folder titled 'bin'. Once here, select your operating system and run 'install-filter-win.exe'.

To install the driver for the Ap1400 printer proceed as follows.

1. Ensure that the Ap1400 printer is externally powered and is switched on.
2. Connect the Ap1400 to the PC using the supplied USB cable.
3. Windows will install the base driver for the Ap1400 automatically and place it in the USB Printing support category.
4. The following dialog will be displayed

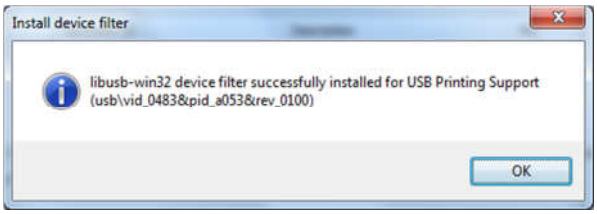


5. Select 'Install a device filter' and click 'next'. The utility will display the 'Device selection' dialog



6. Select the device with a VID:0483 and a PID:A053. and click ‘install’ This device will have a description of USB Printing Support.

7. When complete, the utility will display the completion message. Click Ok and then ‘cancel’ to exit the utility. Driver install is now complete.



8. Once the driver has been installed, disconnect the USB lead from the Ap1400, wait for 5 seconds then reconnect.

Installing USBPrintDll

The USB device interface is accessed via the interface provided by USBPrintDll. Access to this DLL must be included in the users’ application by inclusion of the header ‘USBPrintDll.h’ and by linking to the USBPrintDll.lib file. The installation of this can be done manually as follows.
Visual Studio Example.

1.Identify the Visual Studio install directory.
For 32 bit installs this will normally be
[BootDrive]:\Program Files\Microsoft Visual Studio (version).
For 64 bit installs this will normally be
[BootDrive]:\Program Files X86\Microsoft Visual Studio (version).
2. Identify the language\compiler directory (for C\C++ this would be \VC).
3. Identify the include & library directories (for C\C++ these would be \include & \lib respectively.
4. Copy the supplied USBPrintDll.lib file into the library directory and the USBPrintDll.h file into the include directory.

Manual install of the USBPrintDll

Manual installation of the USBPrintDll.dll file is simply a matter of copying it into the correct directory for the operating system. (Alternatively, if only the users’ application requires access, it may be stored in the application runtime/execution directory).

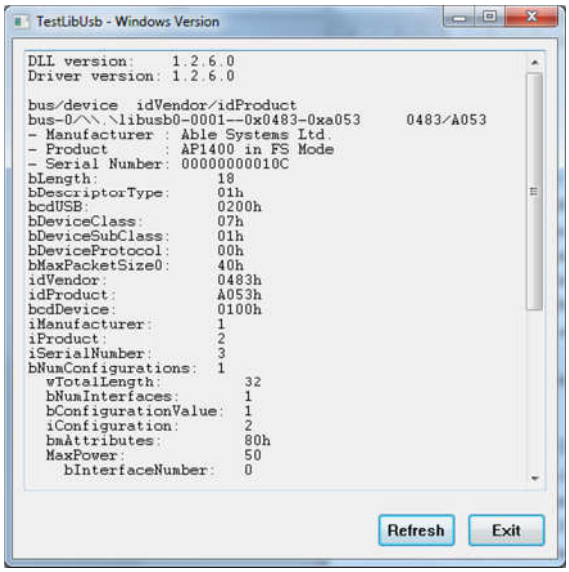
For 32 bit operating systems the USBPrintDll should be copied into the Windows\System32 directory.

For 64 bit operating systems the USBPrintDll64.dll should be renamed to USBPrintDll.dll, then copied into the Windows\System32 & Windows\SysWOW64 directories

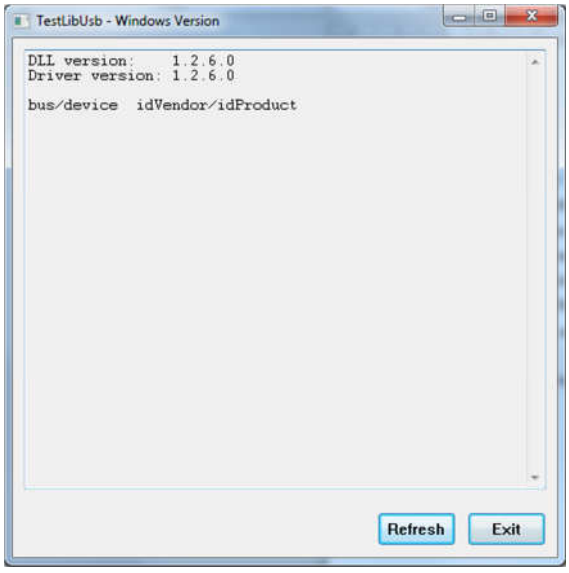
Testing the LIBUSB install.

Following the installation process, it is worth checking that all has gone well. This can be achieved by running the testlibusb-win.exe program. This utility will be found in the relevant bin directory of the LIBUSB install. (See installing LIBUSB).

This utility is a simple single dialog utility which, will show the status of all devices for which LIBUSB filter drivers have been installed. The dialog displayed dynamically refreshes allowing the connection and disconnection of devices to be tested. Connect the printer to the USB and run the utility.
When run, a dialog similar to the one shown will be displayed.



This dialog shows the bus enumeration for the Ap1400 printer (VID 0483 PID A053)
(Note: It may be necessary to scroll down to find the printer if more than one filter driver is installed.)
When the printer is disconnected then the dialog display will change and the entry for the Ap1400 will be removed (see below).



Reconnecting the printer will cause the details to re-display after a short pause.

8. The USBPrintDLL interface.

The following sections describe the functions provided by USBPrintDLL, the arguments required, and the return values.

Name	Function	Page
GetDllVersion	Get the USBPrintDll full version information	25
DllInitialise	Initialise the USBPrint subsystem and LIBUSB drivers	26
EnumPrinters	Enumerates all the available USB printers with an ABLE vendor ID	27
GetEnumPrinterCount	Gets the number of printers currently in an enumerated state	28
ClearEnumList	Clears all enumerations of printer on the USB	29
GetVendor	Gets the vendor ID of an enumerated device	30
GetProduct	Gets the product ID of an enumerated device	31
GetName	Gets the enumerated name of an enumerated device	32
OpenDevice	Opens a device and prepares it to send/receive data	33
CloseDevice	Closes and releases a device and associated handle	34
SendData	Send data to a device	35
ReadData	Read data transmitted from a device	36
SendCtrl	Send a control request or control data to a device	37
SendStatusReq	Get the current device status byte	39
KrnlSendData	Send data without flow control	40

GetDllVersion

Get the USBPrintDll full version information	
Prototype	SUSBPrintVersion WINAPIGetDllVersion(void);
Description	This function returns the full version information of the USBPrintDll. This function has no activity on the USB bus and is provided for tracing purposes. An application should check this version number before accessing the USB bus to ensure correct alignment between builds.
Arguments	None

Returns	Returns a populated SUSBPrintVersion structure describing the major & minor version numbers and the build number. If an error occurs then the system error code is set. Use GetLastError() to retrieve this.
	The SUSBPrintVersion structure is defined as;
	`//`
	`//SUSBPrintVersion`
	`typedef struct SUSBPrintVersion`
	`{`
	`UINT m_uiMajor1;`
	`UINT m_uiMajor2;`
	`UINT m_uiMin1;`
	`UINT m_uiMin2;`
	`UINT m_uiBuild;`
	`} SUSBPrintVersion;`
	`//`
	`//Define pointers to this type`
	`typedef SUSBPrintVersion *PSUSBPrintVersion;`
	`typedef PSUSBPrintVersion *LPSUSBPrintVersion;`

Example	SUSBPrintVersion sVersion; CString csDllVersion; sVersion=GetDllVersion(); csDllVersion.Format(_T("Version [V%d.%d.%d.%d Build %03d.]."), sVersion.m_uiMajor1, sVersion.m_uiMajor2, sVersion.m_uiMin1, sVersion.m_uiMin2, sVersion.m_uiBuild); TRACE(_T("USBPrintDll-%s"),csDllVersion);

DllInitialise

Initialise the USBPrint subsystem and LIBUSB drivers
Prototype int WINAPIDllInitialise(void);
Description This function prepares the LIBUSB subsystem and the USB bus for access. It loads all necessary LIBUSB dll's and initialises them. This function must be called once only, prior to any other access of USBPrintDll functionality (except GetDllVersion). In a foundation class application this call is best placed in the InitInstance function of the CWinApp class.
Arguments None
Returns Integer value (0 or 1) to indicate success or failure, where 0 is a failure. If a failure occurs then DllInitialise will display a dialog explaining the problem. In addition, the system error code is set which can be retrieved by calling GetLastError()

Example
//Initialise the Support DLL
if(DllInitialise())
{
//Declare & show main dialog
CUSBPrintTestDlg cUSBTestDlg(this);
m_pMainWnd=&cUSBTestDlg;
INT_PTR iResponse=cUSBTestDlg.DoModal();
//We are not bothered about the return
}
else
{
CString csError;
csError.Format(_T("USBPrintDll failed to initialise. R/C=%d."),
GetLastError());
AfxMessageBox(csError,MB_ICONSTOP|MB_OK);
}

EnumPrinters

Enumerates all the available USB printers with an ABLE vendor ID
Prototype `int WINAPI EnumPrinters(void);`
Description Enumerates all available printers present on the USB with a valid 'Able Systems' vendor ID. All control structures for each device found, are built and initialised from the printer supplied control information blocks. This function must be called prior to accessing an individual, or group of printers. If the function is called more than once, then each call should be preceded with a call to ClearEnumList.
Arguments None
Returns Integer value representing the number of printers successfully enumerated. If an error occurs then the number returned will be 0 and the system error code will be set. If the return indicates 0 printers and the system error code is ERROR_SUCCESS, then no error has occurred, there are just no printers available. The system error code can be retrieved by calling GetLastError()

Example
`//Clear existing list`
`ClearEnumList();`
`//Enumerate`
`int m_iEnumeratedPrinters=EnumPrinters();`
`//Display results`
`if(GetLastError()==ERROR_SUCCESS)`
`TRACE(_T("Success–Printer count=%d.\n"),m_iEnumeratedPrinters);`
`else`
`TRACE(_T("FAILED-Return code=%d.\n"),GetLastError());`

GetEnumPrinterCount

Gets the number of printers currently in an enumerated state	
Prototype	`int WINAPI GetEnumPrinterCount(void);`
Description	Returns the number of printers currently in an enumerated state or 0 for none. This function does not guarantee that the number of printers enumerated are still active. A call to EnumPrinters() must first be made for this function to return any value other than 0.
Arguments	None
Returns	Integer value representing the number of printers successfully enumerated by a previous call to EnumPrinters. No error condition exists for this function and the system error code is not modified.

Example	`//Enumeration count`
	`int m_iEnumeratedPrinters=GetEnumPrinterCount();`
	`//Display results`
	`TRACE(_T("Printer count=%d.\n"),m_iEnumeratedPrinters);`

ClearEnumList

Clears all enumerations of printer on the USB	
Prototype	`int WINAPI ClearEnumList(void);`
Description	Clears the list to all enumerated printers. All memory associated with the printer is released, and any open handles are closed. All active pipes to printers are terminated and communication to any active printer will be lost. If it is required to re-enumerate the USB then this function should be called first. Although any open handles will be closed and freed, it is good practice to close all open printer handles prior to calling this function, thus terminating communications with the printers in a controlled fashion.
Arguments	None
Returns	Integer value representing the error code. The system error code is also set.

Example	//Clear enumeration
	//Display results
	TRACE(_T("ClearEnumList returned=%d.\n"),ClearEnumList());

GetVendor

Gets the vendor ID of an enumerated device
Prototype `DWORD WINAPI GetVendor(int iIndex);`
Description Retrieves the vendor id of the device as a doubleword value. Vendor Id's are specific to usb device manufactures and can be used in the identification of a device. A call to EnumPrinters must have been made prior to calling this function
Arguments iIndex – The index of the device. This must be an integer value >=0<GetEnumPrinterCount(); If this value is outside of this range then an exception will be thrown.
Returns DWORD representing the vendor id. The system error code will be set to ERROR_SUCCESS regardless of whether the vendor id is valid or not.

Example
`//Check range prior to call`
`if(iDevIndex>=0&&iDevIndex< GetEnumPrinterCount)`
`{`
`DWORD dwVendorID=GetVendor(iDevIndex);`
`//Display results`
`TRACE(_T("Vendor ID for device index %d is %04X.\n"), iDevIndex,dwVendorID);`
`}`

GetProduct

Gets the product ID of an enumerated device
Prototype `DWORD WINAPI GetProduct(int iIndex);`
Description Retrieves the product id of the device as a doubleword value. product Id's are specific to usb device manufacture's individual products and can be used in the identification of a device. A call to EnumPrinters must have been made prior to calling this function
Arguments iIndex – The index of the device. This must be an integer value >=0<GetEnumPrinterCount(); If this value is outside of this range then an exception will be thrown.
Returns DWORD representing the product id. The system error code will be set to ERROR_SUCCESS regardless of whether the product id is valid or not.

Example
`//Check range prior to call`
`if(iDevIndex>=0&&iDevIndex< GetEnumPrinterCount)`
`{`
`DWORD dwProductID=GetProduct(iDevIndex);`
`//Display results`
`TRACE(_T("Product ID for device index %d is %04X.\n"),`
`iDevIndex,dwProductID);`
`}`

GetName

Gets the enumerated name of an enumerated device
Prototype `LPSTR WINAPI GetName(int iIndex);`
Description Retrieves the enumerated name for the device referenced by iIndex. An enumerated name is a manufactured name made up of the driver name, vendor id, product id & a reference number ensuring that each name on the USB is unique. The name can be up to 512 bytes long although more normally it will be between 20~40 characters. A call to EnumPrinters must have been made prior to calling this function
Arguments iIndex – The index of the device. This must be an integer value >=0<GetEnumPrinterCount(); If this value is outside of this range then an exception will be thrown.
Returns An LPSTR pointer value pointing to the null terminated string representing the enumerated name. The system error code will be set to ERROR_SUCCESS regardless of whether the product id is valid or not. The user should not delete this pointer or modify the contents pointed to directly.

Example
`//Check range prior to call`
`if(iDevIndex>=0&&iDevIndex< GetEnumPrinterCount)`
`{`
`  LPSTR pNameStr=GetName(iDevIndex);`
`  //Display results`
`  TRACE(_T("Enumerated Name for device index %d is %S.\n"), iDevIndex,pNameStr);`
`}`

OpenDevice

Opens a device and prepares it to send/receive data
Prototype `int WINAPI OpenDevice(int *pIndex);`
Description Opens a device and associates a unique handle with the device. An open device has the whole interface claimed(all control points & endpoints) making its use exclusive to the handle holder. Once successfully opened, data can be freely exchanged between the host and the device using the returned handle. Use CloseDevice to release the handle and free the attached device.
Arguments pIndex – Pointer to the index of the device. This must be an integer value >=0<GetEnumPrinterCount(); If this value is outside of this range then an exception will be thrown.
Returns An int representing the handle of the device, or INVALID_HANDLE_VALUE if the device open failed. If the returned handle is set to INVALID_HANDLE_VALUE then the system error code will be set to fully describe the problem. This can be retrieved by calling GetLastError().

Example
`//Check range prior to call`
`if(iDevIndex>=0&&iDevIndex< GetEnumPrinterCount)`
`{`
`  int iHandle=OpenDevice(&iDevIndex);`
`  Check result`
`  if(iHandle==INVALID_HANDLE_VALUE)`
`  {`
`  CString csError;`
`  csError.Format(_T("Device open failed. R/C=%d."), GetLastError());`
`  AfxMessageBox(csError,MB_ICONSTOP|MB_OK);`
`  return;`
`  }`
`  //device is open and may be used`
`  TRACE(_T("Device open success. Handle=%d.\n"), iHandle);`
`}`

CloseDevice

Closes and releases a device and associated handle
Prototype `int WINAPI CloseDevice(int *pHandle);`
Description. Closes a device and releases the resources associated with the device. Once closed the handle is invalidated and should be discarded.
Arguments phandle – Pointer to the handle of an open device. This handle must have been issued by a previous call to OpenDevice.
Returns ERROR_SUCCESS if successfully closed, otherwise the value is set to the system error code.

Example
`//Check range prior to call`
`if(CloseDevice(&ihandle)!=ERROR_SUCCESS)`
`{`
`  CString csError;`
`  csError.Format(_T("Close device failed. R/C=%d."), GetLastError());`
`  AfxMessageBox(csError,MB_ICONSTOP|MB_OK);`
`  return;`
`}`
`//Device has been successfully closed`
`TRACE(_T("Device close success. \n"));`

SendData

Send data to a device

Prototype

```
int          WINAPI SendData(int        iHandle,
                             LPCTSTR    pData,
                             LPINT      pDataLen,
                             int        iTimeout);
```

Description.

Sends data to a device with flow control. There is no practical limit on the amount of data that can be sent, but the call is synchronous and will not return until either the data send is completed, a timeout occurs, or an error occurs. Data is sent over a bulk endpoint which has a buffer size of 64 bytes. Data longer than this is transmitted in full 64 byte packets and flow control is checked between each packet. Where the receiver buffer is full the function will wait for a CTS signal for a period not exceeding the value in iTimeout. When CTS is set then the timeout value is reset to zero.

Arguments

ihandle – The handle of an open device. This handle must have been issued by a previous call to OpenDevice.

pData - Pointer to a buffer containing the data to transmit. All data in the buffer up to pDataLen is transmitted including any NULL characters.

pDataLen- Pointer to an integer containing the length of the data to transmit. This value is modified to show the actual amount of data that was successfully transmitted. If a timeout(or other error) occurs during transmission, this value can be used to establish a restart point.

iTimeout- The timeout value in milliseconds. The timeout countdown will only start when a buffer full, or other blocking condition occurs. Should this condition clear during the timeout period, then the period is reset.

Returns

ERROR_SUCCESS if successful, otherwise the value is set to the system error code.

Example

```
CString      m_csData=_T(“This is a data packet”);
TRACE(_T("Sending %d bytes of data.\n"),m_csData.GetLength());
int          iLen=m_csData.GetLength();
DWORD        m_dwStatusCode=SendData(m_iPrinterHandle,
             m_csData,&iLen,1000);
//Check return
if(m_dwStatusCode!=ERROR_SUCCESS)
{
       CString            csError;
       csError.Format(_T("SendData Failed - Reason=%d."),
                                              m_dwStatusCode));
       AfxMessageBox(csError,MB_ICONSTOP|MB_OK);
}
```

ReadData

Read data transmitted from a device

Prototype

```
int          WINAPI ReadData(int        iHandle,
                             LPSTR      pData,
                             LPINT      pLen);
```

Description.

Reads data from a device into the supplied buffer. If there is no data to retrieve, then ReadData will wait for a small period of time(typically 1000m/s) before returning. Data is read over a bulk endpoint which has a buffer size of 64 bytes, Where the supplied buffer is greater than 64 bytes, the USB will be accessed repeatedly until either there is no more data to receive or pLen has been reached. Where a data stream ends before the supplied buffer is full then pLen is set to indicate the actual number of bytes read from the device. If the number of bytes read reaches the value held in pLen then the function returns immediately and any remaining data remains pending. This data can be read by further calls to ReadData

Arguments

ihandle – Handle of an open device. This handle must have been issued by a previous call to OpenDevice.

pData - Pointer to a suitable buffer to receive the data. This buffer must be at least the size of the value held in pLen.

pLen Pointer to an int value containing the number of bytes to read. This value must not be greater that the supplied buffer size. On return this value is modified to indicate the actual number of bytes read.

Returns

ERROR_SUCCESS if successful, otherwise the value is set to the system error code.

Example

```
LPSTR        pBuffer=new char[256];
int          iLen=256;
DWORD        m_dwStatusCode=ReadData(m_iPrinterHandle,
             pbuffer,&iLen);
//Check return
if(m_dwStatusCode!=ERROR_SUCCESS)
{
       CString             csError;
       csError.Format(_T("ReadData Failed - Reason=%d."),
                                              m_dwStatusCode));
       AfxMessageBox(csError,MB_ICONSTOP|MB_OK);
}
else
{
       CString             csMessage;
       csMessage.Format(_T("ReadData success – Bytes read=%d.\n")
                       _T("Data=%S.\n"),iLen,pBuffer);
       AfxMessageBox(csMessage,MB_ICONINFORMATION|MB_OK);
}
```

SendCtrl

Send a control request or control data to a device
Prototype `int WINAPI SendCtrl(int iHandle, int iReqType, int iRequest, int iValue, int iIndex, LPINT pBufSize, LPCTSTR pData);`
Description. Sends a control request or control data to the specified device and reads any return values or data. The request is made over the control endpoint. This endpoint is bi-directional and synchronous so when a data return is expected the function will block until the return data is received or a timeout occurs. The timeout value is controlled by the USB and cannot be set by the user. Care must be taken when constructing the call as invalid data could cause the device to react unpredictably or to close the endpoint pipe. Control requests are high priority and actioned immediately by the device (or as close to immediate as the device can manage), thus uncontrolled control requests could cause undesired effects to the base functionality of the device. (IE jerky or uneven printing). A full list of control requests is available on request.
Arguments ihandle – Handle of an open device. This handle must have been issued by a previous call to OpenDevice. iReqtype-The request type. For the Ap1400 this value will always be `USB_TYPE_CLASS|USB_RECIP_DEVICE|USB_ENDPOINT_IN.` iRequest-The high level command request. Supported user requests are USB_GS_COMMAND and USB_BUFFERSTATE iValue - The sub-command. This value identifies the actual command from within the high level command group. Current supported values are `USB_GS_I_TYPE, USB_GS_ENQ_TYPE and USB_CAN_TYPE.` The low order 8 bits of this field contain the type request. For a list of these values please refer to the serial documentation for the GS 'I' command. `iIndex-` Data block index. This is used to reference a valid data block within the USB control structures. This value is not supported for user based requests and must be 0. `pBufSize-` Pointer to an integer field whch initially contains the length of the supplied buffer for data return. On return from the function, this value will contain the length of any data returned. `pData –` Pointer to a buffer to receive any returned data. This buffer must be large enough to receive all returned data up to a maximum of 64 bytes, and at least as large as the value specified in pBufSize.
Returns ERROR_SUCCESS if successful, otherwise the value is set to the system error code.

SendCtrl (Continued)

Example

The following example shows a ctrl call to retrieve the print buffer status

```
TCHAR        cReturn;
int          iRetLen=sizeof(TCHAR);
//Check the buffer state
DWORD              dwStatusCode=SendCtrl(iHandle,

USB_TYPE_CLASS|USB_RECIP_DEVICE|USB_ENDPOINT_IN,
                                          USB_BUFFERSTATE,
                                          0,
                                          0,
                                          &iRetLen,
                                          &cReturn);
        if(dwStatusCode!=ERROR_SUCCESS)
        {
               TRACE(_T("Buffer state get Failed...\n"));
               CString            csError;
               csError.Format(_T("GetBufferState failed - Reason=%d."),
                                                  dwStatusCode);
               AfxMessageBox(csError,MB_ICONSTOP|MB_OK);
               bError=TRUE;
               continue;
        }
        //if the error code is OK the check the return
        if(cReturn==BUFFER_BUSY)
        {
               TRACE(_T("Buffer state reports BUFFER_BUSY...\n"));
               //Wait for a small period and retry
               iMaxTime+=BUFFER_WAIT_PERIOD;
               Sleep(BUFFER_WAIT_PERIOD);
               continue;
        }
        else if(cReturn==BUFFER_CLEAR)
        {
               iMaxTime=0;
               TRACE(_T("Buffer state reports BUFFER_CLEAR...\n"));
               //Continue printing
               //……
        }
```

SendStatusReq

Get the current device status byte

Prototype

```
UCHAR  WINAPI SendStatusReq(LPINT       pHandle);
```

Description.

SendStatusReq provides a simple, easy to use method of receiving the printer status byte. The function sends a control request (refer to SendCtrl for a fuller description).
The function is a 'real time' request so constantly calling this function will have an impact on the printers performance

Arguments

phandle – Pointer to the handle of an open device. This handle must have been issued by a previous call to OpenDevice.

Returns

An unsigned character representing either the printer status byte, or 0xFF if an error occurs. If the value returned is 0xFF then the system error code will be set to indicate the error. This can be retrieved via a call to GetLastError.
If the value returned is not 0xFF then the byte represents the printer statuis byte and is formatted as follows.

Bit	0	Head Up sensor (if fitted)	0=Head down	1=Head up
Bit	1	Mech running	0=Stopped	1=Running
Bit	2	Data buffer empty	0=No	1=Empty
Bit	3	Paper out	0=No	1=Paper out
Bit	4	Reserved	Always 0	
Bit	5	Spool Mode	0=Normal	1=Spool
Bit	6	Error bit	0=No error	1=Error
Bit	7	Reserved	Always 1	

Example

```
//Get the status byte
UCHAR        cStatus=SendStatusReq(&m_iPrinterHandle);
if(cStatus&0x08)
{
       TRACE(_T("Paper is out….\n"));
       //Do paper out processing…
}
```

KrnlSendData

Send data without flow control
Prototype `int WINAPI KrnlSendData(int iHandle, LPCTSTR pData);`
Description. KrnlSendData sends data over the bulk endpoint but unlike SendData, there is no flow control. It is primarily available for firmware based tools from Able Systems, but in conjunction with SendCtrl could be used to allow end users to define their own flow control strategies. KrnlSendData, in isolation, will send data to the device irrespective of the devices buffer state. If the buffer becomes full, then buffer overwrites will occur. KernelSendData will send data in the buffer until the first NULL character is received. For this reason, it is not possible to send data which contains null characters, SendData must be used.
Arguments ihandle – Handle of an open device. This handle must have been issued by a previous call to OpenDevice. pData - Pointer to the buffer containing the data to be sent. The data must be terminated with a NULL character.
Returns ERROR_SUCCESS if successful, otherwise the system error code is returned.

Example
`//Get the status byte`
`CString csLine=_T("This is a line of data\r\n");`
`dwStatusCode=KrnlSendData(m_iOpenHandle,csLine);`
`TRACE(_T("Sent a line of data. Return code=%d\n"),dwStatusCode);`
`if(dwStatusCode)`
`{`
`  //We have an error`
`  CString csError;`
`  csError.Format(_T("KrnlSendData failed with code=%d.\n"), dwStatusCode);`
`  AfxMessageBox(csError,MB_ICONSTOP|MB_OK);`
`  //Close the printer`
`  CloseDevice(&m_iOpenHandle);`
`  return;`
`}`

You are always welcome to contact Able Systems or your local supplier for specific assistance.

We would also appreciate reports of any errors in our documentation, or suggested improvements.

For technical support please contact: Support@Able-Systems.com or call +44 (0) 1606 48621 and select option 4 for Technical Support.

[End]