Library PCLTCP

This library can be used for print out textual information via a network printer using PCL 5 (e.g. all HP printers with integrated ethernet interface).
The library uses PCL ASCII sequences like described inside the HP PCL reference, and ASCII characters for texts. Therefore the printer needs at least one integrated font.
Not all printers support the full set of PCL5 commands, so the possibilities to change fonts, change font style, a.s.o. depend on the printer functionality.

A page is organized with columns and lines, the number of columns and lines per page depends on the font size, font pitch, page margins and paper format.
With format A4, no margins and default font size and pitch, a Page normally has about 78 characters per line (colums) an 60 lines.

The library only allocates some memory for one or more pages, and some library function calls concat the ASCII text and ASCII PCL commands. If a page is ready, the library sends the data in RAW format to the printer via TCP (default port for RAW printing is 9100).

Use of network spooling protocols like LPR is not possible, the printer must support RAW printing!

Function and function block interface description

All function blocks /functions provide status information at the output ".status" or as return value.

- 0, everything is ok
- 65535, FUB is busy an must be called again in next cycle (only asynchronous FUBs)
- ```
  PCLTCP_ERR_INVALID_IDENT : UINT := 2; (* ident is not valid *)
  PCLTCP_ERR_NO_IDENT : UINT := 1; (* ident is 0 *)
  PCLTCP_ERR_MEMORY : UINT := 3; (* no more page memory left *)
  PCLTCP_ERR_NOIP : UINT := 4; (* no ip address given *)
  PCLTCP_ERR_VALUE : UINT := 5; (* wrong value *)
  PCLTCP_ERR_VALUE_COLRIGHT : UINT := 6; (* illegal value for right margin *)
  PCLTCP_ERR_VALUE_NRROWS : UINT := 7; (* illegal value for number of rows per page *)
  ```
- Any other error number: see Automation Studio help

All function blocks / functions except of "PCLTcpInit" need an ident as input, this ident is provided by "PCLTcpInit". The ident is valid if "PCLTcpInit" output ".status" is 0.
  ```

PCLTcpInit

```
(*Initialize function, ONLY FOR INIT USE*)
FUNCTION_BLOCK PCLTcpInit
	VAR_INPUT
		len : UDINT; (* len of internal page memory in byte *)
	END_VAR
	VAR_OUTPUT
		status : UINT; (* state of fub / function call *)
		ident : UDINT; (* ident for other functions *)
	END_VAR
END_FUNCTION_BLOCK
```

This function block allocates memory for the page(s) and returns an ident, needed by all other functions / function blocks.
PCLTcpInit must be used inside INIT program only!

PCLTcpPrint

```
(* send page memory to printer *)
FUNCTION_BLOCK PCLTcpPrint
	VAR_INPUT
		ident : UDINT; (* ident from init function *)
		pIpAddr : UDINT; (* pointer to ip address of printer, given as string *)
		port : UINT; (* printer tcp port, if set to 0 default = 9100 is used *)
		pInterface : UDINT; (* pointer to string with interface address *)
	END_VAR
	VAR_OUTPUT
		status : UINT; (* state of fub / function call *)
	END_VAR
END_FUNCTION_BLOCK
```

This function block sends the page(s) to the printer IP address given at ".pIpAddr".
PCLTcpInit is asynchronous, so it must be called until the output ".status" is not equal to BUSY (65535).
The default port data has to be sent to is 9100.

PCLResetMemory

```
(* initialize page memory *)
FUNCTION PCLResetMemory : UINT
	VAR_INPUT
		ident : UDINT; (* ident from init function *)
	END_VAR
END_FUNCTION
```

This function resets the internal memory allocated with "PCLTcpInit".
It's needed if a page was printed and a new page should be generated.

PCLAddString

```
(* add a character or command string to page *)
FUNCTION PCLAddString : UINT
	VAR_INPUT
		ident : UDINT; (* ident from init function *)
		pString : UDINT; (* pointer to string to add *)
	END_VAR
END_FUNCTION
```

This functions adds a string to the page memory, this can be

- an ASCII string with data to printout
- an PCL command string

PCLAddLine

```
(* add a character or command string to page, ends with PCL_CRLF *)
FUNCTION PCLAddLine : UINT
	VAR_INPUT
		ident : UDINT; (* ident from init function *)
		pString : UDINT; (* pointer to string to add *)
	END_VAR
END_FUNCTION
```

This functions works like “PCLAddString”, but appends after the given string a PCL command string with a carriage return and line feed.

PCLAddChar

```
(* add a single character value to page *)
FUNCTION PCLAddChar : UINT
	VAR_INPUT
		ident : UDINT; (* ident from init function *)
		charVal : USINT; (* value to add *)
	END_VAR
END_FUNCTION
```

This function adds a single character to the page, given as a unsigned short value.
It can be used to add some special characters who are inside the font table, but not at the keyboard.

PCLAddCRLF

```
(*Adds a PCL conform CR LF via cursor positioning commands*)
FUNCTION PCLAddCRLF : UINT
	VAR_INPUT
		ident : UDINT; (* ident from init function *)
	END_VAR
END_FUNCTION
```

This function adds a carriage return and line feed to page.
Because of some different printer behaviour using ASCII CR+LF, this command uses PCL cursor positioning commands inside.

PCLAddParam

```
 (* add a PCL command parameter to page *)
FUNCTION PCLAddParam : UINT
	VAR_INPUT
		ident : UDINT; (* ident from init function *)
		param : UINT; (* wanted parameter, use constants PCL_PARAM_xxxx *)
		value : REAL; (* wanted value of parameter *)
	END_VAR
END_FUNCTION
```

This function adds a PCL command string with the parameter ".param" inside to the page memory.

The following constants can be used:

```
PCL_PARAM_FONTPITCH_PRIMARY : UINT := 1; (* font width in characters per inch *)
PCL_PARAM_FONTHEIGHT_PRIMARY : USINT := 14; (* font height in points *)
PCL_PARAM_FONTCODE_PRIMARY : USINT := 2; (*font code*)
PCL_PARAM_FONTSTYLE_PRIMARY : USINT := 3; (*font style*)
PCL_PARAM_MARGIN_COL_LEFT : USINT := 4; (*set margin left, in columns*)
PCL_PARAM_MARGIN_COL_RIGHT : USINT := 5; (*set margin right, in columns*)
PCL_PARAM_MARGIN_ROW_TOP : USINT := 6; (*set margin top, in lines*)
PCL_PARAM_MARGIN_NR_LINES : USINT := 7; (*set maximum number of lines per page, in lines*)
PCL_PARAM_CURSOR_ABS_COL : USINT := 8; (*set cursor column position absolute*)
PCL_PARAM_CURSOR_ABS_ROW : USINT := 9; (*set cursor row position absolute*)
PCL_PARAM_CURSOR_REL_POS_COL : USINT := 10; (*set cursor row position, relative, downwards*)
PCL_PARAM_CURSOR_REL_POS_ROW : USINT := 11; (*set cursor column position, relative, to right*)
PCL_PARAM_CURSOR_REL_NEG_COL : USINT := 12; (*set cursor row position, relative, upwards*)
PCL_PARAM_CURSOR_REL_NEG_ROW : USINT := 13; (*set cursor column position, relative, to left*)
```

Normally, the usage of this function is not necessary, because for setting this parameters several own functions are available, see:

- PCLSetCursorAbs
- PCLSetCursorRel
- PCLSetPageMargins
- PCLSetFontSize

PCLAddTestPage

```
(* add a test page to page memory *)
FUNCTION PCLAddTestPage : UINT
	VAR_INPUT
		ident : UDINT; (* ident from init function *)
		printTestSentence : BOOL;
		printAsciiCharset : BOOL; (* print out table of value and sign from 32 - 126 *)
		printExtCharset : BOOL; (* print out table of value and sign 128 - 255 *)
		printInternalCfg : BOOL; (*print out some internal information*)
		printFormFeed : BOOL; (* add form feed at end of page *)
	END_VAR
END_FUNCTION
```

This function adds a test page with markers for lines and columns to page memory.
User can decide if some additional information should be printed out:

- test sequence to see if printer can print cursive and underlined font styles
- ASCII characters set with character and value
- Extended charset with character and value
- Some library setup information

The function is normally used to check out if the printer works fine and to see some printer basic conditions like character per page and character set.

PCLSetCursorAbs

```
 (* set a absolute cursor position, x= row, y = column *)
FUNCTION PCLSetCursorAbs : UINT
	VAR_INPUT
		ident : UDINT; (* ident from init function *)
		x : UINT; (* cursor row *)
		y : UINT; (* cursor column *)
	END_VAR
END_FUNCTION
```

This function moves the cursor to the x (row) and y (line) position, absolute to top left corner of page. Page margins are not considered!

PCLSetCursorRel

```
(* set a relative cursor position, x= row, y = column *)
FUNCTION PCLSetCursorRel : UINT
	VAR_INPUT
		ident : UDINT; (* ident from init function *)
		x : INT; (* cursor row *)
		y : INT; (* cursor column *)
	END_VAR
END_FUNCTION
```

This function moves the cursor to the x (row) and y (line) position, relative to the actual cursor position. Negative values move cursor, backwards, positive move forward.
Page margins are not considered!

PCLSetPageMargins

```
(*Setup margins of page*)
FUNCTION PCLSetPageMargins : UINT
	VAR_INPUT
		ident : UDINT; (* ident from init function *)
		colLeft : INT; (* left margin in columns *)
		colRight : INT; (* right margin in columns *)
		rowTop : INT; (* top margin in rows *)
		rowsPerPage : INT; (* number of rows per page *)
	END_VAR
END_FUNCTION
```

This function sets the page margins, e.g. to leave some space on left and right side of page.

PCLSetFontSize

```
(*Setup font height and pitch, height in pixel, pitch in chars per inch*)
FUNCTION PCLSetFontSize : UINT
	VAR_INPUT
		ident : UDINT; (* ident from init function *)
		height : REAL; (* height in pixel *)
		pitch : REAL; (* pitch in characters per inch *)
	END_VAR
END_FUNCTION
```

This function sets the font size, with height in pixel, and width (“.pitch”) in characters per inch.

Constants description

Inside the library exists a set of predefined constants with PCL command strings.
Some of the constants are used internal only, but some others are maybe interesting when building a page.
Notice that much of PCL commands are printer dependent, so there's no guarantee that they will work at the printer as wanted.

Of course it makes no sense to put all PCL commands as constants inside the library.
But adding needed command sequences inside a page is also possible by using PCLAddString() with the command sequence instead of the constants.

Set cursor to next line, row 0

```
PCL_CRLF : STRING[20] := '$1B&a+1R$1B&a0C';
(*CR and LF made by PCL cursor positioning commands*)
```

Do a Form Feed (release page)

```
PCL_FF : STRING[20] := '$0C';
(*Form Feed escape sequence*)
```

Indiviual initialization sequence, sets some printer setup values

```
PCL_PAGE_INITSEQ_1 : STRING[255] := '$1B&u300D$1B&l1X$1B&126A$1B&l0O$1B&l0F$1B&a0L$1B(19U';
(*Individual init sequence with commands: set 300 dpi, 1 copy, paper A4, orientation portrait,
0 line top margin, 0 col left margin, charset ANSI*)
```

Set character set to ISO8859

```
PCL_CMD_CHARSET_ISO8859 : STRING[20] := '$1B(0N';
(*set charset IS8859, PRINTER DEPENDENT*)
```

Set character set to ASCII

```
PCL_CMD_CHARSET_ASCII : STRING[20] := '$1B(0U';
(*set charset ASCII, PRINTER DEPENDENT*)
```

Set character set to ANSI

```
PCL_CMD_CHARSET_ANSI : STRING[20] := '$1B(19U';
(*set charset ANSI, PRINTER DEPENDENT*)
```

Set paper format to DIN A4

```
PCL_CMD_PAPER_A4 : STRING[20] := '$1B&126A';
(*set paper format to A4 *)
```

Set paper orientation to landscape

```
PCL_CMD_ORIENTATION_LANDSCAPE : STRING[20] := '$1B&l1O';
(* set paper orientation *)
```

Set paper orientation to portrait

```
PCL_CMD_ORIENTATION_PORTRAIT : STRING[20] := '$1B&l0O';
(* set paper orientation *)
```

Set printer font to Courier (if installed on printer!)

```
PCL_CMD_FONT_COURIER : STRING[20] := '$1B(s3T';
(* set printer font, PRINTER DEPENDENT *)
```

Set printer font to CGTimes (if installed on printer!)

```
PCL_CMD_FONT_CGTIMES : STRING[20] := '$1B(s4101T';
(* set printer font, PRINTER DEPENDENT *)
```

Set printer font to Letter Gothic (if installed on printer!)

```
PCL_CMD_FONT_LETTERGOTHIC : STRING[20] := '$1B(s6T';
(* set printer font, PRINTER DEPENDENT *)
```

Set printer font to Lineprinter (if installed on printer!)

```
PCL_CMD_FONT_LINEPRINTER : STRING[20] := '$1B(s0T';
(* set printer font, PRINTER DEPENDENT *)
```

Set printer font to Helvetica (if installed on printer!)

```
PCL_CMD_FONT_HELVETICA : STRING[20] := '$1B(s4T';
(* set printer font, PRINTER DEPENDENT *)
```

Set printer font to Script (if installed on printer!)

```
PCL_CMD_FONT_SCRIPT : STRING[20] := '$1B(s7T';
(* set printer font, PRINTER DEPENDENT *)
```

Set printer font to Prestige (if installed on printer!)

```
PCL_CMD_FONT_PRESTIGE : STRING[20] := '$1B(s8T';
(* set printer font, PRINTER DEPENDENT *)
```

Set printer font to Courier scalable (if installed on printer!)

```
PCL_CMD_FONT_COURIERSCAL : STRING[20] := '$1B(s4099T';
(* set printer font, PRINTER DEPENDENT *)
```

Set printer font to Universe (if installed on printer!)

```
PCL_CMD_FONT_UNIVERSE : STRING[20] := '$1B(s4148T';
(* set printer font, PRINTER DEPENDENT *)
```

Set printer font to Arial (if installed on printer!)

```
PCL_CMD_FONT_ARIAL : STRING[20] := '$1B(s16602T';
(* set printer font, PRINTER DEPENDENT *)
```

Set printer font to proportional character set (if supported by font!)

```
PCL_CMD_CHAR_PROPORTIONAL : STRING[20] := '$1B(s1P';
(* set font spacing, PRINTER AND FONT DEPENDENT *)
```

Set printer font to fixed character set (if supported by font!)

```
PCL_CMD_CHAR_FIXED : STRING[20] := '$1B(s0P';
(* set font spacing, PRINTER AND FONT DEPENDENT *)
```

Start underlining

```
PCL_CMD_UNDERLINE_ON : STRING[20] := '$1B&d1D';
(* switch underline on *)
```

Stop underlining

```
PCL_CMD_UNDERLINE_OFF : STRING[20] := '$1B&d@';
(* switch underline off *)
```

Set font weight to bold

```
PCL_CMD_BOLD_ON : STRING[20] := '$1B(s3B';
(* switch bold on *)
```

Set font weight to normal

```
PCL_CMD_BOLD_OFF : STRING[20] := '$1B(s0B';
(* switch bold off *)
```

Set font style to normal

```
PCL_CMD_STYLE_NORMAL : STRING[20] := '$1B(s0S';
(* set font style to normal *)
```

Set font style to italic

```
PCL_CMD_STYLE_ITALIC : STRING[20] := '$1B(s1S';
(* set font style to italic *)
```

Store actual cursor position

```
PCL_CMD_CURSOR_PUSH_POSITION : STRING[20] := '$1B&f0S';
(*store actual cursor position*)
```

Recall last stored cursor position

```
PCL_CMD_CURSOR_POP_POSITION : STRING[20] := '$1B&f1S'
 (*restore last pushed cursor position*)
```

Demo task

```
PROGRAM _INIT

(* init functionality, allocate 50 kB of memory for pages *)
    PCLTcpInit_0(len := 50000);
    ident := PCLTcpInit_0.ident;

END_PROGRAM


PROGRAM _CYCLIC

    IF PCLTcpInit_0.status = 0 THEN

        CASE test OF

            10:
              (* reset internal page memory, for creating a new page *)
              PCLResetMemory(ident);

              (* print out a test page *)
              PCLAddTestPage(ident, TRUE, TRUE, TRUE, TRUE, TRUE);

              (* set some basic printer initialization commands *)
              PCLAddString(ident, ADR(PCL_PAGE_INITSEQ_1));

              (* set font size a little bit smaller then normal *)
              PCLSetFontSize(ident, 8.0, 14.0);

              (* set page margins*)
              PCLSetPageMargins(ident, 5, 70, 5, 60);

              (* ---- print headline ----- *)
              (* bold and underline on *)
              PCLAddString(ident, ADR(PCL_CMD_BOLD_ON));
              PCLAddString(ident, ADR(PCL_CMD_UNDERLINE_ON));
              (* print some text *)
              PCLAddLine(ident, ADR('PCLTCP library demo task'));
              (* bold and underline off *)
              PCLAddString(ident, ADR(PCL_CMD_UNDERLINE_OFF));
              PCLAddString(ident, ADR(PCL_CMD_BOLD_OFF));

              (* ---- print some data ---- *)
              FOR ii := 1 TO 10 DO
                itoa(ii, ADR(szTmp));
                PCLAddString(ident, ADR('Line '));
                PCLAddString(ident, ADR(szTmp));
                PCLAddLine(ident, ADR(': the quick brown fox jumped over the lazy dog'));
              END_FOR

              (* ---- go somewhere to the end of page, print footer ---- *)
              PCLSetCursorAbs(ident, 5, 57);
              PCLAddLine(ident, ADR('--------------------------------------------------'));
              PCLAddLine(ident, ADR('End of page'));
              PCLAddString(ident, ADR(PCL_FF));
              test := 20;

            20:
              (* print it! *)
              PCLTcpPrint_0(ident := ident, pIpAddr := ADR('10.49.20.24'), port := 9100, pInterface := 0);
              IF PCLTcpPrint_0.status <>16#FFFF THEN
                test := 0;
              END_IF

        END_CASE

    END_IF

END_PROGRAM
```