

CyberSC Scanner/Printer Interface API.

****Copyright(c) 2024 Scanprint Ltd.**** *This program source code is owned by Scanprint Ltd and, subject to any existing agreements or rights of other parties, ScanPrint Ltd is the owner of this source code. The source code may not be copied or disclosed to a third party without permission in writing from ScanPrint Ltd, nor may it be used for any purpose other than that for which it has been supplied.*

HISTORY

1. Add the ForceScan Stay option. Tobias Selwood 03/10/2019 15:04:17
2. Add the First version of the TWAIN API. Tobias Selwood 20/02/2019 13:03:43
3. Add a method to revers out any paper in the scanner. Tobias Selwood 26/10/2018 17:37:49
4. Add the ability to abort a scan after geting part of the data. Tobias Selwood 25/10/2018 21:06:04
5. Start stubbing in the "Dipping" API Tobias Selwood 18/10/2018 10:27:32
6. Add synchronous API subset - Blocking functions for threading model control. Tobias Selwood 21/02/2018 15:07:02
7. Cleaned up documentation. Remove Redundant info - no functional changes. Owen Cullum 16/08/2022
8. Set the default location for the calibration file (Cfg1xxx.dat) to 'C:\programdata\scanprint\scanner'. Owen Cullum 10/06/2024
9. Add Re-calibrate feature. It'ss the same CV_CAL_CALIBRATE but deletes the existing calibration file first. owen Cullum 11/06/2024 22:07
10. Added comments regarding the deprecated DIICv_** functions. These are kept for backwards compatibilty only. Owen Cullum 23/12/2024
11. Support for 64 bit builds. Changed int / dword params to uint_ptr type where those params are used as pointers. Owen Cullum
12. When re-calibrating always save a new file if one does not already exist. Owen Cullum 28/02/2026

OVERVIEW

CyberSC provides a comprehensive library of high level services to operate the scanner and the optional receipt printer.

From V2.0.0.0 CyberSc supports ECP or USB. Note that hot switching is NOT supported but an application restart should be sufficient. Valid Configs:

1. ECP scanner and ECP printer
2. ECP scanner and USB printer
3. USB scanner and USB printer

Applications must load this DLL **in process** to be able to access exported data. The API is highly optimized for performance. The scanned document image itself is never copied. The DLL uses undecorated exports to facilitate runtime loading.

It is assumed that the high-level application can be trusted, therefore internal pointers are returned down to the driver. It is assumed that C, C++ or C# will be used to develop applications using the CyberSc.dll. VB support has been deferred.

Important paramters can be directly set through the application interface and these are also saved in the **registry** but these are not documented here and **shouldn't be modified directly**. This means however, most settings are persistent across DLL activations. (e.g scanner device, scanning resolution, motor current etc). This allows for the writing of very simple applications that use preconfigured settings and developers do not have to concern themselves with a large part of the application interface.

The thread priority of time critical acquisition threads can be set from the registry. The defaults should suffice for all cases there should be no need to change these but the possibility exists. The entries to adjust are PriScan (default 15), PriStatus (default 2).

Using the configuration registry settings and a machine speed measurement during startup the optimum speed is always used. Note however, these values are cached in the (cnf.dat and Cfg*.dat files). For this reason these files should **never be copied from machine to machine**, and if for some reason the speed of the machine is changed these files should be deleted and recreated by a calibration (E.g. via ScSCalibrate()).

The API supports the concept of a transaction cycle as follows:

- 1 - The application is waiting for user input or the scan callback
- 2 - Scanner thread (created during ScInit) detects paper and starts scanning
- 3 - The scanner is disabled while the customer callback is executing.
- 4 - The customer callback is executed in the context of the scanner thread
- 5 - Typically customer code calls ScPOn() and calls print calls to print a receipt
- 6 - Customer code calls SCPEnd(...) to terminate the transaction and print the barcode and cut the paper (print a logo in the area wasted by cut positioning).
- 7 - Customer code calls ScPOff() to terminate printing and restart scanning mode by exiting the callback.
- 8 - Scanner processing can also be done outside the callback but additional calls have to be made to stop the scanner and then restart it when done with the image and / or printer.

Two sets of APIs calls are provided: Callback based asynchronous calls or Blocking synchronous function calls.

GETTING STARTED

See the 'Readme' file supplied with the Scanner / Printer Development kit for a getting started guide.

KNOWN ISSUES.

NOTE. The scanner / printer API is always opened for exclusive access. **Only one application at a time can load CyberSc.dll.** However, Single load is **NOT** currently enforced. If the DLL is loaded multiple times using USB (not using the ECP driver) the first app will have control and seem to work ok and for the other the scanner will fail. Additionally, if an attempt is made to print from the second app this may corrupt printing form the first app or cause it fail. Therefore it is *assumed all applications will control exclusion.*

CONFIGURATION

Master control parameters are applied to the DLL via the Registry, via calibration files and support files. The possible scanner speed is determined by a number of factors:

- CIS frequency,
- data transfer rate to host &
- mechanism feed limit.

The Support files required are Font.asc, Logo.prn. If no logo is required a BLANK logo file should be used.

Calibration files (cnf.dat Cfg*.dat) are automatically generated with default values whenever they are missing during initialization or calibration. If a calibration is required a warning is generated.

REGISTRY SETTINGS

Many values are passed down to the scanner device driver and depend on its functionality.

Registry path: HKEY_CURRENT_USER\Software\ScanPrint\V5.00\Acquisition\

VALUES

MaxImageSize	DWORD defaults to 4320000 ; the image buffer size in bytes, memory can be used more efficiently if this is a multiple of page size = 0x1000. This value is not affected by a reinit. To take effect the DLL has to be unloaded and reloaded.		
MaxCISFreq	DWORD defaults to 6 ; minimum low clock cycles which determines the maximum CIS clock frequency as below. Values below are correct for 25MHz clock (for ISA versions the clock is 33MHz).		
	clock low	max CIS frequency	
	1	5.000 MHz	
	2	4.166 MHz	
	3	3.571 MHz	
	4	3.125 MHz	
	5	2.778 MHz	
	6	2.500 MHz	
MaxMotor	DWORD defaults to 12 ; minimum low clock * 2 cycles which determines the maximum motor speed. Values below are correct for 25MHz clock (for ISA versions the clock is 33MHz).		
	value	max at 200dpi	max at 100 dpi step frequency
	6	305mm/s	305mm/s 2446Hz
	7	305mm/s	277mm/s 2223Hz
	8	254mm/s	254mm/s 2038Hz
	9	254mm/s	235mm/s 1881Hz
	10	218mm/s	218mm/s 1747Hz
	11	218mm/s	203mm/s 1530Hz
	12	191mm/s	191mm/s 1528Hz
	13	191mm/s	179mm/s 1438Hz
	14	169mm/s	169mm/s 1351Hz
	15	169mm/s	160mm/s 1287Hz
Power	DWORD defaults to 0xd0 ; printer power adjustment. Values from 0x80 (lightest) to 0xd0 (darkest) are valid.		
DefMotI	DWORD defaults to 40 ; the scanner motor current setting used during automatic calibration.		
Gain	DWORD defaults to 99 ; the light Level automatically set on entry to Gain adjust oscilloscope mode. This function is normally not used at an application level		
Flags	DWORD default is 0x0x11739 ; The bits are passed on to a cooperating application and not all are directly used by this dll. An *du* indicates direct usage by the DLL. Bit mask of various as below:		
	0x0000 0001 log all messages to a file <tstlog.txt>.		
	0x0000 0002 verbose - enable more elaborate trace primarily in image analysis.		
	0x0000 0004 do not hide initial positioning marks.		
	0x0000 0008 stop on first blank game area.		
	0x0000 0010 profile - report scan print analyze times.		
	0x0000 0020 generate and check high level barcode checksum in the 14th digit		
	0x0000 0040 rest the message window each scan to avoid scrolling.		
	0x0000 0080 *du* indicates the scanner has a gear ratio of 400steps/25.4mm.		

Normal gear ratio is 200steps/25.4mm.

0x0000 0100 load with Image Analysis Mode ON. (Used by EngBench)
0x0000 0200 load with Image Zoom Factor "/2". (Used by EngBench)
0x0000 0400 load in Maximized mode. (Used by EngBench)
0x0000 0800 *du* DO NOT report data overrun errors (for demo's).
0x0000 1000 display true image x,y coordinates or CIS frequency. (Used by EngBench)
0x0000 2000 display hot area/threshold info for dirty hot/cross zones. (Used by EngBench)
0x0000 4000 *du* Disable acceleration/deceleration during scan start/stop.
0x0000 8000 set displays heirarchy strings for football coupons
0x0001 0000 set, disables server coms
0x0002 0000 set, disables retry dialog when coms error
0x0004 0000 set, disables game interpretation so heirarchy strings are printed
0x0008 0000 set, disables double hight characters on receipt (pan malasian slips)
0x0010 0000 *du* set, Xac cutting position leaving receipt geometry identical
but logo quality may suffer

0x0020 0000 *du* set, Xac cutting with good quality logo (about 5mm more paper used)
when set bit 0x100000 is ignored

0x0040 0000 *du* set, Adds more space on the end of receipt to bandaid terminal
cutter position misalignment.

0x0080 0000 *du* set, Unconditionally does a synch /feed sequence on statup. This is
intended for a gaming macine that must be off to change paper. Note by
default nothing is done. You must change the registry to enable.

0x0100 0000 set, load with printer hot keys enabled - should reset this for
non terminal environments.

0x0200 0000 set, save each barcode scanned on the printer barcode reader to a
file "Barcodes.txt". Note this is a change from previous behaviour.

FeedLength	DWORD default 50mm, the length to feed in mm for the scrap section (to prevent jams?) new Auto home/feed / cut sequence when come on line after a paper/head open. If the value is 50 (the default) the function is disabled.
FeedDelay	DWORD default 2200mili sec, the time to delay between a good status and the new Auto home/feed / cut sequence when come on line after a paper/head open occuring. This is disabled if FeedLength is 50.
Osc	DWORD default 25Mhz, the frequency of scanner clock in MHz. For ISA set to 33Mhz. NOTE: In W2K version this entry needs to be duplicated in the driver ServiceEntry (Cy).
PriScan	DWORD (default 15) priority of scanner,I/O threads (USB versions since 2.0.0.8 (21/11/2006) only)
PriStatus	DWORD (default 2) priority of status thread (USB versions since 2.0.0.8 (21/11/2006) only)
RunoutScale	DWORD (default 20) devisor for the lines scanned so far during a JAM so compensation can be made for the fact reverse lines are not counted in real time due to the white detection loop running at about 20ms. This allows adjustement of the max rollback (to try and remove the paper from the scanner) if white detection or paper presence is jammed. If "RunoutScale" is set to 0 or 1, no scaling occures. This nan be changed if the rollback is unduly short or long (most likely due to loop time variations).
Delay	DWORD (default 0, disabled) Configurable delay specified in milli seconds which is the mimimum time between scanner starts. This is only in "autointit" versions. This allows limiting the duty is loop tests (paper always present).

DEBUGGING

You should ALWAYS have a copy of EngBench (in the same directory as CyberSc.dll) handy to resolve issues.

Where possible returns from the API functions should be checked for success and appropriate dianostic messages issued. Some callbacks have been added to facilitate debugging

ScInit() distinguishes between driver load failure and scanner not found. It fails if the driver fails to load or initilise for some reason. In this case no printing is possible and an app relying on printing should not continue to load. (E.g. running 2 apps using the printer at the same time or 2 instances of the same). On success ScInit() returns the version numbers of driver components.

There are some text mode error callback messages (see below) relevant to the user. These can be directly popped up in a window or log file to alert the user of the specific problem.

All debugging/diagnostic messages are prefixed with a period '.' (YES it's a brilliant GUI design!) So messages can be easily filtered out so as to not annoy users with debugging messages but still retain debugging functionality as desired. (E.g. "Printer font NOT found")

ERROR MESSAGES

Text mode error messages;

```
"Scan raise priority failed pri=%d gle=%d",dwPriScan,GetLastError());

"Expand: image too high");

"Printer Font NOT found");

"Scanner NOT found");          // for those that don't want to interpret all error messages

".Print Sts [%3X]",Status);

".IMPBitMap: impEv created");

"IO raise priority failed pri=%d gle=%d",dwPriScan,GetLastError());

".USB scanner not found");

".USB scanner <%s>",szDeviceName);

"Get pipe info failed");

"USB Pipe count incorrect %d (expect 2)",m_pipeCount);

"Status raise priority failed pri=%d gle=%d",dwPriStat,GetLastError());

"ECP Driver Open Failed (%d)",GetLastError());

"Driver Info Failed");

"Driver Alloc Failed");        // Windows NT

".Printer type <%d>",gPrinter);

"[Cnf.dat] not found");        // Windows NT
```

Debug Callbacks

The error codes given hereunder may change at any time. They are associated with CV_DEBUG callback strings. They are intended for display only (or logging) for debugging purposes by the error callback procedure (callbacks with a leading period '.' are informational in nature and can be ignored).

It is recommended an app at least log the following to aid in problem solving: "Config not FOUND, PLEASE CALIBRATE"

The following correspond to driver returned errors (defined in e.h)

".Lost Scan(s)"	EERLINELOST	
"Read timeout"	EERRRDDAT	
"ECP phase 1 timeout"	EERRNEGOTIATE1	
"ECP phase 2 timeout"	EERRNEGOTIATE2	
"F2R flush timeout"	EERRFWD2REV1	
"F2R ack timeout"	EERRFWD2REV2	
"R2F ack timeout"	EERREV2FWD	
"CPU too slow"	EERRTOOSLOW	
""	NOFREEBUF	
".Scan timeout"	EERRSCANTIMEOUT	
".Scan jam"	EERRJAM	
"DMA timeout"	EERRDMATO	
"No buffer"	EERNOBUF	
"Internal error"	EERPROGERROR	- submit error report
"Low CIS Output"	EERRLIGHT	- during calibration - check scanning speed, scanner CIS leds/drive problem
"Paper Timeout"	EERRPAPERTIMEOUT	- during calibration - follow calibration procedure, scanner paper detector problem
".USB Transfer"	EERRUSB	- check cables - scanner powered

PRINTING NOTES

Printer only configurations are also supported, however null scanner callback is still required and ScPon/Off sequences are advised in transaction or receipt printing to maximise error recovery.

If the barcode reader is used in printer only ECP configurations, this requires special consideration (I.e. ScPOn must be in effect when barcode scans are to be processed. It is still recommended to start a print cycle with a ScPOn).

There is an automatic printer initialisation sequence every time the paper is loaded.

When an app requires to print asynchronously (not in the scanner callback) it must first stop the scanner. Do its printing then restart the scanner to allow scanning again. The functions to call are:

```
ScSCalibrate(CV_CAL_STOPSCAN,0);    // this call will block until the interface is free.
                                    // It is possible to get any callback while waiting here,
                                    // including scanner callback

ScPOn()                             // this call does not block
.....

ScPOff()
ScSCalibrate(CV_CAL_STARTSCAN,0);
```

If the driver initialises without error, this does not mean the printer is in a print ready state. The app must monitor the asynch error callback, OR, it can determine if a printer is ready to print by calling ScPStatus() at any time within the ScPOn()/ScPOff() sequence. This way the app can determine if the printer is ready before actually printing anything and without using the asynch error callback. The return value of ScPStatus() is the printer status mask, the printer is on line when this mask is 0.

Most print functions return a status. The app should act on printer spool errors.

it is possible to test for printer presence before doing any print calls (that will trigger printer errors if one is not on line). The printer type is reported in a .DEBUG callback. For more details please see CV_CAL_GETPRINTERTYPE ScSCalibrate() step.

If a logo.prn is not accessible, in XAC compatibility mode no cut occurs. This is because the cut is performed half way through printing the logo. This may look like the printer is malfunctioning!

HEADER DEFINES / STRUCTURES

```
ifndef DLLBUILD                      // declared in the make file
#define PORTEE_DLLCV_LB DllImport
#else
#define PORTEE_DLLCV_LB DllExport    // scope of functions exported
#endif

#define CIMGmPRINT      0x1
```

• Flags Bit Masks

The dwFlags bit masks (set in Flags DWORD in the registry See above, used by EngBench / external apps)
('N -> action description' means bit value N gives the result 'action description'.)

```
#define CFLGGEAR      0x0080    // 0 -> times X1 gear ratio, 1 -> times *2 gear ratio
#define CFLGANA      0x0100    // 0 -> default image analysis OFF          (for EngBench only)
#define CFLGZOOM      0x0200    // 0 -> default zoom / 2 OFF          (for EngBench only)
#define CFLGMAX      0x0400    // 0 -> show normal, 1 -> show maximised (for EngBench only)
#define CFLGLOST      0x0800    // 0 -> report lost line errors, 1 hide
#define IFXYPOS      0x1000    // show true image x,y coords or CIS frequency. (for EngBench only)
#define IFCROSS      0x2000    // show hot area/threshold infor for dirty hot/cross zones. (for EngBench only)
#define CFLGRAMP      0x4000    // 0 -> use ramp, 1 -> no ramp
#define CFLGEPSON     0x00100000 // 1 -> XAC cutter but mimicking EPSON geomtry (logo split to cut)
#define CFLGSHORT     0x00200000 // 1 -> XAC cutter new geometry (logo not split to cut)
#define CFLGMECHCRAP   0x00400000 // cutter offset on epson terminals
#define CFLGFEEEDONSTART 0x00800000 // 1 -> feed Auto home/Feed on start
#define CFLGSPPOOLSYNCH 0x10000000 // 1 -> StartScanner blocks until all USB spooling as completed
                                   // (this prevents simultaneous scanning and printing). For ECP
                                   // this has no effect (scanning cannot proceed in parallel
                                   // with printing (same i/f is used))
#define CFLGSPPOOLNO   0x20000000 // 1 -> don't use the spooler (this is to provide previous functionality,
                                   //no extra overheads).
```

1. __CImp IMAGE INTERFACE STRUCTURE

This is the interface structure that passes status and image data to the customer supplied callback that needs to be called when a scanned image is available. NONE of these fields can be modified by the callback. The Scanner produces device dependent bitmaps. The bPlanes member is always the number of bits per colour

```
typedef struct __CImp    // Image interface structure for callback
{
    BYTE    *pIm;        // pointer to image (packed bit array, 32bit aligned)
    BYTE    *pCoef;      // correction coefficients currently in use
    BYTE    *pGamma;     // gamma correction table currently in use
    BYTE    *pTDens;     // bit count table for BYTE (can be used for fast counting of bits within a byte)
    BYTE    *pTSwap;     // bit reverse table for BYTE (can be used for fast bit reverse, mirroring b/w images)
#ifdef VB
    HBITMAP WinBITMAP;    // Windows Bitmap handle
    HDC      WinHCD;      // Windows device context
#endif
```

```

#endif
long    lTime;        // use to pass the time
WORD    wWidBy;       // image width in bytes
WORD    wWid;         // image width in pixels
WORD    wHei;         // image height in pixels
WORD    wNoClr;       // no of colors (1 = monochrome, 3= rgb)
BYTE    bPlanes;      // no of planes (1,4,5,8 supported) (pixel depth for each color)
BYTE    bType;        // type of data (colors present)
                        // Bits Function
                        // 0 -> red channel (default monochrome)
                        // 1 -> green channel
                        // 2 -> blue channel
                        // 3 -> Gray
// #define CV_COLOURS_MASK    0x07    // bits relevent for indicating colour mode
// BYTE    bColours;
BYTE    bRet;        // return value is now supported (not used reserved)
}
_CImg;

```

2. __CCnf CONFIGURATION INTERFACE STRUCTURE

This interface structure is used to pass back configuration information during the CV_CAL_GET_ALL call. which is normally reserved for EngBench use. The dll remembers the last set configuration in the registry across loads and this is a convenient way to recover this information

DEVICE codes: these must match the scanner being used
please note some devices are in dots per inch (dpi) and some are in dots per mm (dpm)

```

#define EDEV200A6_864    0    // On special request
#define EDEV200A4_1728   1    // CST5000-2 full width scanning A4 8 dots per mm (dpm)
#define EDEV300A6_1248   2    // On special request
#define EDEV300A4_2592   3    // CST5000-3 full width scanning A4 300 dots per inch (dpi)
#define EDEV400B7_1408   4    // On special request
#define EDEV400A4_3456   5    // On special request A4 16 d
#define EDEV200A4_1344   6    // CST5000-2 scanning only the first 1344 pixels (for faster scanning on smaller slot)
#define EDEV200A4_100v   7    // CST5000-2 (default) this is A4 8 dpm horizontal * 4 dpm vertical (faster scanning)
#define EDEV200A4_960    8    // CST5000-2 scanning only the first 960 pixels (for faster scanning on smaller slot)
#define EDEV600A4_5184   9    // TPH added 091012

```

MODE selection: essentially scanning at native CIS resolution and 1/2 the native resolution is supported by the hardware in the 3 bpp modes of 1 4 & 8

```

#define EACRmFULL256    0x00    // full cis resolution 256 levels (8bpp)
#define EACRmHALF256    0x01    // half cis resolution 256 levels (8bpp)
#define EACRmFULL2      0x02    // full cis resolution B/W (default) (1bpp)
#define EACRmHALF2      0x03    // half cis resolution B/W (1bpp)
#define EACRmFULL16     0x04    // full cis resolution 16 levels (4bpp)
#define EACRmHALF16     0x05    // half cis resolution 16 levels (4bpp)

```

```

typedef struct __CCnf    // configuration interface structure DLL to APPLICATION
{
    int    Dev;          // as EDEVXXXX above
    int    Mode;         // as EACRmXXXX above

    int    Light;        // the calibrated light level (0..99) for Dev/Mode setting.
    int    Mot;          // the calibrated motor devisor (1..4096) for Dev/Mode. Affects vertical resolution.
    int    BwG;          // in B/W modes the B/W threshold (0..255), in gray scale the gamma correction (1..50)
    int    Current;      // the calibrated motor current (1..70) for Dev/Mode setting.
    BYTE    Planes;      // the calibrated bpp (1,4,8) for Dev/Mode setting.
    BYTE    Factor;      // reserved
    WORD    Flags;       // the low 16 bits of the registry "Flags" key
}
__CCnf;

```

3. __CCaps CAPABILITIES INTERFACE STRUCTURE

Interface structure to support passing back printer and scanner capabilities. An app wishing to get capabilities should first check the DLLVER returned by the ScInit call. If it is >= 21 the call CV_CAL_GET_CAPS is supported, otherwise the app should assume it's a GRAY scale 200dpi Scanner only.

All undefined bits are reserved and 0 meaning not implemented/supported
Some of these caps cannot be automatically detected on some hardware. In that case this may be set during installation or assumed the capability exists (but may not work)

```

#define CV_CAP_TPH    0x0001    // scanner has branding printer
#define CV_CAP_COLOUR    0x0002    // scanner has colour capability
#define CV_CAP_LIGHT    0x0004    // scanner supports 0..255 level light setting (otherwise 0..99)

```

```

typedef struct __CCaps    // interface structure DLL to APPLI
{
    DWORD    cap_version;    // scanner firmware version
    DWORD    cap_hw;
    DWORD    cap_sw;
    DWORD    cap_devices;    // bit mask of devices supported (device 0 is bit 0 and so on)
    DWORD    cap_modes;      // bit mask of modes supported (mode 0 is bit 0 and so on)
    DWORD    cap_reserved[4]; // future capabiliyt information
}
__CCaps;

```

4. __CCaps Maintenance Information Interface Structure

This interface structure is used to pass back maintenance information during the CV_CAL_MAINT call. This information can be used to calculate scanning performance parameters.

NOTE: The possible scanner speed is determined by a number of factors;

- CIS frequency,
- data transfer rate to host and
- mechanism feed limit.

Using the configuration registry settings and a machine speed measurement during startup the optimum speed is always used. Note, however these values are cached in the (cnf.dat and Cfg*.dat files). For this reason these files should never be copied from machine to machine, and if for some reason the speed of the machine is changed these files should be deleted and recreated by a calibration.

```
typedef struct __CMaint      // interface structure DLL to APPLI
{
    // maintenance functions

    DWORD   STime;           // time of scan up to the stop of acquisition
    BYTE    CisClkH;         // reserved
    BYTE    CisClkL;         // the low CIS clock time (CIS frequency can be calculated)
}
_CMaint;
```

CALLBACKS

• FINSCANDIBFCT

This is the user defined asynchronous scan complete processing procedure. It is called in the context of the scanning thread after a scan has completed. In this limited version you can only rely on the structure members to be valid;

```
pIm,
wWidBy,
wWid,
wHei and
bPlanes
```

This callback should not return until it has disabled scanning or has finished with the RAW image. The image bit data is packed in one contiguous block line by line. Lines are always 32bit aligned. b/w images are packed 8 bits per byte, 4 bit/pix images are packed 2 bits per byte, 8 bit/pix images are one bit per byte. All these formats are directly compatible with windows DIB format BOTTOM UP image.

When in this callback the scanner is disabled and if required the scanner must be stopped with a special parameter when calling CV_CAL_STOPSCAN. This is only required when image processing is done outside the callback, see BmpEx examples for further details.
Not required for versions switching v2.0.0.0 (051109) in CyberSc(051109).zip and above.
Not required for versions ECP only V1.0.0.23 (10/11/2005) in ECP_CyberSc1.23(051110).zip and above.

```
typedef void (*CALLBACK*/ CALLCONV *FINSCANDIBFCT)(const _CImg *);
```

• PART_SCAN_CALLBACK

Callback method used to return scan data while the scan is still ongoing. This is used with the ScSetupPartScan method. The _CImg parameter is the same as used in the normal FINSCANDIBFCT callback. If the function returns true then the scan will continue, if false the scan will stop and the paper will be rejected.

```
typedef BOOL(*CALLBACK*/ CALLCONV *PART_SCAN_CALLBACK)(const _CImg *);
```

• FINSCANFCT

Reserved

```
typedef void (*CALLBACK*/ CALLCONV *FINSCANFCT)(HBITMAP);
```

• ERRORFCT

This is the user defined error callback procedure called asynchronously in the context of one of the internal DLL threads (printer status APC, scanner or Image print) or application threads making DLL function calls to:

- 1) Signal an error condition.
- 2) To signal the removal of an error condition.
- 3) To signal the reception of a barcode. (dwType -> const char*)
- 4) To signal a debug message. (dwType -> const char*)

The second (dwType) parameter must be interpreted on the basis of the first parameter (dwError) because for CV_BARCODE, CV_DEBUG it is actually a pointer to a string. All pointers are to memory within the DLL and should not be modified.

The error codes are defined below. If the application does not want to treat an error, then simply ignore the error and resume execution.

The callback should return promptly, (e.g. not to call a MessageBox function or such like)

```
typedef void (*CALLBACK*/ CALLCONV *ERRORFCT)(const DWORD, const uintptr_t);
```

SCAN AND CONFIG FUNCTIONS

• ScSConfigPath()

Set the directory to look for the scanner config files. This will only have an effect if it's called before ScInit. The directory must include the trailing "\". At the moment this will only work on the Colour USB scanner.

```
input params:
    configDir    -> pointer to the first character of a zero terminated string
-----
PORTEE_DLLCV_LB void CALLCONV ScSConfigPath(const char* configDir);
-----
```

• ScPOff()

ScPOff switches the interface from printer mode to scanner mode. It is used before starting the scanner. Use ScPOn() to switch the interface to print mode (see below)

```
input params: None
-----
PORTEE_DLLCV_LB void CALLCONV ScPOff(void);
-----
```

• ScInit()

ScInit is used to initialize the scanner and register the callback functions which will signal important events. It returns version information if successful and the scanner is initialized but disabled. Otherwise it returns the value 0 (false).

This function connects to the underlying driver, initializes the scanning including the allocation of the image buffer and the loading of the current calibration file, initializes the printing system including the loading of fonts, and defines the end of scan and asynchronous error procedures. Although not enforced, this call must be made before any other functions in the DLL.

An ScInit call can be re-made at any time to re-initialize the system however routine re-initialization is not recommended. Note that ScInit calls may be necessary if there is a problem, for example after a cable has been temporarily disconnected or the scanner has been powered down, or if a fatal error is reported.

It is important to note that it is possible to receive an error callback even before this function returns.

The image buffer size is read from the registry key MaxImageSize and is specified in bytes. In printer only configurations this should be set to a small (4096 bytes) value, so as to not to waste memory.

The current calibration file is determined from registry limits (MaxCISFreq, MaxMotor) and the machine speed and the current settings (Device, Mode).

The default font file is hardwired to "Font.asc" in the current directory.

```
input params:
    fpFinScanning -> pointer to a user defined scanner callback function. (see above)
    fpError       -> pointer to a user defined scanner callback function. (see above)
```

```
returns:
    0 -> failed (no version information is returned).
    !0 -> succeeded. The value returned is a packed combination of byte version numbers, LSB to MSB order:
```

```
    LSB          -> hardware version number (bottom nibble is always 1).
    Next Byte    -> Device driver version number
    Next Byte    -> CyberSc version number (this dll)
    MSB          -> always 0, reserved for application to fill in its version number.
```

Driver version numbers follow the following convention (if loaded)
USB hardware is used first whenever available, in this case the ECP driver won't be reported.

```
    0x00 -> 0x3F counting up   VXD ECP   current 0x10
    0x7F -> 0x40 counting down VXD ISA   current 0x7F
    0x80 -> 0xBF counting up   SYS ECP   current 0x81
    0xFF -> 0xC0 counting down SYS ISA   current 0xF4
```

E.g. If the value 1344289 is returned then in HEX this is 148321 which means left to right
(14) CyberSc version 20,
(83) ECP driver Cy.sys version 83,
(2) ECP hardware version 2,
(1) init succeeded.

```
-----
PORTEE_DLLCV_LB int CALLCONV ScInit(FARPROC fpFinScanning,FARPROC fpError);
-----
```

• ----- Enumerations for ScSCalibrate() -----

Calibrate step enumeration (for ScSCalibrate function). Param should be 0 unless stated otherwise. Unless otherwise stated the function returns boolean where true -> success, false -> failure.

stops the internal dll scanner thread (created during ScInit) from polling the scanner detector sensors or accessing the hardware. This is used for example when the app has to use the printer or does not want allow scans again e.g. during a calibration. This function may block and a scan callback may occur while waiting in this function. The function always succeeds. Also switches off the scanner leds. Use param = 0x55 if calling from inside the scanner callback.

```
#define CV_CAL_STOPSCAN          0
```

Runs the calibration function. When called it is assumed the app has arranged that the white calibration paper is already seated in the scanner entry slot. If paper is not detected within a timeout the function fails. After the function returns the new calibration is in effect if successful. The return value is boolean, false if calibration failed.

```
#define CV_CAL_CALIBRATE        1
```

Performs the same calibration function as CV_CAL_CALIBRATE but deletes the existing calibration file first. This must be called when calibrating a new scanner on a system which previously had a scanner

```
#define CV_CAL_RECALIBRATE      47
```

Sets the black and white threshold specified in Param (0..255) if in B/W resolution. Else sets the GAMMA (usefull values 1.100). This functionality is now supereceded by the CV_CAL_SET_GAMMA method that can change the gamma also for BW modes also.

```
#define CV_CAL_SET_BW          2
```

Gets the current black and white threshold, value 0..255 returned if in B/W resolution. Else returns the GAMMA setting. This functionality is now supereceded by the CV_CAL_GET_GAMMA method.

```
#define CV_CAL_GET_BW          3
```

Sets the current bpp mode specified in Param, value 0..5. For further details see Mode field of struct __CCnf
versions supporting colour (21 and +), bits 7 .. 5 (mask 0xe0) now set the colour mode as follows
0 -> red channel (default monochrome)
1 -> green channel
2 -> blue channel
3 -> RGB
4 -> monochrome

```
#define CV_CAL_SET_RESO        4
```

Gets the current bpp (bits per pixel) mode, value 0..5 returned.
versions supporting colour (21 and +), bits 7 .. 5 (mask 0xe0) now set the colour mode as follows
0 -> red channel (default monochrome)
1 -> green channel
2 -> blue channel
3 -> RGB
4 -> monochrome`

```
#define CV_CAL_GET_RESO        5
```

Starts the scanner polling if the scannet is connected and initilised properly.

```
#define CV_CAL_STARTSCAN       6
```

Gets the current light level for this configuration (Mode / Dev), value returned.

```
#define CV_CAL_GET_LIGHT       7
```

Sets the current light level for this configuration (Mode / Dev). This value is set by the calibration but this function may be used to tweak it. The light level is specified in Param 0..99. If a value > 99 is specified 99 is used without error.

added functionality since V2.0.0.1 for colour support. The MSB byte is now defined as follows

- 0x00 -> previous functionality the Light is adjusted globally for all colours
- 0x01 -> changes red light level - byte
- 0x02 -> changes green light level - byte
- 0x03 -> changes blue light level - byte
- 0x04 -> changes red offset level - word
- 0x05 -> changes green offset level - word
- 0x06 -> changes blue offset level - word
- 0x07 -> changes black level - word

```
#define CV_CAL_SET_LIGHT       8
```

Gets the current CIS device, value 0..8 returned.

```
#define CV_CAL_GET_DEV         9
```

Sets the current CIS device specified in Param, value 0..8. For further details see Dev field of struct __CCnf. Only devices supported by the scanner should be used or incorrect scanning will result.

```
#define CV_CAL_SET_DEV          10
```

This function can be used to make the scanner start if it does not want to start naturally because the leading edge is too rough or dark. The scanner must still sense paper to start, only the paper edge start condition does not need to be met.

```
#define CV_CAL_FORCESCAN       11
```

Retrieves information in the struct __CCnf structure. Reserved for EngBench use.

```
#define CV_CAL_GET_ALL         12
```

Sets the motor scaling coefficient. This value is set by the calibration but this function may be used to tweak it. Advanced use, use at your own risk.

```
#define CV_CAL_SET_MOT         13
```

Reserved for EngBench use.

```
#define CV_CAL_SET_USE         14
```

Sets the motor drive current. This value is set by the calibration but this function may be used to tweak it. Advanced use, use at your own risk. Note setting the current unnecessarily large may actually reduce the motor torque and overheat the motor at large operating duty.

```
#define CV_CAL_SET_CURRENT     15
```

Reserved for EngBench use.

```
#define CV_CAL_MOT             16
```

```
#define CV_CAL_SERVICE         17
```

```
#define CV_CAL_MAINT           18
```

Used to set the interrupt margin in microseconds. This can be increased from the default if there are lost scan lines occurring too often, but will reduce scan speed. Only helpful for 8bit bpp modes. Advanced use, use at your own risk.

```
#define CV_CAL_MARGIN          19
```

Superseded - do not use for new apps.

```
#define CV_CAL_U_STOPSCAN      20
```

Gets the current interrupt margin, returned as a value (in microseconds added per scan line)

```
#define CV_CAL_GET_MARGIN      21 // this will be init time parameters
```

Reserved for EngBench use.

```
#define CV_CAL_GET_PTR         22
```

Toggles the jam detection setting. By default when the driver is opened the jam is always on. NOTE: with paper jam detection turned off this may destroy the original and/or result in more AI failures as images can be more clipped. Advanced use, use at your own risk.

```
#define CV_CAL_TOGGLE_JAM      23
```

Gets the current paper detector reading, returned as a bit mask (set -> detector active). masks for paper detection returned by CV_CAL_PAPER.

```
//#define EmPRESENT             0x1e    // presence detectors ( defined in e.h )  
//#define EmJAM                 0x21    // side jammed detectors ( defined in e.h )  
//#define EmINTERRUPT           0x800   // printer interrupt ( defined in e.h )  
#define CV_CAL_PAPER            24
```

This function is called to gracefully close down all resources allocated by the dll. call this method while callbacks are still correctly processed, ...

```
#define CV_CAL_TERM            25
```

These functions are currently private - They will be documented as required.

```
#define CV_CAL_PRN_RAW          26 // 010130 new, (paper feed with cap open)  
#define CV_CAL_NOPRN           27 // 010221 new, (use 3rd party printer, in print mode)  
#define CV_CAL_PSTATUS          28 // 010306 new, (test only, request status, in print mode)  
  
#define CV_CAL_SET_GAMMA        29 // 031002 new, (allows setting of gamma in B/W modes)  
#define CV_CAL_GET_GAMMA        30 // 031002 new,
```

Sets the background file (.prn including the extension). P0 is a char * to the filename,

if NULL cancels the background. Is automatically canceled with a ScPEnd that actually does a cut.
 If the filename begins with an 's' only a single run of the file is made (when the end of file is reached background printing stops) otherwise the file rolls over to the beginning. In this case the file should be a multiple of the text line (24 scans) or small gaps may result during rollover.
 The background is merged with the foreground with the exclusive or function (invert)
 This is useful for printing samples or discrete logo/advert/instructions, ...
 The background occurs only during text printing, suppressed during image and barcode printing
 NOTE: Due to the background implementation it is best if the lines is a multiple of 24 lines - 1 byte.
 Available from version 2.0.0.3

```
#define CV_CAL_SET_BACKGROUND      31
```

Whenever a USB printer is connected returns the USB firmware version as a WORD.
 If no printer or an ECP printer is connected returns 0.

NOTE: the printer controller firmware version is different, defined for all printer types, and requested with the ScPPower call. Available from version 2.0.0.9

```
#define CV_CAL_GETUSBPRINTERVER    32
```

This allows printer detection without actually doing any print calls and thus triggering error callbacks.
 Available from version 2.0.0.10. It returns ;

```
0 -> No Printer
1 -> USB full speed (XAC or original Epson),
2 -> ECP
3 -> USB high speed Seiko,
4 -> USB high speed Epson
```

```
#define CV_CAL_GETPRINTERTYPE      33
```

Set printer speed
 0 -> Fast speed (default),
 1 -> Medium Speed, 2 -> Slow Speed

```
#define CV_CAL_SETPRINTERSPEED     34
```

```
#define CV_CAL_GETSPOOLEDBYTES     35 // returns the spooled bytes in USB spooler
#define CV_CAL_WAITSPPOOL          36 // blocks until the USB spooler is empty (e.g. in situations
// where next step is permitted before all data is spooled use this
// before the next operation uses the printer [e.g. scan completion])
```

Scanner status masks used by CV_CAL_STATUS_ACTIVITY

```
#define SS_LODAED      0x01 // scanner driver found and loaded
#define SS_THREAD      0x02 // scanner thread started and hasn't terminated
#define SS_BUSY        0x04 // scanner is busy (scanner thread not idle sleeping)
#define SS_STOPPED     0x08 // scanner polling is disabled (note: it is still possible to get BUSY
// but the hardware is not used)
#define SS_SCANNING     0x10 // scanner is currently in the process of scanning
#define SS_CALLBACK     0x20 // scanner thread is in user scan callback
#define SS_OSCILLO      0x40 // scanner is in oscilloscope display mode (in this mode the POLL counter
// increments much faster)
#define SS_PAPER        0x00ff00 // paper detectors at last poll
#define SS_POLL         0xff0000 // mask of 8 bit scanner thread activity counter - incremented each poll
// (nominally every 150ms). During a scan this counter will not increment at this rate)

#define CV_CAL_STATUS_ACTIVITY 37 // Scanner health check - returns various scanner status info in a 32 bit integer
// via SS_XXXX flags ( E.g. last poll paper status and poll activity counter etc)
```

Feed scanner paper by N Lines.

Parameters
 bit[15..0] -> Number of lines N to feed
 bit 16 -> Direction: 0 Forward, 1 Reverse

```
#define CV_CAL_TPHFEED 38 //used to feed scanner paper by N lines
```

Control of TPH motor.

Parameters
 bit[15..0] -> Motor step duration in uSecond
 bit 16 -> Direction: 0 Forward, 1 Reverse
 bit 17 -> OnOff: 0 Off, 1 On

```
#define CV_CAL_TPHMOTOR 39 //used to control TPH motor
```

Control of TPH motor speed (only useful when motor is running)

Parameters
 bit[15..0] -> Motor step duration in uSecond

```
#define CV_CAL_TPHSPEED 40 //used to set TPH motor speed
```

Control of TPH motor timeout

Parameters
 bit[31..16] -> Timeout when Motor is moving forward (in hundreds of uSec)
 value 0 forces the default firmware value to be used

 bit[15..0] -> Timeout when Motor is moving reverse (in hundreds of uSec)
 value 0 forces the default firmware value to be used

```

#define CV_CAL_TPHTIMEOUT 40 //used to set TPH motor timeout

Print image with TPH
Parameters
    bit[15..0]    -> First line to print
    bit[31..16]   -> Last line to print

#define CV_CAL_TPHPRINT 41 //used to print image with TPH

Load image to print with TPH
Parameter to be cast to PBYTE (pointer to buffer)
    First 2 bytes (MSB first) indicate number of the first line.
    Next 2 bytes (MSB first) indicate number N of following lines (56 byte each) in the buffer.
    Following bytes are the image (N lines of 56 bytes)

#define CV_CAL_TPHLOAD 42 //used to load image to print with TPH

TPH Status
returns TPH status

#define CV_CAL_TPHSTS 43 //used to retrieve TPH status

Get connected hardware and software information
CyberSc behaves in legacy mode (all images will be monochrome (only 1 colour plane)) independent of the
saved settings in the registry until this function (or CV_CAL_GET_COLOURS / CV_CAL_SET_COLOURS) is called
NOTE - Apps can set operating parameters but should always read back the actually DLL used parameters.
The dll should not accept or set any mode it cannot currently handle.
NOTE: the _CIImg structure always contains actual image bpp for any returned image

#define CV_CAL_GET_CAPS 44

Get extended light setting information
Actively used to show this is in use (i.e. colourscanner)
typedef struct __CLightEx // interface structure DLL to APPLI
{
    BYTE    r_light, g_light, b_light, active;
    WORD    r_offset, g_offset, b_offset;
    WORD    black;
}
__CLightEx;

#define CV_CAL_GET_LIGHT_EX 45 // in ScSCalibrate()'s Param pass the address of a _CLightEx struct
// the values will be set via the struct's members

Turns starting scanning on as soon as paper is detected and keeps it on.
The difference between this and CV_CAL_FORCESCAN is this isn't automatically disabled after a scan.
    Param = 1 enable force scan
    Param = 0 disable force scan.
#define CV_CAL_FORCESCAN_STAY 46

```

• ----- End of Enumerations for ScSCalibrate() -----

• ScSCalibrate()

ScSCalibrate is used to perform simple miscellaneous functions, including calibration, maintenance, parameter setting / recovery. This function provides extensible functionality without having to change the interface. Step is one of the CV_CAL... values defined above and defines the meaning of the parameter and the return value. If the current version doesn't support the function requested returns 0 -> failure.

```

input params:
    Step    -> CV_CAL... defined above
    Param   -> see CV_CAL definiton
Returns:
    -> see CV_CAL definiton */

```

```

-----
PORTEE_DLLCV_LB int CALLCONV ScSCalibrate(int Step, uintptr_t Param);
-----

```

• ScSetupPartScan()

ScSetupPartScan is used to register a callback that will get parts of the scanned image as it's scanned. When scanning an larger piece of paper this will let you get some of the image faster than waiting for the whole scan to complete. The method registered with ScInit will still be called at the end of the scan, so ScInit should always have been called before ScSetupPartScan.

"steps" is a pointer to an array of integers. Each integer is when to return after that many lines (in pixels) have been read. So [100, 300, 200] will return a 100 line image, a 400 line image and a 600 line image.

"numSteps" is the number of entries in steps.

```
"fpScanning" is the callback function. It's of type FINS CANDIBFCT. (The same as ScInit). */
-----
PORTEE_DLLCV_LB int CALLCONV ScSetupPartScan(int* steps, int numSteps, FARPROC fpScanning);
-----
```

• ScRejectPaper()

If after a scan there is still paper left in the scanner call this function to reject the paper. If the scanner doesn't detect paper this function will do nothing. */

```
-----
PORTEE_DLLCV_LB void CALLCONV ScRejectPaper();
-----
```

PRINT FUNCTIONS

• ScPOn()

ScPOn switches the interface to print mode. It is assumed the scanner is stopped. This function also clears any non persistent error on the printer so further printing is attempted. Use this function to begin printing a receipt. Use ScPOff() to switch the interface to scan mode (see above). NOTE: This function does not return a completion status. You can call ScPError to recover this. */

```
-----
PORTEE_DLLCV_LB void CALLCONV ScPOn(void);
-----
```

• ScPStatus()

ScPStatus reports the current printer status (the last status mask received from the printer) The Status byte, is always the first byte of a reverse request:

```
mask 0x01 -> paper out error
mask 0x02 -> head unloaded
mask 0x04 -> head temperature error
mask 0x08 -> cover open
mask 0x10 -> cut timeout
mask 0x20 -> head voltage error (##W same bit)
mask 0x20 -> data stream error (##W same bit)
mask 0x40 -> nearly out of paper
mask 0x80 -> cut toggles whenever enabled and a cut
```

completes (used to support print checkpoints, or synchronous printing) */

```
-----
PORTEE_DLLCV_LB int CALLCONV ScPStatus(void);
-----
```

• ScPError()

ScPError reports the result of the last Write to the printer

```
returns:
false-> printing failed.
true -> printing was successfully queued.
```

NOTES: It is common practice not to test the return of each call to a print function. Once the printing system detects an error new calls return immediately and do not attempt the function. So it is permissible to use this function to test the overall print result after a series of print calls if it is not important to know exactly where the print failed (this cannot be determined with any precision anyway because of spooling). It follows that special action is required to reset an error and re-synchronize the print stream once an error occurs. This is done by a ScPOff/ScPOn cycle. */

```
-----
PORTEE_DLLCV_LB BOOL CALLCONV ScPError(void);
-----
```

• ScPString()

ScPString prints the null terminated string on the printer in a 16 * 24 fixed font defined in the default "Font.asc" or a font loaded with ScPFont(char *fName);. This file is user editable to define the typeface. Font cells stack together so continuous graphics are easily implemented. Line feeds can be inserted within the string to print multiple lines in the same call. Lines longer then the print width (36 characters) are wrapped onto the next line. This function is only allowed in print mode even though not enforced.

```
input params:
Str -> Pointer to a NULL terminated string to print
returns:
false-> printing failed.
true -> printing was successfully queued.
```

The function recognises control codes in the string to set double width or double height are as follows:

```
^1 -> \001 -> double width & double height
^2 -> \002 -> double width only
^3 -> \003 -> double height only
^4 -> \004 -> reverse B/W
^5 -> \005 -> simple bold. toggle for the current line / character position.
```

```

^6 -> \006 -> first in string - triggers interpretation as text descriptors (new 091002)
^9 -> \009 -> text centering. It must be the first control code on the line, it's a toggle that spans lines
      but after a cut it's reset to no center. Wide text modes are supported as well.

```

These codes can occur multiple times on a line. Each use toggles the state. For the height, the state in effect at the end of the line determines if it is printed double height or not. When using double width, (this may be only a few characters in a line), it is the users responsibility to ensure the whole line fits in the print width. Anything outside is clipped (not wrapped as with normal text).

```

eg  ScPString("The word is \002wide\002 is double width");
     ScPSting("Its a wonderful day\003") and
     ScPSting("Its a won\003derfull day") and
     ScPSting("\003Its a \003wonderfull day\003") are equivalent and will print in double height

```

Note - \n's are recognized and cause a line feed, however except lone \n on a line are dropped. If you need to insert a blank line, include at least 1 space character. If a line is longer then 36 printable characters the line is automatically wrapped for as many lines as needed.

All fields (including the last) are delimited by a reserved character "^"

Text representation of descriptors.

This is supported from version 2.0.0.10

All co-ordinates are in printer pixels with X going across the paper left to right (0..575), and Y going along the print axis top (first printed) to bottom (last printed) (0..n)

The first field is int = <total size of block> then up to 32, 7 field descriptors specifying print commands as per descriptor doc.

```

int      Height;          // font height (in pixels) unless stated otherwise
int      Width;           // font width (in pixels) unless stated otherwise
int      Flags;           // font flags (see below) unless stated otherwise
char     *Font;           // font name or special processing trigger
int      ofsX,ofsY;       // x (along line) and y (vertical from top) offsets within the block
char     *Str;            // string to render unless stated otherwise

```

For full details of the triggers and Flags, see the ScPFmtBlock() & ScPFmtString() functions below.

For "int" fields all formats are supported, (e.g. 0x for hex).

Additionally "Flags" can be prefixed by 'b' or 'c' or 'd' as a shortcut to rotate text 90, 180 or 270 degrees respectively. Finer rotation can also be used but the number must be worked out in this case. */

```

-----
PORTEE_DLLCV_LB  BOOL CALLCONV ScPString(const char *Str);
-----

```

• ScPBmp()

ScPBmp prints a monochrome only bitmap image described by a Windows defined BITMAP structure.

This function can be called when the DLL is in print mode, along with any other print function in any order.

The bit image is a top down image (as the image passed to the callback) pointed to by the field "bmBits". The size is determined by the fields "bmHeight" and "bmWidthBytes". The bit image should be 0 filled if it is not a BYTE multiple of the line size as the pixel width is not used, full width of the image will be used.

All other fields in the BITMAP structure are ignored except the bmType field. The usual WINDOWS value of 0 for the bmType field specifies a normal top down image, and a value of 1 specifies a bottom up image (non WINDOWS). For this reason the bmType field **MUST** be initialized in calls to ScPBmp (normally to 0). The printed image width is scaled (preserving the aspect ratio) to fit the printer width of 72 bytes. Images of 72 bytes (576bits, native width of printer) wide are more efficient to print.

This function operates asynchronously so processing of the transaction can continue as the image is printed (which usually is large and takes a long time), however the source image is not required after this function returns. The application is free to continue processing while the printing takes place but must call ScPSynch before calling any other print function including Another ScPBmp and ScPOff. ScPSynch will BLOCK while spooling is taking place. The return status corresponds only up to the processing stage. BITMAPS with a null bmBits are ignored without generating an error. Printing failures will be reported by callbacks.

This function limits the print density to 75% by filtering consecutive lines over 75% with a rotating 50% mask. As such it is suitable for printing scanned images. For predefined image decorations it is much more convenient and efficient to convert to .prn (with density limited to 75% or 50% (preferred)) files and print with the ScPImage function.

Here it is assumed it is a black on white bitmap -> pix = 0 => black, pix = 1 => white

```

input params:
  pBmp -> pointer to a windows monochrome BITMAP structure.
returns:
  false-> printing failed.
  true -> printing was successfully queued.

```

NOTES: If bmType is 0 then it is a top down image and will print correctly on the printer, a bottom up will print mirrored in the vertical direction. If bmType is 1 then it is a bottom up image and To use windows functions with the bitmap you may have to set bmType back to 0 after the image print. Images that are scaled up (original width is < 72 bytes) are limited to 2000 lines scaled up or the image will be truncated. Images >= 72 bytes wide the height is limited only by memory. If the intermediate buffer required for the resulting print stream cannot be allocated or the bitmap bits (bmBits) pointer is NULL the image is not printed, no error given due to the asynch nature. */

```
-----
PORTEE_DLLCV_LB  BOOL CALLCONV ScPBmp(const BITMAP *pBmp);
-----
```

• ScPImage()

ScPImage prints a specially formatted print file (usual extension .prn) specified by a full or partial path. If the file cannot be opened, the function does nothing without returning an error. Printer prints black -> pix = 1 => black, pix = 0 => white

input params:
FullFileName -> the path to the file

NOTE: PRN files are essentially a complete byte stream to the printer (a sequence of print commands). A utility exists (bmp2prn) to convert bmp images to the required format. Restrictions apply. A utility also exists to view existing prn files. */

```
-----
PORTEE_DLLCV_LB  void CALLCONV ScPImage(const char *FullFileName);
-----
```

• ScPSynch()

ScPSynch is used to wait for the completion of a ScPBmp print. As these can be quite large (and so take a long time to print) the printing is performed asynchronously. i.e. the ScPBmp function returns as soon as the image is no longer needed. The app can use this thread to continue other processing. When the app needs to use another CyberSc function it uses this function to wait until it is safe to do so.

returns:
none -> this function always succeeds (the ScPBmp print completes) waits indefinitely until printing finishes or times out. */

```
-----
PORTEE_DLLCV_LB  void CALLCONV ScPSynch(void);
-----
```

• ScPPower()

ScPPower is used to set the darkness of the printout. The lowest satisfactory power setting should always be used to maximise the print head life. This may be required to adjust for paper sensitivity. This is an advanced use function, and is not required for basic functionality. The printer defaults to a given power setting whenever it is reset.

input params:
Pow -> the power setting as follows
0xF0 -> highest power
0xE0
0xD0
0xC0
0xB0
0xA0
0x90
0x80 -> lowest power
0 -> request firmware version
1 -> setting from registry (default 0xD0)

Returns:
none -> Use ScPError to determine if the function successfully spooled the command. */

```
-----
PORTEE_DLLCV_LB  void CALLCONV ScPPower(BYTE Pow);
-----
```

• ScPFeed()

ScPFeed is used to feed paper by scan line.

input params:
Scans -> 1..255, 1 scan line corresponds to 1/8th of a millimeter.

Returns:
none -> Use ScPError to determine if the function successfully spooled the command. */

```
-----
PORTEE_DLLCV_LB  void CALLCONV ScPFeed(BYTE Scans);
-----
```

• ScPCut()

ScPCut cuts the paper according to the cut type. No Logo printing or spacing off the head cut position past the last printed scan line takes place. If this is required it is better to call ScPEnd.

input params:
Type -> cut type as follows.
0 -> 1/2 cut
1 -> full cut
2 -> claitons (no cut) - can be used for synchronising to a particular point on the printout to be sure it has printed without error. (advanced use)
4 -> home cutter (only moves if cutter not already homed)

NOTE: This function does not return a completion status. You can call ScPError() to recover this. */

```
-----
PORTEE_DLLCV_LB  void CALLCONV ScPCut(BYTE Type);
-----
```

• ScPInvalidate()

Reserved do not use */

```
PORTEE_DLLCV_LB void CALLCONV ScPInvalidate(void);
```

• SCPEnd()

ScPEnd is used to print an optional centered interleaved 2 of 5 barcode and to optionally feed out the paper past the last printing line and cut the paper. In this process the otherwise wasted paper is printed with a custom logo which serves as a header for the next printing.

input params:

Code -> pointer to string containing the I2of5 barcode string. As such only 0..9 digits are allowed. A max of about 24 digits will fit. If an odd length is supplied the barcode is padded with a trailing 0. If Code is 0 length or NULL no barcode is printed. If invalid characters are in the code string an invalid barcode is generated.

Cut -> specifies the cut type. This is the cut type as for ScPCut shifted one to the left. If the LSB bit is 0 no cut is performed.

0 -> no cut (only barcode if specified is printed)

1 -> 1/2 cut

3 -> full cut

5 -> claitons (no cut) - can be used for synchronising to a particular point on the printout to be sure it has printed without error. (advanced use)

9 -> home cutter (only moves if cutter not already homed)

returns:

false-> printing failed.

true -> printing was successfully queued.

```
PORTEE_DLLCV_LB BOOL CALLCONV ScPEnd(const char *Code, int Cut);
```

• ScPScan()

ScPScan prints exactly 1 scan line of data defined as binary data. This function is used mainly for testing. If more lines are to be printed it is better to use one of the image printing functions.

input params:

Buf -> pointer to a 73 BYTE buffer. The first byte must be 0. the other 72 bytes define the 576 pixels of the print head in msb to lsb left to right order.

NOTE: This function does not return a completion status. You can call ScPError to recover this. */

```
PORTEE_DLLCV_LB void CALLCONV ScPScan(const char *Buf);
```

• ScPFont()

This loads the specified .asc font file.

The failure to load the default font "Font.asc" is no longer a fatal initialisation error, and is kept for compatibility. Only one font can be active at a time.

The font is replaced so the font in effect is always the one defined by the last call to this function or "Font.asc" if this was never called

```
PORTEE_DLLCV_LB BOOL CALLCONV ScPFont(const char *fName);
```

• ScPFmtString()

ScPFmtString prints 1 line (or 1 block) of Windows rendered text on the printer. The area printed is exactly Height times the full width of the printer. If the text does not fit into this area it is clipped.

No control codes or line feeds are recognised. 1 line is emitted for each call. Proportional fonts can be used. For further details on how the font is selected see the CreateFont() Windows function.

flags used by ScPFmtString():

```
#define DLLCV_FMT_BOLD      0x03ff    // mask that affects text boldness
#define DLLCV_FMT_ITALIC    0x8000    // selects italics
#define DLLCV_FMT_UNDERLINE 0x4000    // selects underline
#define DLLCV_FMT_STRIKEOUT 0x2000    // selects strikeout
#define DLLCV_FMT_OPAQUE    0x1000    // selects opaque text drawing (by default transparent)
```

input params:

Height -> Height in printer pixels to print. (must be >= 8) This also determines the height of the text allowing for margin / descenders

For barcode specifies element width (or barcode density). useful range 1..5.

Width -> Average width of character. This can be used to change the aspect ratio of the character set. If 0 the default aspect ratio for the font is used. This selection is very approximate subject to available windows scaling.

Flags -> Packed flags that select the variants of the font as follows: Use the OR operator to combine features.

DLLCV_FMT_BOLD	0x03ff	mask that selects boldness
Windows defines the following use only 1 of these at a time:		
FW_DONTCARE	0	
FW_THIN	100	
FW_EXTRALIGHT	200	
FW_ULTRALIGHT	200	
FW_LIGHT	300	
FW_NORMAL	400	
FW_REGULAR	400	
FW_MEDIUM	500	
FW_SEMIBOLD	600	
FW_DEMIBOLD	600	
FW_BOLD	700	
FW_EXTRABOLD	800	
FW_ULTRABOLD	800	
FW_HEAVY	900	
FW_BLACK	900	
DLLCV_FMT_ITALIC	0x8000	selects italics
DLLCV_FMT_UNDERLINE	0x4000	selects underline
DLLCV_FMT_STRIKEOUT	0x2000	selects strikeout
DLLCV_FMT_OPAQUE	0x1000	selects opaque text drawing (by default transparent)

The upper WORD of Flags specifies the rotation of text in tens of degrees counterclockwise.

Font -> The ascii string specifying the font. Abbreviations can be used. The closest available font is selected. Select barcode mode by specifying the string "|".

StartX -> The starting position of the text. For ScPFmtString 16 is added to this value as a left margin and StartX can be negative to reduce this if desired. ScPFmtBlock provides NO default margin (this would hide rotated text, ...)

Str -> The text to print. Whatever does not fit is clipped. No control characters are recognised. For barcode mode is a barcode string containing only digits '0'..'9'. If odd number of digits a 0 is inserted at the end of the barcode. This is standard practice.

Unicode -> if true, Str is interpreted as a WCHAR (16bits) including the terminating NULL, false -> Str is a CHAR (byte) string. Note you will have to cast the WCHAR as a byte string to pass into the (char*) parameter.

Added support for landscape (vertical) interleaved 2 of 5 barcodes.
set "Font" -> "|", "Height" -> element width

returns:
false -> failed probably due to a printer status error.
true -> success line has been rendered and successfully spooled to the printer. */

```
PORTEE_DLLCV_LB BOOL ScPFmtString(int Height, int Width, int Flags, const char *Font, int StartX, const char *Str, BOOL Unicode);
```

• ScPFmtBlock()

You can use the ScPFmtBlock() to print on the same line starting in as many places as you like in as many fonts and sizes as you like. You can do this line by line or a whole block if you like. Font parameters for Flags and font names are as for the ScPFmtString function above. This function can for instance print US style casino tickets.

input params:

BlockHeight -> the vertical height of the block in pixels (8 pre mm) of the block when printed

Unicode -> flag to indicate str (within all descriptors) is in fact unicode (short *)

StrDesc -> pointer to an array of _CTextDesc structures (one per text element to be inserted within the block)

StrDescEntries -> number of descriptors supplied above

returns:
false -> failed probably due to a printer status error.
true -> success line has been rendered and successfully spooled to the printer.

Flags and structures used by ScPFmtBlock();

```
typedef struct _CTextDesc // interface structure DLL to APPLI
{
    int    Height;        // font height (in pixels) unless stated otherwise
    int    Width;         // font width (in pixels) unless stated otherwise
    int    Flags;         // font flags (see above) unless stated otherwise
    char   *Font;         // font name or special processing trigger
    int    ofsX,ofsY;     // x (along line) and y (vertical from top) offsets within the block
    char   *Str;          // string to render unless stated otherwise
}
_CTextDesc;
```

```
#define DLLCV_FMT_ROTATION 0xffff0000 // part of flags that contains the rotation vector
```

Changes as of version v2.0.0.12

1) Transparent background

The text background can be now opaque or transparent (now the default): The background is selected with the DLLCV_FMT_OPAQUE flag.

Barcodes are always rendered with a transparent background (white pixels are untouched).

2) Text rotation:

The upper WORD of Flags specifies the rotation in tenth's of degrees counterclockwise. E.g. 90(270) deg vertical is 900(2700) respectively. Text is rotated about the point ofsX, ofsY, the origin of the text being the lower left corner. Here's an example that prints the text "ROTATED TEXT" every 15 degrees (the second 1/2 with a white background) thus forming a circle ;

```
#define NELEMENTS 24
_CTextDesc d[NELEMENTS];
for( i=0; i<NELEMENTS; i++)
{
    d[i].Height = 40;
    d[i].Width  = 15;
    d[i].Flags  = ((i*15*10)<<16) | (i > NELEMENTS/2 ? DLLCV_FMT_OPAQUE : 0); // every 15 degrees
    d[i].Font   = TESTFONT;
    d[i].ofsX   = 576/2;
    d[i].ofsY   = 576/2;
    d[i].Str    = "ROTATED TEXT";
}
ScPFmtBlock(576,0,d,NELEMENTS);
```

3) The default offset of 16 pixels to ofsX parameter used by ScPFmtString() does **NOT** apply here

4) Landscape interleaved 2 of 5 barcode support (variable scale):

This is via special use of descriptor parameters, Height = narrow element width (pix), Font = "|".

The origin of the barcode is the lower left corner. The barcode length (vertical) is determined by the element parameter and the number of digits in the barcode which should be even (if odd is supplied is padded with 0 at the end)

The descriptor entry looks like this:

```
{ <element_pix>, <width>, 0, "|", <startx>, <starty>, "<barcode>" },

<element_pix> -> int Height; = the number of pixels for a thin bar (5 gives a similar scale to US casino tickets)
<width>       -> int Width = width of the barcode in pixels (horizontal, along print line)
0             -> int Flags = 0 reserved
"|"          -> char *Font = the string "|" (marks this as a barcode request)
<startx>     -> int ofsX = x offset in pixels (horizontal, along print line)
<starty>     -> int ofsY = y offset in pixels
<barcode>    -> char *Str = the barcode string. Only digits '0' .. '9' are allowed.
```

example:

```
{ 5,120, 0, "|", ,230, 360-100, "006024109751317897" },
```

NOTE: if the call is made in unicode mode the barcode string is still specified as a byte character string. If invalid characters occur in the barcode an invalid barcode is generated without error.

5) Support for inserting monochrome images at arbitrary offsets:

This provides support for building arbitrarily complex receipts at run time from predefined images and rendered text with pixel accuracy (e.g. display a card game receipt in a picturesque receipt).

Images of type ".imx" are prepared with a new utility and can be of any size up to the print width of 576 pixels. Parts of the image that fall outside the block are clipped. Images are located and specified on bit boundaries.

The descriptor entry looks like this:

```
{ <reserved>, <reserved>, <flags>, "|i", <startx>, <starty>, "<path>.imx" },

<reserved> -> currently unused but may be used to scale images in the future (0 -> no scaling)
<flags>    -> currently define only the pixel processing mode as follows
0 -> Copy background is clobbered
1 -> Toggle (XOR)
2 -> Set (OR)
"|i"       -> image mode trigger
"<path>.imx" -> the relative or absolute path to the file
```

example:

```
{ 0, 0, 0, "|i", 107, 800, "ds.imx" },
```

All functionality can be intermixed within one descriptor list.

```
PORTEE_DLLCV_LB BOOL ScPFmtBlock(int BlockHeight,BOOL Unicode,const _CTextDesc ** *StrDesc,int StrDescEntries);
```

BLOCKING SYNCHRONOUS FUNCTIONS

- ----- Blocking functions to give the calling code control over the threading model -----

This must be called before any of the other sync functions.

```
PORTEE_DLLCV_LB BOOL ScSyncInit();
```

```

-----
Blocks until a scan finishes, then returns a pointer to a bitmap with the scanned image.
The bitmap must be freed with ScSyncFreeBitmap.
Return NULL if there is an error or a debug message.
-----

```

```

PORTEE_DLLCV_LB LPBITMAPINFO ScSyncScan(BOOL stopOnDebug);
-----

```

```

Free a bitmap returned from ScSyncScan.
-----

```

```

PORTEE_DLLCV_LB void ScSyncFreeBitmap(LPBITMAPINFO bitmap);
-----

```

```

// Stop the sync scan command.
// After calling this ScSyncLastError with return an error with dwError = CV_SCAN_ABORTED
-----

```

```

PORTEE_DLLCV_LB void ScSyncStopScan();
-----

```

```

#define CV_ERROR_BUFFER_SIZE 1024

```

```

// Returns true if more errors, false whej the last error is read.
// Buffer must be at least CV_ERROR_BUFFER_SIZE in size
-----

```

```

PORTEE_DLLCV_LB BOOL ScSyncLastError(DWORD* dwError, DWORD* dwType, LPSTR buffer);
-----

```

```

/////////////////////////////////////////////////////////////////
//      Definition of barcode types
/////////////////////////////////////////////////////////////////
#define BARCODE_NDEF          0
#define BARCODE_UPC_A        1
#define BARCODE_UPC_E        2
#define BARCODE_EAN13        3
#define BARCODE_EAN8         4
#define BARCODE_39           5
#define BARCODE_ITF          6
#define BARCODE_CODABAR      7
#define BARCODE_CODE93       8
#define BARCODE_CODE128      9
#define BARCODE_LOGO         256    // bit mask

```

```

/////////////////////////////////////////////////////////////////
//      Definition of printing modes
/////////////////////////////////////////////////////////////////
#define IMP_8DOT_SINGLE       0
#define IMP_8DOT_DOUBLE      1
#define IMP_24DOT_SINGLE     2
#define IMP_24DOT_DOUBLE     3

#define IMP_MODE_STANDARD    0
#define IMP_MODE_PAGE        1

```

```

/////////////////////////////////////////////////////////////////
//      Definition of errors and warnings
/////////////////////////////////////////////////////////////////

// Type of Errors
#define CV_FATAL_ERROR       0
#define CV_ERROR             1
#define CV_WARNING           2
#define CV_NO_ERROR          3

```

```

// Error Codes
#define CV_SCANNER_NOT_CONNECTED  0
#define CV_SCANNER_ERROR_CALIB    1
#define CV_PRINTER_ERROR          2
#define CV_PRINTER_OFFLINE        3
#define CV_PRINTER_PAPER_ERROR    4
#define CV_BARCODE                5
#define CV_DEBUG                  6
#define CV_PRINTER_COVER_ERROR     7
#define CV_PRINTER_HEAD_UNLOAD    8
#define CV_PIPE_BROKEN            9
#define CV_PRINTER_NEARLY_OUT     10
#define CV_BARCODE_CHECKSUM_BAD   11
#define CV_OCR                    12
#define CV_SCAN_ABORTED           13

#define CV_CAL_DPI_100            0
#define CV_CAL_DPI_200            1

```

```

// FLAGS FOR MOTTEST (duplicates from "e.h")
#define EmMOTON      0x8000
#define EmMOTREV     0x4000

```

START OF DEPRECATED BLOC

These are old and now obsolete versions of the DLL's functions
However they will be kept in case there are legacy apps relying on them
DO NOT USE for new applications. In some cases they have different functionality
and in others cases limited functionality compared to the Sc** equivalents

```
PORTEE_DLLCV_LB int CALLCONV DllCv_Init(FARPROC fpFinScanning,FARPROC fpError);
PORTEE_DLLCV_LB BOOL CALLCONV DllCv_PrintFmtString(int Height,int Width,int Flags,char *Font,int StartX,char *Str);
PORTEE_DLLCV_LB BOOL CALLCONV DllCv_PrintFile (char *FullFileName);
PORTEE_DLLCV_LB BOOL CALLCONV DllCv_BarCode (char *Code,int TypeCode);
PORTEE_DLLCV_LB BOOL CALLCONV DllCv_CutPaper (void);
PORTEE_DLLCV_LB BOOL CALLCONV DllCv_InvalidateReceipt (void);
PORTEE_DLLCV_LB BOOL CALLCONV DllCv_EndTransac(char *Code,int TypeCode);
PORTEE_DLLCV_LB int CALLCONV DllCv_Calibrate(int Step,int Param);
PORTEE_DLLCV_LB BOOL CALLCONV DllCv_ModeImp(int ResoImp, int ModeImp);
PORTEE_DLLCV_LB BOOL CALLCONV DllCv_StopScanner (void);
PORTEE_DLLCV_LB BOOL CALLCONV DllCv_StartScanner (void);
PORTEE_DLLCV_LB BOOL CALLCONV DllCv_PrintImage(BITMAP *pBmp);
PORTEE_DLLCV_LB BOOL CALLCONV DllCv_InitModeDib(FARPROC fpFinScanning,FARPROC fpError);
PORTEE_DLLCV_LB BOOL CALLCONV DllCv_StartPrinting(void);
PORTEE_DLLCV_LB BOOL CALLCONV DllCv_EndPrinting(void);
PORTEE_DLLCV_LB BOOL CALLCONV DllCv_PrintString(char *Str);
```

END OF DEPRECATED BLOC
