

# CyberScExtend Scanner Interface API.

---

**\*\*Copyright(c) 2025 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

- V1.0 - 20th November 2025 - O.Cullum
- V1.1 - 26thNov 2025 - O.Cullum
  - Added **Barcode reading** support (string passed back via ProcessedScanCallback).
  - Improved Auto Cropping algorithm.
  - Improved preprocessing performance considerably.
- V2.0 - 9th Jan 2026 - O.Cullum
  - Add a new **more robust cropping** algorithm. *CropEdgeTolerance is now no longer used but kept for compatibility.*
  - Crop border can be set **positive (add border) or negative (trim edge)**. Defaults to `-"Crop Noise Filter Size+2"` which gives minimal trimming to avoid over-cropping.
  - black edge fill is now off by default
- V2.1 - 20th Jan 2026 - O.Cullum
  - **improved black edge fill** - If enabled there is now no flood filling of edge content, filling is limited to the edge area and is smoother.
  - added **preserve1bppFormat** option to `SetImageProcessingSettings()` to allow original 1bpp output (no deskew antialiasing) when needed.
  - This fixes a bug where 1 bit B/W scans were always returned as 8 bit greyscale. if `preserve1bppFormat` is true 1bpp is now returned.
- v2.5 - 5th mar 2026 - O.Cullum
  - Improved cropping. Now both faster and more reliable. Also improved black edge filling

## OVERVIEW

The CyberScExtend managed Preprocessing Wrapper is a high-level API designed to simplify the integration of scanning functionality into .Net based applications. It builds upon the existing CyberSc.dll API, providing a more user-friendly but also more powerful interface for developers. It uses the same call signatures as the original CyberSc.dll API making it easy to migrate existing applications.

The `ProcessedScanCallback()` callback delegate along with the associated `SInitWithPreProcessing()` method encapsulates the main new functionality of CyberScExtend. However, there are also some helper functions to manage some common tasks.

Note that the fact that CyberScExtend is a wrapper around the native unmanaged CyberSc.dll is important. Except for new functionality as detailed below, all calls are simply passed through to the underlying CyberSc.dll. Therefore, for all structures, enums and the following functions you should refer to the CyberSC-API documentation:

### Key Features

- *dotNet managed assembly*
- *Type Safety*: Enums for all configuration settings and error codes
- *Helper Methods*: Utilities like `GetCallbackString()` for callback handling
- *XML Documentation*: Full IntelliSense support
- *Same Conventions*: Maintains P/Invoke compatibility with existing CyberSc native DLL
- *Preprocessing Options*: Built-in support for Deskewing, Cropping etc.
- *64 bit and 32 bit support*

### Important Notes

- ScanPrint Printer Support: This is currently **NOT** supported. Use the native CyberSc.DLL API if printing is required.
- Thread Safety: Callbacks execute on the scanner's thread, use `Invoke()` for UI updates

## REPLACEMENT FUNCTIONS

The following functions are provided in CyberScExtend with the same signatures as those in CyberSc

### Scanner Functions

- `SInit()` - Initialize scanner and register callbacks - docs as `ScInit()`
- `SPOff()` - Switches the interface from scanner to printer mode - as `ScPOff()`
- `SCalibrate()` - Perform calibration and configuration as `ScSCalibrate()`
- `SRejectPaper()` - Eject paper from scanner - as `ScRejectPaper()`
- `SConfigPath()` - Set the path to the scanner configuration\Calibration Folder as `ScSConfigPath()`
- `SSetupPartScan()` - Setup the handler for partial scanning as `ScSetupPartScan()`

### Callbacks

- `ScanDataCallback` - Called when scanning completes
- `ErrorReportCallback` - Called for errors and debug messages
- `ScanPartDataCallback` - Called for partial scan completes

## NEW CALLBACKS

**\*\* ProcessedScanCallback(\*\***

New callback invoked when scanning completes. Scans are processed before invoking the callback delegate

use `SetImageProcessingSettings()` to configure the preprocessing options. Bitmaps are always rotate flipped ( around Y ) as raw scans arrive rotate flipped.

When you are done with the Bitmap object it should be disposed of via `bitmap.Dispose()` to free resources.

```
params:
    bitmap - Processed OutputBitmap image
    rawdata - Reference to scan image data structure (no processing done on this)
    barcode - Barcode string extracted from the scan (if processing option for it is enabled), otherwise empty string)
```

```
public delegate void ProcessedScanCallback( Bitmap bitmap, ref CyberImg rawdata, string barcode );
```

## NEW METHODS

### SInitWithPreProcessing()

Initialize the scanner for automatic preprocessed scans and error handling. Image Flipping (always), deskewing and/or cropping (optional) is done on scan completion before invoking the registered callback.

use `SetImageProcessingSettings()` to configure the preprocessing options.  
The `fpProcessedScanning` callback is invoked after preprocessing of the output Bitmap when scanning completes

Returns a packed 32-bit version value (non-zero on success, 0 on failure). Note identical to `SInit()`.  
Version info breakdown (least significant byte first):  
LSB: Hardware version  
Byte 2: Device driver version  
Byte 3: CyberSc Internal DLL version - different from the X.Y.Z format file version  
MSB: Reserved for application version

```
params
    fpProcessedScanning - Callback invoked with a preprocessed output bitmap when scanning completes
    fpError - Callback invoked for errors and debug messages. see cvErrors.CV_DEBUG.
    use GetCallbackString(UINTPtr) on the second parameter to retrieve the message text.
```

```
public static int SInitWithPreProcessing( ProcessedScanCallback fpProcessedScanning, ErrorReportCallback fpError )
```

### SetImageProcessingSettings()

Set the image pre processing settings for the scanner. Functions such as deskewing, cropping, margin removal, etc can be configured here.

For default settings, call with no parameters.

Defaults:

- Remove White Specks: Disabled - using this blurs the image slightly
- Auto Fill Black Edges: Disabled (false)
- Search For Barcodes: Disabled (false) - enables barcode detection during preprocessing
- Auto Deskew: Enabled (true)
- Auto Crop Scans: Enabled (true)
- Preserve 1bpp Format: Disabled (false) - if disabled 8 bit values are used for higher image quality from anti-aliasing after deskewing
- Crop Edge Tolerance: 20 - Not Currently used. Provides left/right safety zone. Specifies how close to edge to look for contents
- Crop Noise Filter Size: 3 - noise elements smaller than this + 1 are filtered out during cropping
- Horizontal Margins: 1.0% - Simple percentage based left/right trim applied BEFORE deskewing / cropping
- Crop Horizontal Border: defaults to -"Crop Noise Filter Size+2" - Sets a border (if positive) or a trim amount (if negative) applied during cropping
- Crop Vertical Border: defaults to -"Crop Noise Filter Size+2" - Sets a border (if positive) or a trim amount (if negative) applied during cropping
- Max Fill Black Edge Width: 50 - If black edge fill is enabled this limits the width of the filled edge area to avoid excessive filling.

If set to zero or negative the value of 1 is used. The value is scaled based on DPI above 200 DPI (e.g. at 300 DPI = 1.5x this value)

```
public static void SetImageProcessingSettings( bool removeWhiteSpecks = false,
    bool autoFillBlackEdges = false,
    bool searchForBarcodes = false,
    bool autoDeSkew = true,
    bool autoCropScans = true,
    bool preserve1bppFormat = false,
    int cropEdgeTolerance = 20, // Not currently used
    int cropNoiseFilterSize = 3,
    float defaultHorizMarginsPercent = 1.0f,
    int cropHorizontalBorderPixels = null,
    int cropVerticalBorderPixels = null,
    int maxFillBlackEdgeWidth = 50)
```

### GetInitMode()

Get the current initialization mode of the scanner.

Returns <== Initialization mode:  
0 = Not initialized  
1 = Initialized in raw mode ( SInit )  
2 = Initialized in preprocessing mode ( SInitWithPreProcessing )

```
public static int GetInitMode()
```

### SResetCallbacks()

Resets the initialization state. Call this before switching between `SInit()` and `SInitWithPreProcessing()` modes.  
Note: This does NOT uninitialize the hardware - it only clears the managed state.  
You should call `CV_CAL_STOPSCAN` via `SCalibrate()` before calling this.

```
public static void SResetCallbacks()
```

### GetCurrentDPI()

Helper to get the current DPI setting from the scanner hardware.

Returns  
DPI value (200 or 300), or 200 as default if query fails

```
public static int GetCurrentDPI()
```

### GetCallbackString()

Helper function (with basic error checking) to extract string from UIntPtr for CV\_BARCODE and CV\_DEBUG callbacks

Params

dwType - UIntPtr that may contain a string pointer

Returns

String if valid pointer, null otherwise

```
public static string GetCallbackString( UIntPtr dwType )
```

### CyberImgToBitmap()

Helper method to convert CyberImg structure to Bitmap

Note that bitmap.Dispose() should be called to free up memory when the bitmap is no longer needed

params

rawdata - CyberImg structure (returned by the scanner callback)

Return

Bitmap representation of the image

```
public static Bitmap CyberImgToBitmap(ref CyberImg rawdata)
```

## STRUCTURES AND ENUMS USED

These are defined internally in the CyberScExtend class and are the same as those used in the CyberSc.dll API. However, all CyberSc defined structures and enums value can be used directly with CyberScExtend as well.

```
// =====
// STRUCTURES
// =====

// Image interface structure for callback
[StructLayout(LayoutKind.Sequential)]
public struct CyberImg
{
    public IntPtr pIm;           // pointer to image (packed bit array, 32bit aligned)
    public IntPtr pCoef;         // correction coefficients currently in use
    public IntPtr pGamma;        // gamma correction table currently in use
    public IntPtr pTDens;        // bit count table for BYTE
    public IntPtr pTSwap;        // bit reverse table for BYTE
                                //public IntPtr WinBITMAP;
                                //public IntPtr WinHDC;
    public int lTime;            // time
    public short wWidBy;         // image width in bytes
    public short wWid;           // image width in pixels
    public short wHei;           // image height in pixels
    public short wNoClr;         // no of colors (1 = monochrome, 3= rgb)
    public byte bPlanes;         // no of planes (1,4,5,8 supported)
    public byte bType;           // type of data (colors present)
    public byte bRet;            // return value (reserved)
}

// =====
// CONFIGURATION SETTINGS ENUM
// =====

//Configuration settings for ScScalibrate function
public enum ConfigSettings : int
{
    CV_CAL_STOPSCAN = 0,
    CV_CAL_CALIBRATE = 1,           // Done once for each scanner. Survives re-boots but must be re-done if hardware changes
                                    // Calibration file saved by default to %ProgramData%\Scanprint\Scanner\
    CV_CAL_SET_BW = 2,             // Black and White threshold ( 0 to 255 ) for B/W scanning modes
    CV_CAL_GET_BW = 3,             // superceeded by gamma setting for all modes
    CV_CAL_GET_RESO = 4,           // Despite the name this is actually used to set scan color depth
    CV_CAL_SET_RESO = 5,
    CV_CAL_STARTSCAN = 6,          // scans started as soon as clean paper edge detected
    CV_CAL_GET_LIGHT = 7,
    CV_CAL_SET_LIGHT = 8,          // Light level set by calibration but can be tweaked (0 to 99 max)
    CV_CAL_GET_DEV = 9,
    CV_CAL_SET_DEV = 10,           // The scan resolution DPI is set via this

    CV_CAL_FORCESCAN = 11,         // Force scan as soon as any paper is detected. This is rather than waiting for
                                    // the full paper edge to be detected. Images can be more clipped or skewed. Use with care.
    CV_CAL_TOGGLE_JAM = 23,        // 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 ruin the paper. use at your own risk.

    CV_CAL_PAPER = 24,             // get the current scanner paper status. returns bitmask of paper status flags (set -> paper present)
                                    // (see ScannerStatus enum)

    CV_CAL_SET_GAMMA = 29,         // superceeds SET_BW. Use this to set scan sensitivity
    CV_CAL_GET_GAMMA = 30,
    CV_CAL_STATUS_ACTIVITY = 37,   // activity status of scanner (see ScannerStatus enum)

    CV_CAL_FORCESCAN_STAY = 46,    // Like FORCESCAN but stays in force mode until STOPSCAN called
    CV_CAL_RECALIBRATE = 47        // Safe Calibration - always overwrites existing calibration file
}

// =====
// ERROR CODES ENUM
// =====

// Error codes for error callback
public enum cvErrors : uint
```

```

{
    CV_SCANNER_NOT_CONNECTED = 0,
    CV_SCANNER_ERROR_CALIB = 1,
    CV_PRINTER_ERROR = 2,
    CV_PRINTER_OFFLINE = 3,
    CV_PRINTER_PAPER_ERROR = 4,
    CV_BARCODE = 5,           // this is a deprecated barcode callback - use the new ProcessedScanCallback method instead
    CV_DEBUG = 6,
    CV_PRINTER_COVER_ERROR = 7,
    CV_PRINTER_HEAD_UNLOAD = 8,
    CV_PIPE_BROKEN = 9,
    CV_PRINTER_NEARLY_OUT = 10,
    CV_BARCODE_CHECKSUM_BAD = 11, // this is a deprecated barcode callback - use the new ProcessedScanCallback method instead
    CV_OCR = 12,
    CV_SCAN_ABORTED = 13
}

public enum cvErrorType : int
{
    CV_FATAL_ERROR = 0,
    CV_ERROR = 1,
    CV_WARNING = 2,
    CV_NO_ERROR = 3
}

// =====
// RESOLUTION/COLOR DEPTH MODES
// =====

// Scanning resolution and color depth modes
public enum Resolutions : int
{
    EACRmFULL256 = 0,        // full cis resolution 256 levels (8bpp)
    EACRmHALF256 = 1,        // half cis resolution 256 levels (8bpp)
    EACRmFULL2 = 2           // Full res in black and white.
}

// =====
// DEVICE CODES (DPI SETTINGS)
// =====

// Scanner device codes (DPI settings)
public enum DeviceCode : int
{
    EDEV200A4_1728 = 1,      // 200 DPI
    EDEV300A4_2592 = 3       // 300 DPI
}

// =====
// SCANNER STATUS FLAGS
// =====

// Scanner status mask flags used by CV_CAL_STATUS_ACTIVITY
[Flags]
public enum ScannerStatus : int
{
    SS_LOADED = 0x01,        // scanner driver found and loaded
    SS_THREAD = 0x02,        // scanner thread started and hasn't terminated
    SS_BUSY = 0x04,          // scanner is busy (scanner thread not idle sleeping)
    SS_STOPPED = 0x08,       // scanner polling is disabled (note: it is still possible to get BUSY
                             // but the hardware is not used)
    SS_SCANNING = 0x10,      // scanner is currently in the process of scanning
    SS_CALLBACK = 0x20,      // scanner thread is in user scan callback
    SS_OSCILLO = 0x40,       // scanner is in oscilloscope display mode (in this mode the POLL counter
                             // increments much faster)
    SS_PAPER = 0x00ff00,     // paper detectors at last poll
    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)
}

```