

ScanPrint Dev-Kit Quick Start Guide

This is a guide to help people start developing software for the ScanPrint SP5001 USB scanner.

Prerequisites

This guide assumes that you have experience developing in C/C++/C# on Windows and already have Visual Studio installed. The sample code has been built and tested on Windows 11 and Visual Studio 2022. The free community version includes everything you'll need).

During installation of Visual Studio you will need to select the 'Desktop development with C++' workload.

Installing

All the software needed can be downloaded from the ScanPrint website at <http://scanprint.co.uk/Drivers>. The main files needed are:

ScanPrint-Setup.exe is the Combined Auto Installer for the required Windows Drivers, which are needed before software can connect to the scanner and also for the **Engbench3** test application, which allows you to test and configure the scanner.

Contents of the Dev-Kit

ScanPrint-Dev-Kit-X.Y.Z.zip is the development kit and it contains the essential binaries plus SDK files and development example source files needed for development.

1. Examples --- simple C++ examples
2. PrinterDemo --- C# example
3. ScanDemo --- C# example - the recommended starting point for real C# development if using the CyberSc.dll high level driver (64 or 32 bit)
4. **CyberSc-SDK** --- The CyberSc API / high level driver release and Debug DLL plus header and lib files needed by applications
5. **CyberScExtend-SDK** --- The API for the new Managed preprocessing Wrapper for the high level driver release
6. ScanExtendedDemo --- C# example - the recommended starting point for real C# development if using the CyberScExtend Wrapper (64 or 32 bit)
7. CyberScExtendConsoleDemo --- C# example -- A simple demo app to get you started using the CyberScExtend Wrapper
8. Twain32 API (Deprecated - use at your own risk)
9. scanprintdriver-XXbit.msi - the 32 or 64 bit low level Windows driver for the Scanprint scanner and Scanprint printer.
10. EngBench3 - binaries for the test and config app

NOTE - In many cases, depending on the test facilities you build into your application you may not need EngBench3 in which case you will only need to distribute the appropriate SDK binaries and optionally **Cfg1xxx.dat** (the Scanner calibration configuration - see below) files and obviously the scanprintdriver-XXbit.msi device drivers.

Guide overview

This guide can be divided into two sections.

PART A

This first covers development using the original CyberSc.dll high level driver API. This is suitable for both C++ and C# development.

PART B

The second section covers development using the new CyberScExtend managed preprocessing wrapper for the high level driver. This is best suited for .NET managed languages such as VB.NET and C# and it offers advanced features such as automatic Deskewing and cropping of scans. All of the useful scanning functions of the original CyberSc.dll API are available using the same function signatures making it easy to migrate existing applications.

PART A - Using the CyberSc.dll High Level Driver API

Getting Started

CyberSC.dll is a high performance driver providing application access to the Scanprint Scanners and Printers. see the CyberSC-API documentation for details.

A useful starting point is the sample code in \examples\. These are simple command line C++ programs that will scan an image from the scanner and save it as a bitmap.

There are two APIs you can use talk to the scanner. Blocking synchronous function calls (Sync) or a callback based asynchronous calls (Async).

Sync API

A very simple program to read an image from the scanner is below.

```
#include <windows.h>
#include "CyberSc.h"
int main()
{
    ScSyncInit();
    BITMAPINFO* pBitmap = ScSyncScan(false);
    //Do something with the bitmap.
}
```

A fuller version with error checking, and freeing the bitmap is in the examples\sync-cpp\ folder

ScSyncInit Initialise the scanner for the synchronous API. ScSyncScan Block waiting for an image to be scanned or an error. If the parameter is true the function treats debug messages as errors and will return on every debug message.

Async API

A sample showing the async api is in examples\async-cpp\

The scanner initialization function is ScInit. This takes two functions to be used as callbacks. One for when an image has been scanned and one as an error reporting method, e.g. ScInit((FARPROC)ImageScanned, (FARPROC)ErrorNotify). The callback functions will be called on a different thread.

Then to tell the scanner to be ready to scan call ScSCalibrate with CV_CAL_STARTSCAN, e.g. ScSCalibrate(CV_CAL_STARTSCAN, 0)

Building and running the sample.

From Visual Studio.

Open "examples/examples.sln" Rebuild the projects from the menu.

From the command line

Open the "x86 Native Tools Command Prompt for VS 2019 or VS2022"

```
cd examples
msbuild
```

Running

The example command line applications can be run from the inside Visual Studio under the debugger or from the command line in the examples/Debug directory. They will save the scanned image into the working directory and will terminate when an image has been scanned, an error has been received or if Ctrl-C is pressed.

Real world C# based applications

The recommended starting point for C# development is the **ScanDemo** C# example application included in the Dev-Kit. This is a Winform based application which shows how to use the CyberSc.dll API from C#. it covers most of the essential scanning functions including

- scanner initialization,
- scanning in colour and B&W modes,
- error handling and
- scanner calibration. The source code is well commented and should be easy to follow.

Calibration Note

Each physical scanner needs to be calibrated for the best performance. Using the **Cfg1xxx.dat default** calibration configuration file will in some cases lead to **sub-optimal performance**, particularly in colour scan modes. It is recommended that your application also offers the user a calibration facility via the ScSCalibrate(CV_CAL_RECALIBRATE, 0) command (see CyberSc_API.pdf)

Version 7.0.0 of the CyberSc.dll API introduced improved calibration routines. The calibration config file (Cfg1xxx.dat) is now **ALWAYS** saved by default into the Windows ProgramData\ScanPrint\Scanner folder. This change was made so that Calibration always works correctly irrespective of the user privilege level of the application.

Here's an example C# Calibrate command for a Winform based app.

```
private void CalibrateToolStripMenuItem_Click(object sender, EventArgs e)
{
    try
    {
        CyberCs.ScSCalibrate(CalibrationSettings.CV_CAL_STOPSCAN, 0);
        if (MessageBox.Show("Insert Blank A4 sheet of paper and then click OK", "ReCalibration",
            MessageBoxButtons.OKCancel, MessageBoxIcon.Warning) == DialogResult.OK)
        {
            if (CyberCs.ScSCalibrate(CalibrationSettings.CV_CAL_RECALIBRATE, 0) > 0)
            {
                StatusText = "Scanner Calibrated OK";
                m_scannerInitDoneOK = true;
            }
        }
    }
}
```

```

        else
        {
            StatusText = "Problem ReCalibrating scanner";
        }
    }
    CyberCs.ScSCalibrate(CalibrationSettings.CV_CAL_STARTSCAN, 0);
}
catch (Exception ex)
{
    StatusText = "Error Calibrating scanner \n\n" + ex.Message;
    MessageBox.Show(StatusText, m_caption, MessageBoxButtons.OK, MessageBoxIcon.Error);
    CriticalErrorOccured = true;
    CyberCs.ScSCalibrate(CalibrationSettings.CV_CAL_STARTSCAN, 0);
}
}
}

```

Raw image orientation Note

Raw images arrive upside down from the scanner so we need to flip them. this can be done with ;

```
bitmap.RotateFlip(RotateFlipType.RotateNoneFlipY);
```

(Note that CyberScExtend Processed images are always flipped the right way up even if crop and/deskew is disabled)

PART B - Using the CyberScExtend DLL Managed Preprocessing Wrapper

Introduction

The CyberScExtend Managed Preprocessing Wrapper is a high-level API designed to simplify the integration of scanning functionalities 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.

Key Features

- **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 native DLL
- **Preprocessing Options:** Built-in support for Deskewing, Cropping, Barcode Reading etc.
- **64 bit and 32 bit support**

Where appropriate the wrapper functions have the same signatures as the original CyberSc native DLL API.

Scanner Functions

- `SInit()` - Initialize scanner and register callbacks
- `SPOff()` - Switches the interface from scanner to printer mode
- `SCalibrate()` - Perform calibration and configuration
- `SRejectPaper()` - Eject paper from scanner
- `SConfigPath()` - Set the path to the scanner configuration\Calibration Folder
- `SSetupPartScan()` - Setup the handler for partial scanning
- `SInitWithPreProcessing()` - Initialize scanner with preprocessing specified - **NEW**
- `SetImageProcessingSettings()` - Set preprocessing options (Deskew, Crop, Barcode Reading etc) - **NEW**
- `GetInitMode()` - Get current scanner initialization mode - **NEW**
- `SResetCallbacks()` Resets the initialization state. Call this before switching between the `SInit()` and `SInitWithPreProcessing()` modes - **NEW**

Callbacks

- `ScanDataCallback` - Called when scanning completes
- `ErrorReportCallback` - Called for errors and debug messages
- `ProcessedScanCallback` - Called when a preprocessed scan completes **NEW**
- `ScanPartDataCallback` - Called for partial scan completes

helper Methods

- `GetCallbackString()` - Convert error callback data to string - **NEW**
- `CyberImgToBitmap()` - Convert scan data to Bitmap object - **NEW**
- `GetCurrentDPI()` - Get the current DPI setting from the scanner hardware. - **NEW**

Important Notes

- **ScanPrint Printer Support:** This is currently **NOT** supported. Use the native CyberSc.DLL API if printing is required.
- **Initialization Order:** Call `SInit()` or `SInitWithPreProcessing` before any other scanner functions.
- **Callback Lifetime:** you must keep references to callback delegates to prevent garbage collection.
- **Thread Safety:** Callbacks execute on the scanner's thread, use `Invoke()` for UI updates
- **Dependencies:** There are a number of DLLs (E.g. CyberSc.dll) which must be in the same directory or in PATH as your application. Make sure to copy the contents of the appropriate CyberScExtend-SDK folder to your application output folder.
- **32 bit vs 64 bit:** Ensure your application targets the correct architecture matching the DLLs used.
- **Calibration File Location:** The calibration file (Cfg1xxx.dat) is saved in `ProgramData\ScanPrint\Scanner` by default.
- As CyberScExtend is a managed wrapper around the native CyberSc.dll API the structures and enums used are a subset of those used in the native API. Except for new functionality the behaviour will be as described in the CyberSc_API documentation.

Getting Started

There are two C# sample applications included in the Dev-Kit that demonstrate how to use the new CyberScExtend high level wrapper

Note these demos programs will (as originally supplied in the Dev-kit) only build correctly for x64 bit builds. This is because these projects only allow a single version of an assembly at a time and the 64 bit version of CyberScExtended.dll is currently referenced. if you need to build for x86 (32-bits) you will need to change the project reference to the Win32 version of CyberScExtend.dll

CyberScExtendConsoleDemo.

This is a simple console application covering the essentials. Here's a cut down version of the main code.

```
using System;
using System.Drawing;
using System.IO;

using CyberEx = CyberScExtend.CyberSc; // this is the actual extended CyberSc DLL with image pre-processing facilities

using CyberConfigs = CyberScExtend.ConfigSettings;
using CyberResolutions = CyberScExtend.Resolutions;
using CyberDeviceCode = CyberScExtend.DeviceCode;
using CyberCvErrors = CyberScExtend.cvErrors;

namespace CyberScExtendDemo
{
    internal static class Program
    {
        // basic required callbacks
        private static CyberEx.ErrorReportCallback errCb = ErrorCallback;
        private static CyberEx.ProcessedScanCallback ProScanCb = ProcessecdScanCallback ;

        [STAThread]
        private static void Main(string[] args)
        {
            // Init wrapper
            // int ver = CyberEx.SInit(scanCb, errCb);
            int ver = CyberEx.SInitWithPreProcessing(ProScanCb, errCb);
            if (ver == 0)
            {
                Console.WriteLine("Init failed.");
                return;
            }
            CyberEx.SPOff(); // Ensure scanner is ON ( printer is off and) before config

            // Enable pre-processing (deskew, autocrop, barcode reading etc) - optional
            bool removeWhiteSpcs = false;
            bool autoFillBlackEdges = true;
            bool autoReadBarcodes = true;
            bool autoDeskew = true;
            bool autoCrop = true;
            CyberEx.SetImageProcessingSettings(removeWhiteSpcs, autoFillBlackEdges, autoReadBarcodes, autoDeskew, autoCrop);

            CyberEx.SCalibrate((int)CyberConfigs.CV_CAL_STOPSCAN, UIntPtr.Zero);

            CyberEx.SCalibrate((int)CyberConfigs.CV_CAL_SET_DEV, (UIntPtr)(int)CyberDeviceCode.EDEV200A4_1728); // 200 DPI

            CyberEx.SCalibrate((int)CyberConfigs.CV_CAL_SET_RESO, (UIntPtr)(int)CyberResolutions.EACRmFULL256); // 24 bit color
            CyberEx.SCalibrate((int)CyberConfigs.CV_CAL_SET_RESO, (UIntPtr)0x60);

            CyberEx.SCalibrate((int)CyberConfigs.CV_CAL_STARTSCAN, UIntPtr.Zero);

            Console.WriteLine("Ready. Insert paper into scanner OR Press any key to exit.");
            Console.ReadKey();
        }

        // -----
        // Callback implementations
        // -----
    }
}
```

```

private static void ProcessedScanCallback(Bitmap bitmap, ref CyberScExtend.CyberImg img, string barcode)
{
    // images will be flipped, Deskewed and Cropped by Default !

    Console.WriteLine($"(Processed Scan callback) w={img.wWid} h={img.wHei} colour mode={img.wNoClr} Bit planes={img.bPlanes}");
    Console.WriteLine($"(Processed Scan callback) barcode = {barcode}");

    // write out image to file for demo purposes in png format in scans folder
    string folderPath = Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "scans");
    Directory.CreateDirectory(folderPath);
    string filename = Path.Combine(folderPath, $"scan_{DateTime.Now.Ticks}.png");
    bitmap.Save(filename, System.Drawing.Imaging.ImageFormat.Png);

    bitmap.Dispose(); // free bitmap
}

private static void ErrorCallback(uint dwError, UIntPtr dwType)
{
    switch (dwError)
    {
        case (uint)CyberCvErrors.CV_BARCODE:           // note this not the new barcode reading facility
                                                         // but a deprecated legacy feature
        case (uint)CyberCvErrors.CV_DEBUG:
            string message = CyberEx.GetCallbackString(dwType);
            Console.WriteLine($"Message: {message}");
            break;

        case (uint)CyberCvErrors.CV_SCANNER_NOT_CONNECTED:
            Console.WriteLine("Scanner not connected");
            break;

        default:
            Console.WriteLine($"Error: {dwError}");
            break;
    }
}
}
}

```

ScanExtendedDemo -

This is the recommended starting point for real C# development using the CyberScExtend wrapper. Its a Winform based application which shows how to use the CyberScExtend.dll API from C#. it covers most of the essential scanning functions including

- scanner initialization,
- setting preprocessing options such as Deskew, cropping and barcode reading etc
- scanning in colour and B&W modes,
- error handling and
- scanner calibration etc The source code is well commented and should be easy to follow.

CyberScExtend Upgrade Steps

If you are migrating an existing application from using the CyberSc.dll API to the new CyberScExtend wrapper here are the basic steps needed

1. Copy the **full** contents of the CyberScExtend-SDK\x64-Binaries or CyberScExtend-SDK\x64-Binaries folder, as appropriate, to your application output folder. (i.e. the same folder as your application exe)
2. For .NET projects add a reference to the appropriate CyberScExtend.dll (x64 or x86) to your project.
3. Change all references from using CyberCs = CyberSc.CyberSc; to using CyberEx = CyberScExtend.CyberSc;
4. Note the slightly changed function names for existing functionality. In general
 - a. ScFunctionName becomes SFunctionName
 - b. ScSFunctionName becomes SFunctionName
5. Change all references to enums. E.g. Change CyberSc.ConfigSettings to CyberScExtend.ConfigSettings etc.
6. Change all calls to ScInit() to SInitWithPreProcessing() if you want to use the preprocessing features. Note
 - a. keep a reference to the new ProcessedScanCallback delegate to prevent garbage collection.
 - b. The bitmaps returned by the ProcessedScanCallback are already deskewed and cropped by default. This a bitmap can be used directly in your application but call bitmap.Dispose() when finished with it to free up memory.
 - c. The raw scan data passed to the callback is still available if needed.