Sense 20/20

User's Guide to Sense 2020 SDK

Version 2.0.6.0

Document No. 94-101227

BaySpec, Inc.

101 Hammond Avenue, Fremont, CA 94539

Tel. (510) 661 2008 • Fax: (510) 661 2009

Web: www.bayspec.com

October 2009

Table of Contents:

1. Overview

BaySpec Sense 20/20 Software Development Kit (SDK) supplies the interface for the software developer to access the BaySpec SuperGamut™ and Nunavut™ series spectral engines. The Dynamic Link Library (DLL) in the SDK can be used under different programming environments: C, C++, Visual Basic and LabVIEW. The library is compatible with Windows 2000, XP and Vista. The SDK provides a set of functions that allow user to configure and control the spectral engines, acquire spectra and post-process the spectrum data. The sample codes of Visual C++ 6.0 and LabVIEW 8.2 are included in this SDK.

2. System Requirements

2.1 Hardware

The minimum requirements for the computer are listed below:

OS:	Microsoft Windows 2000/XP/Vista(Ultimate version)
CPU:	1 GHz, recommend Intel Core 2 Duo processor or equivalent
RAM:	512 MB, recommend 1GB or higher
USB Port:	USB 2.0

2.2 Software

USB driver: Set up communication between BaySpec spectral engine and application program running on the host computer.

_Usb20irDevOpDLL.dll: Low level DLL to access BaySpec spectral engine

Sense2020Dll.h: Header file (for C and C++).

Sense2020Dll.lib: Library file (for C and C++).

Sense2020Dll.dll: Wrapped DLL file with data processing functions (Revision 2.0.4.2)

3. DLL Functions

3.1 Get DLL Version

BOOL DLL_Get_DLL_Versions(DWORD* pdwDLLVersion);

Description: Get DLL version

Parameters: DWORD* **pdwDLLVersion**: Pointer of DLL revision

Return value: Boolean

TRUE: Open device OK

FALSE: Open device failed

3.2 Initialize Device

BOOL DLL_Open_Device();

Description: Create an instance and initialize the device

Parameters: None

Return value: Boolean

TRUE: Open device OK

FALSE: Open device failed

Note: This function allocate internal memories for data acquisition.

DLL_Close_Device must be called to release these memories

3.3 Close Device

void DLL_Close_Device();

Description: Delete the instance and release allocated memories

Parameter: None

Return value: None

Note: The memories allocated for data acquisition are deleted in this function

3.4 Get Device Status

BOOL DLL_Is_Device_OK();

Description: Check device current status

Parameter: None

Return value: Boolean

TRUE: Device operated correctly

FALSE: Device not initialized or connected

3.5 Get Device Count

int DLL_Get_Device_Count();

Description: Check connected USB20BS device count

Parameter: None

Return value: Signed integer

≠ -1: Device Count

-1: Operation failed

3.6 Select Device

BOOL DLL_Select_Device(WORD wDeviceIndex);

Description: Select device with index and active it

Parameter: WORD wDeviceIndex: Index of selected device (start from 0)

Return value: Boolean

TRUE: Device operated correctly

FALSE: Device not initialized or connected

3.7 Get Device Serial Number

BOOL DLL_Get_Device_SN(LPSTR lpszSN);

Description: Get device serial number

Parameter: LPSTR lpszSN: Pointer of character array allocated by the user.

(Buffer size >= 8 bytes.)

Return value: Boolean

TRUE: Operation completed

FALSE: Operation failed

3.8 Get Device Hardware and Firmware Revision

BOOL DLL_Get_HW_FW_REV(DWORD* pdwHW,

DWORD* pdwFW);

Description: Get device's hardware and firmware revision

Parameter: DWORD* pdwHW: Pointer of hardware revision

DWORD* pdwFW: Pointer of firmware revision

Return value: Boolean

TRUE: Operation completed

FALSE: Operation failed

3.9 Get Device Pixel Count

int DLL_Get_Pixel_Count();

Description: Get device pixel count

Parameter: None

Return value: Signed integer

≠ -1: Pixel Count

-1: Operation failed

3.10 Set Device Integration Time

BOOL DLL_Set_Integration_Time_Line_Rate(DWORD dwIntegrationTime,
WORD wLineRate);

Description: Set device integration time and line rate

Parameter: DWORD dwIntegrationTime: Integration time in μs.

(Range: 0 to 300000000 μs)

WORD wLineRate: Spectrum sampling rate in Hz

(Range: 1 to 5000 Hz, Condition: 10^6 / wLineRate >= dwIntegrationTime)

Return value: Boolean

TRUE: Operation completed

FALSE: Operation failed

3.11 Get Device Integration Time

BOOL DLL_Get_Integration_Time_Line_Rate (DWORD* pdwIntegrationTime,
WORD* pwLineRate);

Description: Get device integration time

Parameter: DWORD* pdwIntegrationTime: Pointer of integration time in μs

WORD* wLineRate: Pointer of spectrum sampling rate in Hz

Return value: Boolean

TRUE: Operation completed

FALSE: Operation failed

3.12 Get Device Temperature

BOOL DLL_Get_Temperature(double* pdblCaseTemperature,
double* pdblSensorTemperature);

Description: Get device case and sensor temperature in Celsius.

Parameter: double* pdblCaseTemperature: Pionter of case temperature.
double* pdblSensorTemperature: Pionter of sensor temperature.

Return value: Boolean
TRUE: Operation completed
FALSE: Operation failed

3.13 Get Snapshot Spectrum

BOOL DLL_Get_Spectra(double* pdblData,
double* pdblCaseTemperature);

Description: Acquire one snapshot spectrum.

Parameter: double* pdblData: Pointer of spectrum power array allocated by the user (Buffer size >= pixel count)
double* pdblCaseTemperature: Pointer of case temperature in current acquisition.

Return value: Boolean
TRUE: Operation completed
FALSE: Operation failed

3.14 Load Calibration Data

int DLL_Load_Calibration_Data(WORD wDeviceCount,
LPCSTR lpszCalibFilePath);

Description: Load calibration data for selected device

Parameter: WORD wDeviceCount: Index of selected device
LPCSTR lpszCalibFilePath: Full file path of calibration data file

Return value: Integer
0: Operation completed
1: Device not connected

2: Device index >= Connected device count

3: Open file failed;

4: Wrong revision of the calibration data;

5: Wrong calibration data format;

6: Device serial number is not match with that in calibration file

3.15 Get Wavelength Mapping

BOOL DLL_Get_Wavelength(double dblCaseTemperature,
double* pdblWavelength);

Description: Get the calibrated wavelength for each pixel at current case temperature.

Parameter: double dblCaseTemperature: Current device case temperature in Celsius

double* pdblWavelength:

Pointer of spectrum wavelength array allocated by the user buffer. (Buffer size >= pixel count)

Return value: Boolean

TRUE: Operation completed

FALSE: Operation failed

Note: The calibration data must be loaded before calling this function

3.16 Search Peaks in Spectrum

BOOL DLL_Search_Peaks(WORD wPeakSearchMode
WORD wPixelCount,
double* pdblWL,
double* pdblPwr,
double dblNoiseThreshold,
double* dblNoiseThresholdProfile,
DWORD* pdwPeakInfo);

Description: Search all peaks in the spectrum

Parameter: WORD wPeakSearchMode: Peak search mode

0 - Use uniform noise threshold

1 - Use profile noise threshold

WORD wPixelCount: Pixel count

Double* pdblWL:

Pointer of wavelength or pixel array allocated and passed by the user (Buffer size >= pixel count. For pixel array, pdblWL[] = { 0, 1, 2, ..., pixel count – 1}

double* pdblPwr:

Pointer of spectrum power array allocated and passed by the user (Buffer size >= pixel count, It is recommended that the spectrum intensities have been subtracted by background before calling this function)

double wNoiseThreshold:

Noise threshold. The power of all peaks must be larger than this noise threshold. The power increase of adjacent pixels at beginning of rising edge of all peaks must be larger than tenth of noise threshold.

double* wNoiseThresholdProfile:

Pointer of noise threshold profile array allocated and passed by the user (Buffer size >= pixel count.). For n^{th} pixel, its noise threshold is in n^{th} element of this array.

DWORD* pdwPeakInfo: Pointer of searched spectrum peak information array allocated by the user (Buffer size >= 512)

Peak information format: pdwPeakInfo[0] = 0xFFFFFFFF

pdwPeakInfo[1] = Peak count,

pdwPeakInfo[2] to pdwPeakInfo[128]

= Peak pixels or wavelengths × 1000

pdwPeakInfo[129] to pdwPeakInfo [255]

= Peak power × 1000

Return value: Boolean

TRUE: Operation completed

FALSE: Operation failed

3.17 Start Constantly Grabbing spectra

int DLL_Start_Constantly_Grab_Spectra (WORD wProcessMode,

WORD wPeakSearchMode,
WORD wExternalTimingMode,
WORD wExternalTriggerMode,
double* pdblBackgroundData,
double dblNoiseThreshold,
double* pdblNoiseThresholdProfile);

Description: Start constantly grabbing spectra.

Parameter: WORD wProcessMode:

0 - Save raw data only

1 - Find and save peak pixel and peak power only

2 - Find and save peak wavelength and peak power only

3 - Save raw data, find and save peak pixel and peak power

4 - Save raw data, find and save peak wavelength and peak power

WORD wPeakSearchMode: Peak search mode

0 - Use uniform noise threshold

1 - Use profile noise threshold

WORD wExternalTimingMode: External trigger timing mode

0 - Synchronous external triggering

1 - Asynchronous external triggering

2 - Disable external triggering

WORD wExternalTriggerMode: External triggering mode

0- Level triggering

1 - Rising edge triggering

double* pdblBackgroundData: Pointer to background array buffer, buffer size = pixel count in mode 1, 2, 3 and 4

double dblNoiseThreshold:

The powers of all peaks must > dblNoiseThreshold and the power increase of adjacent pixels at beginning of rising edge of all peaks must > 0.1 * dblNoiseThreshold in mode 1, 2, 3 and 4

double* wNoiseThresholdProfile:

Pointer of noise threshold profile array allocated and passed by the user

(Buffer size >= pixel count.). For n^{th} pixel, its noise threshold is in n^{th} element of this array.

Return value: Integer

0 - Operation OK

1 - Device not connected

2 - nProcessMode is larger than device count

3 - Constant grab spectra in running

4 - pdblNoiseThresholdProfile = NULL when wPeakSearchMode = 2

5 - Calibration data not loaded

6 - Allocate memory failed

3.18 Stop Constantly Grabbing spectra

void DLL_Stop_Constantly_Grab_Spectra();

Description: Stop constantly grabbing spectra.

Parameter: None

Return value: None

3.19 Get the Last Spectra Index

int DLL_Get_Last_Spectrum_Index()

Description: Get the last spectrum index saved in internal memory in constant acquisition mode.

Parameter: None

Return value: Integer

≠ -1 - The last spectrum index

-1 - Operation failed

3.20 Get One Spectrum of a Line Number

BOOL DLL_Get_Constantly_Grab_Spectra(DWORD dwSpectrumIndex,
DWORD* pdwSpectrumLineStamp,
double* pdblData);

Description: Get one spectrum of given index in constant acquisition mode.

Parameter: DWORD dwSpectrumIndex: Spectrum Index saved in internal memory

DWORD* pdwSpectrumLineStamp: Spectrum line stamp generated by hardware for the spectrum with the index = dwLineNumber

double* pdblData:

Pointer of spectrum power array allocated by the user (Buffer size >= pixel count)

Return value: Boolean

TRUE: Operation completed

FALSE: Operation failed

3.21 Get All Peaks Information of One Spectrum at a Line Number

BOOL DLL_Get_Constantly_Grab_Peaks(DWORD dwSpectrumIndex,
DWORD* pdwPeakInfo);;

Description: Get all peaks information of one spectrum of given index in constant acquisition mode.

Parameter: DWORD dwSpectrumIndex: Spectrum Index in saved internal memory

DWORD* pdwPeakInfo: Pointer of searched spectrum peak information array allocated by the user (Buffer size >= 512)

Peak information format:

pdwPeakInfo[0] = Spectrum line stamp

pdwPeakInfo[1] = Peak count,

pdwPeakInfo[2] to pdwPeakInfo[128]

= Peak pixels or wavelengths × 1000

pdwPeakInfo[129] to pdwPeakInfo [255]

= Peak power × 1000

Return value: Boolean

TRUE: Operation completed

FALSE: Operation failed

3.22 Set Device Sensor Mode

BOOL DLL_Set_Sensor_Mode(WORD wSensorMode);

Description: Set device sensor mode

Parameter: WORD wSensorMode: 0 – High sensitive mode
1 – High dynamic range mode

Return value: Boolean
TRUE: Operation completed
FALSE: Operation failed

3.23 Get Device Sensor Mode Status

int DLL_Get_Sensor_Mode_Status();

Description: Get device's sensor mode status

Parameter: None

Return value: Signed integer
0: High sensitive mode
1: High dynamic range mode
-1: Operation failed

3.24 Set 1×4 Optical Switch State

BOOL DLL_Set_Optical_Switch (BOOL bAutoChannelScan, WORD wChannelIndex);

Description: Set 1×4 optical switch state

Parameter: BOOL bAutoChannelScan: TRUE - Auto scan channel 1 to 4 repeatedly during continue data acquisition
FALSE - Disable auto channel scan
WORD wChannelIndex: 1 - Channel 1
2 - Channel 2
3 - Channel 3
4 - Channel

Return value: Boolean
TRUE: Operation completed
FALSE: Operation failed

3.25 Get 1×4 Optical Switch Status

BOOL DLL_Get_Optical_Switch_Status(BOOL* pbAutoChannelScan,
WORD* pwChannelIndex);

Description: Get 1×4 optical switch status

Parameter: BOOL* pbAutoChannelScan: Pointer of current auto channel scan status
WORD* pwChannelIndex: Pointer of current selected channel index

Return value: Boolean
TRUE: Operation completed
FALSE: Operation failed

3.26 Set GPIO State

BOOL DLL_Set_GPIO_State (BYTE byteGPIOState);

Description: Set GPIO state

Parameter: BYTE byteGPIOState: Each bit for one GPIO pin state

Return value: Boolean
TRUE: Operation completed
FALSE: Operation failed

3.27 Get GPIO State

BOOL DLL_Get_GPIO_State(BYTE* pbyteGPIOState);

Description: Get GPIO state

Parameter: BYTE* pbyteGPIOState: Pointer of current GPIO state

Return value: Boolean
TRUE: Operation completed
FALSE: Operation failed