Spectral Products

Spectrometers · Spectrophotometers · Color Instruments · Spectrographs · Monochromators

111 Highland Drive · Putnam, CT · 06260 · USA
PHONE (860) 928-5834· FAX (860) 928-2676
http://www.spectralproducts.com

SM32Pro SDK

Spectrometer Operating

-Software Development Kit-

USER MANUAL

Table of Contents

Warranty And Liability

This SM product is warranted against defects in material and workmanship for a period of one year from the date of shipment. During the warranty period, **Spectral Products (SP)** will, without charge, repair or replace, at its discretion, the defective product or component parts.

For warranty service or repair, this product must be returned to a service facility designated by **SP**. For products returned under warranty, the Buyer shall prepay shipping charges (including shipping charges, duties, and taxes for products returned to **SP** from another country), and **SP** will pay for shipping charges to return the product to the Buyer.

This warranty does not apply in the event of misuse or abuse of the product or as a result of unauthorized alterations, modifications or repairs, if the serial number is altered, defaced or removed, the improper or inadequate maintenance by the Buyer, Buyer-supplied software or interfacing, operation outside of the environmental specifications for the product, or improper site preparation or maintenance. No other warranty is expressed or implied. **SP** shall not be liable for any consequential damages, including without limitation, damages resulting from loss of use, as permitted by law.

The Location of the SDK source code for USB 2.0

If the software was installed with the installation CD, the SDK source codes are located in the directory that the software was installed. If the user didn't change the installation directory and the Windows OS had been installed on "C:\" drive, the location of the SDK source codes is "C:\Program Files\Spectral Products\SM32Pro\SDK Examples\VC++_SDKs\SDK2XX-USB2.0" in general. The SDK examples can be downloaded from **SP**'s website (Click here).

In case of other interface users, please refer to the manual located in each SDK example directory. The SDK directory name indicates the interface type.

System Requirements

Requirements for the Software

- Any IBM compatible computer
- A hard drive with at least 50 MB free space
- A CD-ROM
- A VGA or compatible display
- 32 MB RMA or higher
- A mouse or other pointing device
- Microsoft Windows® 98, 98SE, ME, NT, 2000, and XP

Check System Package Contents

Check that your SM system package contains all the required components.

Common system packages contain the fallowing:

- Spectrometer
- USB 2.0 Cable
- Any accessories ordered.

Note: Package contents may vary from unit to unit and order to order. If you have any questions about the contents of your package, please contact the support team referred to in the back of this manual.

Getting Started

Welcome to the **Spectral Products**' SM32Pro Software Development Kit.

This kit documents a library of functions for accessing the USB 2.0/1.1 Data Acquisition Board used by the CCD detector units. Two fundamental CCD imaging concepts are the Collection of Data and Generating Meaningful Values from that data.

Collection of Data

In our array detector spectrometers, the light is dispersed across 512 (Hamamatsu, InGaAs), 1024 (Hamamatsu, Back-thinned TE cooled CCD), 2048 (Sony CCD or Hamamatsu back thinned CCD) or 3648 (Toshiba CCD) pixel detector array. Data is collected by each pixel and converted to a relative value by the analog-to-digital (A/D) converter ranging from 0 to 65535 and representing the intensity of the light at each pixel. We can control the amount of time that the pixels collect light and thus read signals of varying strengths. Using this library of functions, you will be able to adjust the signal capture time (Integration Time) and collect the converted signals from each pixel. The criteria to adjust the integration is to try different lengths of time with a reference signal representing the maximum possible signal level during the measurement until the peak value across the range of pixels is close to, but not at 65535. We recommend trying to get the peak at 65000 around. This way you can be sure that you are getting an optimized measurement condition with no saturation of CCD elements.

Generating Meaningful Values

The information you collect from the system is only the relative signal size at each detector pixel until it has been calibrated to absolute (certified reference) values. That is in the raw data, the pixel location represents spatial distribution of the CCD elements and the corresponding intensity is represented by a digital value. Since we can adjust the strength of the light source and the integration time of the CCD, this A/D value only provides a relative representation of the real world parameters.

We address both problems. We connect pixels to wavelengths by use of known emission lines (e. g. **SP**'s calibration light sources or narrow band filters) across the range of pixels together with a table relating pixels to wavelengths in nanometers. Curve fitting functions can then be applied to generate a polynomial function for the conversion of all other pixels to wavelengths.

We address the second problem in one of two ways. Both ways involve "normalization." Normalization involves measurements of signal strength based on its ratio with respect to reference signal intensity. If we are only interested in the spectral distribution of the sample signal, then we can normalize a sample signal scan to one reference value by, for example, intensities of all the array elements divided by the peak intensity. If we want to measure percent transmission, the light can be measured first with only air in the light path as 100% reference and then the sample can be inserted into the light path and a sample scan followed. Consequently, the divisions of sample scan readings by the 100% reference yields the relative transmission values, or percent transmission when multiplied by 100%. For reflectance measurement similar practice can be applied and a high reflector can be used as a reference air in many cases.

It is strongly recommended that a measurement be taken prior to any sample scan, in which no external light but only the detector noise will be sensed and subsequently subtracted from all the following measurements. This will subtract thermally generated “background” DC offset level and statistically establish a baseline of zero for all subsequent measurements.

Using the SP Libraries

For Windows C/C++, you must include the correct "include" file and "lib" file for the functions you are calling:

For Visual Basic, please include the correct "BAS" file. The function declarations listing in this file demonstrate the correct data types. To pass a "pointer" or array to a DLL from Visual basic, simply pass the first element of the array (IE SampleData[0]).

Please refer to the samples or give us a call if you need any further technical assistance.

Procedure for using the .dll with VC++

1. The SPdbUSBm.dll and SPdbUSBm.lib need to be copied to the work or release folder. Those are located in the "[Software installed directory]\SDK Examples\VC++_SDKs\SDK2XX-USB2.0\DLL".
2. Add SPdbUSBm.lib to the project/settings/link/object/library module.

General Overview

Some basic fundamentals of utilizing the SP SDK functions

First spTestAllChannels() has to be called to get the total number of channels that are currently connected to the computer. It returns the total number of channels connected. Then spSetupAllChannels() or spSetupGivenChannel() should be called to set up each USB port connection. The spInitAllChannels() or spInitGivenChannel() needs to be called to initialize the setup components and values. After consecutively calling the three functions above, you can call any function in the DLL.

Integration time is the time period the CCD pixels are exposed to light before the resulting charges are read out. A longer integration time can allow you to detect a lower light level signal. The longer your integration time is the more background signal will accumulate.

Using Curve Fitting to Calibrate SM32Pro

Curve fitting is used in SM32Pro/SDK to correlate the physical locations of pixels on the CCD with the known wavelength of the radiation falling on them.

This is done by identifying the pixel locations where the maximal of the known wavelength is at. These peak intensity wavelengths and pixels are used by spPolyCalc() and spPolyFit() to generate a correlating polynomial function which best represents all the data points. We have found that using a third order polynomial function produced the most desirable results for most cases. In cases where high dispersion elements are used, lower order polynomial functions may have to be utilized due to the limited known wavelengths available from the calibration lamps.

Calibration Files

Each unit's calibration set is included in the SM32Pro software settings, "SM32Pro.ini". This text file contains calibration data of the form "DataX=Wavelength;Pixel". The file also contains the regression coefficients "A_0 value", "A_1 value", … , "B_3 value" that satisfy the equations.

$$I_i = A_0 + A_1 P_i + A_2 P_{i\,2} + A_3 P_{i\,3}$$

$$P_i = B_0 + B_1 I_i + B\ 2\ I_{i\,2} + B_3 I_{i\,3} .$$

The A values are the coefficients for conversions from a pixel number to a wavelength in "nm" by use of the above first third order polynomial function. The B values allow the conversions from a desired wavelength in "nm" to a pixel number.

SDK Functions

Data Acquisition

spTestAllChannels

- Used to check up the total number of channels connected. The USB connection order type needs to be set in this function.

spSetupAllChannels
spSetupGivenChannel

- Used to set up and establish the USB connection in all channels or each channel.

spInitAllChannels
spInitGivenChannel

- Used to initialize the components and values. The total pixel number of the array detector, the detector type (Sony CCD, Toshiba CCD, or Hamamatsu back thinned CCD/InGaAs) and the initial integration time are set in this function.

spSetTrgEx

- Used to set triggering type. The external triggering option can be set in this function.

spSetIntEx

- Used to set the detector integration time.

spSetTEC (SM303-Si/SM303-InGaAs Only)

- Used to turn the TE cooling on/off.

spSelectCF (SM303-InGaAs Only)

- Used to set the capacity size (1pF/10pF).

spReadDark
spReadDataEx
spReadDataAdvEx
spReadDataExOutTrg

- Used to collect spectral data from the detector.

spCloseAllChannels
spCloseGivenChannel

- Used to close the USB connection. This function should be called at the exiting of program.

Channel Configuration

spReadChannelID

- Used to read the channel ID assigned and saved in the EEPROM on the USB board.

spWriteChannelID

- Used to write a new channel ID on the EEPROM on the USB board.

spGetAssignedChannelID

- Used to get the information of all channel IDs assigned.

Signal Processing

spSetBaselineCorrection

- Used to flatten the baseline of the detector signals to improve the fixed pattern noise level.

spSetOffsetCorrection

- Used to correct the offset value of the baseline against the reference optical black pixels.

Calibration

spPolyFit

- Used to generate a calibration function from pixels to wavelength. This function calculates the coefficients for a polynomial curve fitting function given an array of independent variables and a corresponding array of dependent variables.

spPolyCalc

- Used for determining the wavelength for a given pixel location. This function calculates a polynomial function given the independent variable and a coefficient array.

SDK Function Descriptions

spTestAllChannels

```
short spTestAllChannels
(
        short sOrderType = SP_ORDER_BY_CHANNELID
                        // The order of USB connection:
                        // SP_ORDER_BY_CHANNELID or
                        // SP_ORDER_BY_USBPORTNUM
)
```

This function is used to test and check the connection of USB boards.

sOrderType is the type of the USB port connection order. If the user wants to define each channel by the order of the USB port connection, set the **sOrderType** as "SP_ORDER_BY_USBPORTNUM" which is "0". If the user wants to define each channel by the assigned channel ID, set the value as "SP_ORDER_BY_CHANNELID" which is "1".

The default is "1" i.e. SP_ORDER_BY_CHANNELID. When the "SP_ORDER_BY_CHANNELID" was selected but the USB boards connected don't have proper channel IDs assigned, this function will reassign the channel ID as the same as the USB port number of each USB board.

RETURN

If the function works properly, it will return the total number of channels connected. If not, it will return a negative number.

spSetupAllChannels

```
short spSetupAllChannels
(
)
```

This function is used to set up and check the connections of all USB boards at once.

RETURN

If there is no error, it will return a positive value (SP_NO_ERROR). If not, a negative one.

spSetupGivenChannel

```
short spSetupGivenChannel
(
        short sChannel = 0      // The channel ID or USB port number
)
```

This function is used to set up and check the connection of the given USB board.

sChannel could be the assigned channel ID or the USB port number according to the setting of the spTestAllChannels() function.

RETURN

If there is no error, it will return a positive value (SP_NO_ERROR). If not, a negative one.

spInitAllChannels

```
short spInitAllChannels

(

        short sCCDType = SP_CCD_SONY // the type of CCD installed in the
                                     // spectrometer
)
```

This function is used to initialize all USB boards connected at once.

sCCDType indicates what kind of detector is installed in the spectrometer. If the detector is Sony ILX511 (SM2xx series), then it should be "SP_CCD_SONY (0)", if Toshiba TCD1304AP (SM4xx series), then "SP_CCD_TOSHIBA (1)", if Hamamatsu Back-thinned TE cooled CCD S7031-1006 (SM303-Si), then "SP_CCD_PDA (2)", if Hamamatsu InGaAs array G9212 (SM303-InGaAs), then "SP_CCD_G9212 (3)", and if Hamamatsu Back thinned CCD S10420 (SM642), then "SP_CCD_S10420 (4)".

When all spectrometers have the same type of CCD, use this function to initialize.

RETURN

If there is no error, it will return a positive value (SP_NO_ERROR). If not, a negative one.

spInitGivenChannel

```
short spInitAllChannels
(
        short sCCDType = SP_CCD_SONY, // the type of CCD installed in the
                                       // spectrometer
        short sChannel = 0    // The channel ID or USB port number
)
```

This function is used to initialize the given USB board.

sCCDType indicates what kind of detector is installed in the spectrometer. If the detector is Sony ILX511, then it should be "SP_CCD_SONY" which is 0, if Toshiba TCD1304AP, then "SP_CCD_TOSHIBA" which is 1, if Hamamatsu Back-thinned CCD, then "SP_CCD_PDA" which is 2, if Hamamatsu InGaAs array, then "SP_CCD_G9212" which is 3, and if Hamamatsu Back thinned CCD S10420, then "SP_CCD_S10420" which is 4.

sChannel could be the assigned channel ID or the USB port number according to the setting of the spTestAllChannels() function.

Use this function to initialize each USB board separately when each spectrometer has a different CCD.

RETURN

If there is no error, it will return a positive value (SP_NO_ERROR). If not, a negative one.

spSetIntEx

```
short spSetIntEx
(
        long const lIntTime,    //new integration time
        short sChannel = 0    // The channel ID or USB port number
)
```

This function is used to set and change the integration time.

lIntTime is amount of time in milliseconds for Sony CCD or Hamamatsu PDA detector or in 10 microseconds for Toshiba CCD to set as a new integration time. This number should range between 1 and 65535 for Sony CCD or Hamamatsu InGaAs detector, between 7 and 65535 for Hamamatsu back-thinned CCD and between 1 and 6553500 for Toshiba CCD. In the case of

Toshiba CCD, the minimum integration time is 10 microseconds and the value setting is based on 10 microseconds. For example, **lIntTime** of "100" is 1 millisecond. To avoid the misleading, it is recommended for Toshiba CCD spectrometer users to use the **spSetDblIntEx()** function for setting the integration time.

sChannel could be the assigned channel ID or the USB port number according to the setting of the spTestAllChannels() function.

RETURN

If there is no error, it will return a positive value (SP_NO_ERROR). If not, a negative one.

spSetDblIntEx

```
short spSetDblIntEx
(
        double const dIntTime,          //new integration time
        short sChannel = 0              // The channel ID or USB port number
)
```

This function is used to set and change the integration time.

dIntTime is amount of time in milliseconds to set as a new integration time. This number should range between 0.01 and 65535.0 for Toshiba CCD, between 1.0 and 65535.0 for Sony CCD or Hamamatsu InGaAs detector and between 7.0 and 65535.0 for Hamamatsu back-thinned CCD. In the case of Toshiba CCD, the value will be rounded off to two decimal places. In the case of other detectors, all numbers below decimal point will be off so it is recommended using the **spSetIntEx()** function rather than this one.

sChannel could be the assigned channel ID or the USB port number according to the setting of the spTestAllChannels() function.

RETURN

If there is no error, it will return a positive value (SP_NO_ERROR). If not, a negative one.

spSetTrgEx

```
short spSetTrgEx
(
        short const sTrgMode //Sets triggering mode
        short sChannel = 0      // The channel ID or USB port number
)
```

This function is used to set and change the triggering mode.

sTrgMode is used to set the trigger mode. If this value was set as “SP_TRIGGER_OFF” which is 10, the USB board runs freely. If an external triggering is needed, this value should be set as “SP_TRIGGER_EXTERNAL” which is 12. When the external trigger mode needs to be release to make the USB board run freely, set this value as “SP_TRIGGER_INTERNAL” which is 11. The default value is “SP_TRIGGER_OFF”. If the trigger mode doesn’t need to be set, then do not call this function.

sChannel could be the assigned channel ID or the USB port number according to the setting of the spTestAllChannels() function.

RETURN

If there is no error, it will return a positive value (SP_NO_ERROR). If not, a negative one.

spSetTEC (NOTE: SM303-Si/SM303-InGaAs Only)

```
short spSetTEC
(
        long lTEC               //Sets TE Cooling On/Off
        short sChannel = 0      // The channel ID or USB port number
)
```

This function is used to turn the TE Cooler on or off.

lTEC is used to set the TE Cooling On/Off. If this value was set as “1”, the USB board turns on the TE Cooling. If it is “0”, the TE Cooling mode will turn off.

sChannel could be the assigned channel ID or the USB port number according to the setting of the spTestAllChannels() function.

RETURN

If there is no error, it will return a positive value (SP_NO_ERROR). If not, a negative one.

spSelectCF (NOTE: SM304 series Only)

```
short spSelectCF
(
        long lCF                    //Sets Capacity value as 1pF or 10pF
        short sChannel = 0      // The channel ID or USB port number
)
```

This function is used to select the capacity value for InGaAs array detector.

lCF is used to set the capacity value of the detector. If this value was set as "1", the capacity value will be 10pF. If it is "0", the capacity value will be 1pF. 1pF capacity will give higher sensitivity but less stable signal and 10pF vice versa.

sChannel could be the assigned channel ID or the USB port number according to the setting of the spTestAllChannels() function.

RETURN

If there is no error, it will return a positive value (SP_NO_ERROR). If not, a negative one.

spReadDataEx:

```
short spReadDataEx
(
        long * pArray;      // The array in which spectral data is stored
        short sChannel = 0      // The channel ID or USB port number
)
```

This function is used to read the CCD signal data through USB board.

pArray points to a read memory address. Its memory size should be larger than (or at least the same as) the detector pixel number defined at **spInitAllChannels or spInitGivenChannel**. The total CCD pixel number to be read is one of "SP_CCD_PIXEL_PDA" which is 1056, "SP_CCD_PIXEL_G9212" which is 512, "SP_CCD_PIXEL_SONY" which is 2080, "SP_CCD_PIXEL_S10420" which is 2080, or "SP_CCD_PIXEL_TOSHIBA" which is 3680. Some detector contains the dummy pixels or optical blank pixels. The real data pixel number is "SP_CCD_PIXEL_PDA_REAL" which is 1024, "SP_CCD_PIXEL_G9212_REAL" which is 512, "SP_CCD_PIXEL_SONY_REAL" which is 2048, "SP_CCD_PIXEL_S10420_REAL" which is 2048, or "SP_CCD_PIXEL_TOSHIBA_REAL" which is 3648. In case of Sony and Toshiba CCDs, the first 32 pixels are optical blank and in case of Hamamatsu back-thinned CCDs, the first 10 pixels are optical blank. This function returns just raw data. Some detectors may give reversed data. The SM303-InGaAs give left-right mirrored data so those have to be re-mirrored. Also due to the intrinsic property of the InGaAs array detector's response, all output signals are revered based on the A/D resolution so it has to be flipped over like data_real = 65535-data_read, where 65535 represents "16bit" A/D resolution and the "data_read" is the raw data.

sChannel could be the assigned channel ID or the USB port number according to the setting of the spTestAllChannels() function.

RETURN

If there is no error, it will return a positive value (SP_NO_ERROR). If not, a negative one.

spReadDataAdvEx:

```
short spReadDataAdvEx
(
        long * pArray;          // The array in which spectral data is stored
        short sTimeAve,         // Time averaging
        short sFFT              // FFT Smoothing
        short sBadPxlNum,       // Total number of bad pixel
        short *spBad,           // Bad pixel location information
        short sChannel = 0      // The channel ID or USB port number
)
```

This function is used to read the detector signal data through USB board in advanced mode.

pArray points to a read memory address. Its memory size should be larger than (or at least the same as) the detector pixel number defined at **spInitAllChannels or spInitGivenChannel**. The total CCD pixel number to be read is one of "SP_CCD_PIXEL_PDA" which is 1056, "SP_CCD_PIXEL_G9212" which is 512, "SP_CCD_PIXEL_SONY" which is 2080, "SP_CCD_PIXEL_S10420" which is 2080, or "SP_CCD_PIXEL_TOSHIBA" which is 3680. Some detector contains the dummy pixels or optical blank pixels. The real data pixel number is "SP_CCD_PIXEL_PDA_REAL" which is 1024, "SP_CCD_PIXEL_G9212_REAL" which is 512, "SP_CCD_PIXEL_SONY_REAL" which is 2048, "SP_CCD_PIXEL_S10420_REAL" which is 2048, or "SP_CCD_PIXEL_TOSHIBA_REAL" which is 3648. In case of Sony and Toshiba CCDs, the first 32 pixels are optical blank and in case of Hamamatsu back-thinned CCDs, the first 10 pixels are optical blank. In this function, all data conversion will be done in this function so they can be used as they are without any additional treatment.

sTimeAve is for setting the total number for time averaging to get more stable and less noisy data. The data will be gathered for a given number and return as average. Its default is "1".

sFFT is FFT Smoothing value. Filter high frequency noise in the frequency domain to reduce dark noise. As this value increases, the spectrum with a narrow bandwidth is distorted.

In general, dark noise can be improved in a countinuous wavelength light source having a wide bandwidth. The range of value is 0 to 100, and the default is "0".

sBadPxlNum is used to set the total number of bad pixels. Due to the limitation in the current state-of-art for manufacturing array InGaAs and/or back thinned CCD, a few (<5) pixels could be bad ones. Even regular CCDs may generate some bad pixels when used for long time. Its default is “0”.

spBad points to a bad pixel information. Each value indicates the location of the memory address, which uses zero-based numbering. Also each detector has its own optical blank pixels before active ones so the total optical blank pixel number has to be added. For example, if there is one bad pixel in the back-thinned CCD detector and its location is the 243rd pixel in the active array, the value has to be “243 – 1 + 10” where “-1” is for converting 1-based numbering to 0-based one and “+10” is for compensating the optical blank pixels of the back thinned CCD, which is “10”. Its default is “NULL”.

sChannel could be the assigned channel ID or the USB port number according to the setting of the spTestAllChannels() function.

RETURN

If there is no error, it will return a positive value (SP_NO_ERROR). If not, a negative one.

spCloseAllChannels

short spCloseAllChannels
(
)

This function is called to close all USB board connections. It should be called when exiting the application.

RETURN

If there is no error, it will return a positive value (SP_NO_ERROR). If not, a negative one.

spCloseGivenChannel

short spCloseGivenChannel

```
(
        short sChannel = 0      // The channel ID or USB port number
)
```

This function is called to close the given USB board connection.

sChannel could be the assigned channel ID or the USB port number according to the setting of the spTestAllChannels() function.

RETURN

If there is no error, it will return a positive value (SP_NO_ERROR). If not, a negative one.

spReadChannelID

```
short spReadChannelID
(
        short *spNum,           // The pointer that the Channel ID will be saved in
        short sChannel = 0      // The channel ID or USB port number
)
```

This function is called to read the channel ID of the given USB board.

spNum is the pointer that the assigned channel ID of the USB board will be saved in.

sChannel could be the assigned channel ID or the USB port number according to the setting of the spTestAllChannels() function.

RETURN

If there is no error, it will return a positive value (SP_NO_ERROR). If not, a negative one.

spWriteChannelID

```
short spWriteChannelID
(
        short sNum,             // The Channel ID to save
        short sChannel = 0      // The channel ID or USB port number
)
```

This function is called to write the channel ID on the given USB board.

sNum is the new channel ID to be saved on the USB board.

sChannel could be the assigned channel ID or the USB port number according to the setting of the spTestAllChannels() function.

RETURN

If there is no error, it will return a positive value (SP_NO_ERROR). If not, a negative one.

spGetAssignedChannelID

```
void spGetAssignedChannelID
(
        short *spChannelID,             // The pointer of the Channel ID assigned
)
```

This function is called to get the assigned channel ID information.

It is only available when the "SP_ORDER_BY_CHANNELID" was set on the **spTestAllChannels()** function.

spChannelID is the pointer of the assigned channel ID information. The memory of this pointer has to be allocated before using this function. The total memory size must be the same as the total USB connection number which is returned by the **spTestAllChannels()** function.

RETURN

None.

spPolyFit

```
short spPolyFit
(
        double *x,      // Array of independent variables
        double *y,      // Array of dependent variables
        short numPts,  // Number of points in independent and dependent arrays
        double *coefs, // Pointer to array to hold calculated coefficients [index: 0 - order]
        short order     //  Order of polynomial
)
```

This curve fitting function is used to find a polynomial function to calculate the wavelength of a given pixel. This function is used for calibration purposes. Either a calibration light source or a series of narrow band filters are scanned and the pixel locations of all known peaks are identified along with the known wavelength at that peak. These peak locations and wavelengths are stored in the arrays x and y, respectively. The arrays indices should range from 0 to [Number_of_Points - 1]. They are passed to the function along with a requested order for the polynomial fitting function and an array large enough to hold the coefficients (). This array is then used with **spPolyCalc** to calculate wavelength from pixels.

x is an array containing the independent variables. It should range from 0 to (numPts-1).
y is an array containing the dependent variables. It should range from 0 to (numPts-1).
numPts is the number of points in the variable arrays.
coefs is a pointer to the array that will contain the polynomial coefficients. It should range from 0 to (order-1).
order is the desired order of the polynomial. We have determined third order to be the optimum for wavelength calibration for most cases.

RETURN

This function will return 1 if the function is successful. Otherwise it will return negative.

spPolyCalc

```
void spPolyCalc
(
        double *coefs,
        short order,
        double x,
        double *y
)
```

This function calculates for the following formula:

$y = a_0 + a_1*x^1 + a_2*x^2 + ... + a_N*x^N$, where * specified multiplication.

coefs is a pointer to an array containing the polynomial coefficients. These can be calculated using **spPolyFit**.

order specified the order of the polynomial equation and must be less than or equal to the number of elements in coefs.

x is the independent variable, in this case, the pixel number.

y is the value to be calculated.

RETURN

None

spGetDevIsNew

```
short spGetDevIsNew

(

        short sChannel = 0
)
```

This is a function that checks to see if the connected device is a newer version of the device. The new version adds new features that are not supported in older versions, so you can use this function to apply new API functions only to the new version.

RETURN

0 means old device, 1 means new device.

spSetIntMode

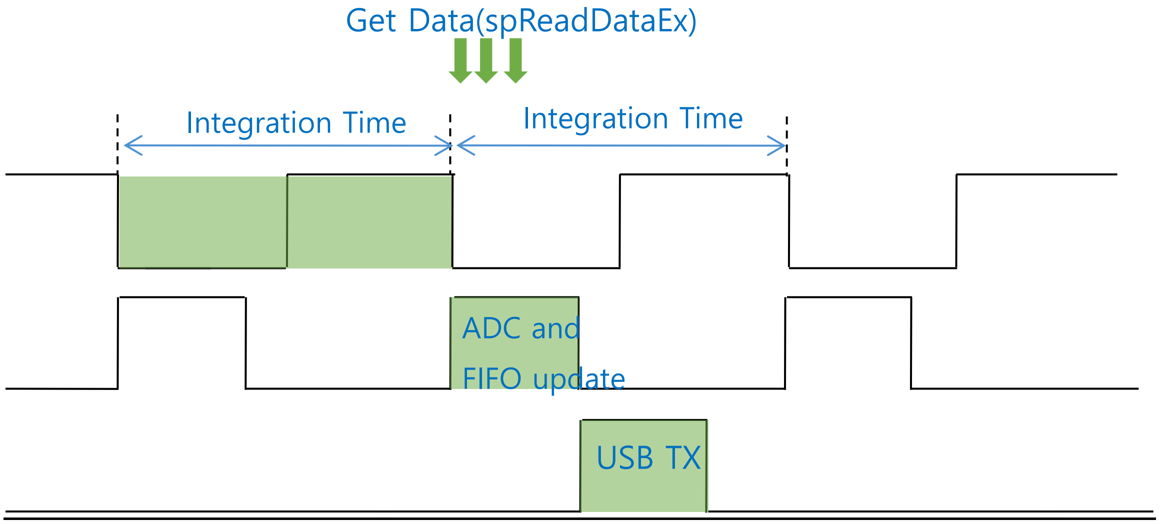
```
short spSetIntMode
(
        short ReadMode = SP_INTMODE_SWTRIGGER,
        const double dIntTime,
        short sChannel = 0
)
```

This function is to set the operation method in Internal Trigger Mode, and it is only available in the new version of SM245 & SM642 Spectrometer.
The operation mode operates in three modes: Software Trigger, Free run Previous, and Free run Next. SM642 supports all operation modes, whereas the SM245 does not support Free run Next mode. If SM245 is set to Free run Next mode, it will be set to Free run Previous mode automatically in the function.

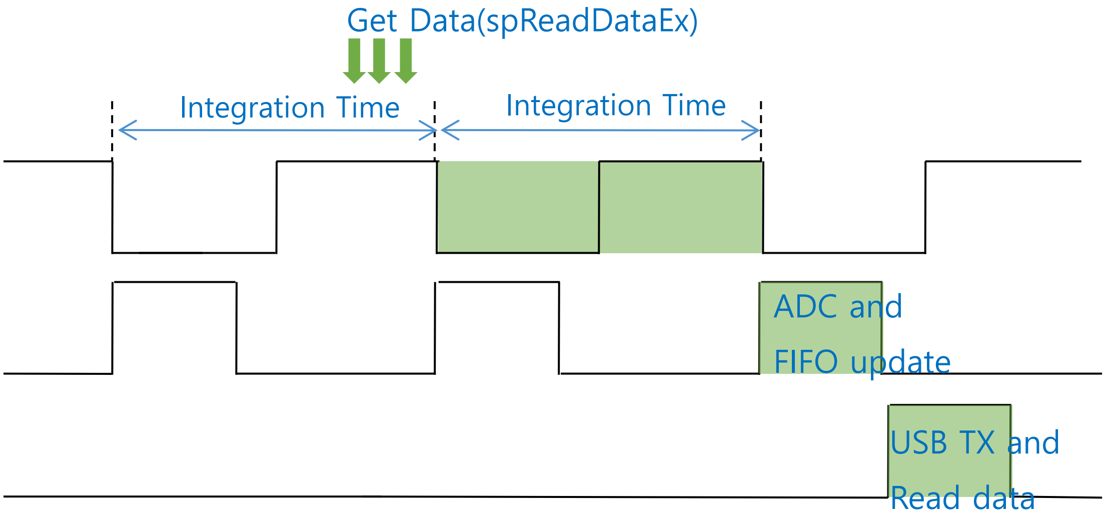
RETURN
SP_ERRPR_NOTSUPORT_DEV (-7): No corresponding function behavior

Timing chart for each mode.

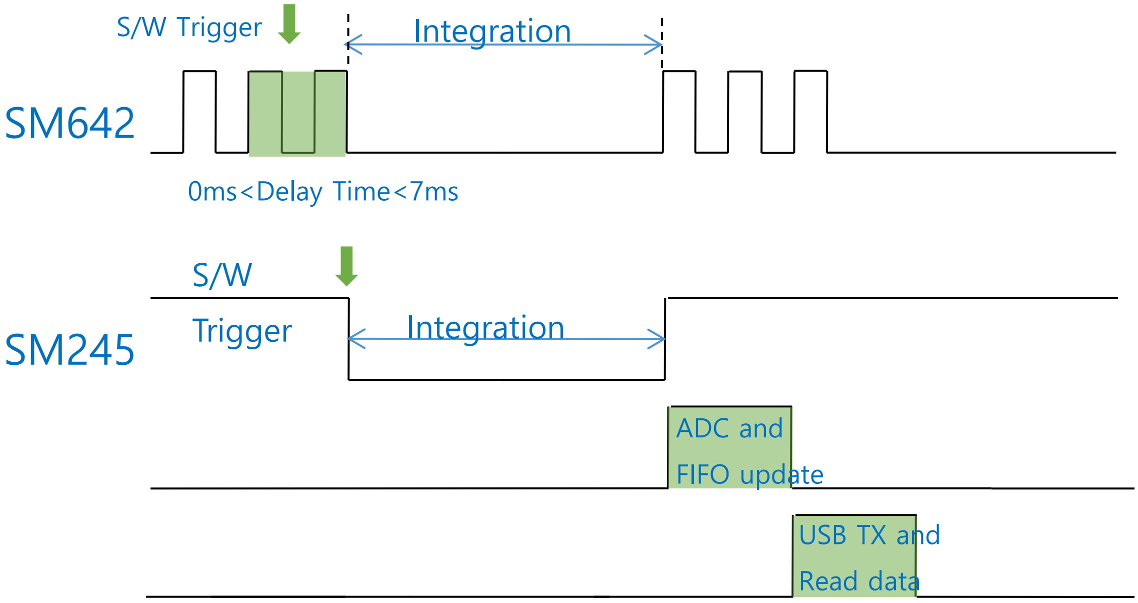
1. Free run Previous (SP_INTMODE_FREERUN_PREV = 1)



2. Free run Next(SP_INTMODE_FREERUN_NEXT = 2)



3. Software Trigger(SP_INTMODE_SWTRIGGER = 0)



4. SM245, SM642 Comparison Table

		SM245	SM642
Trigger Mode	Free run Prev	O	O
	Free run Next	X	O
	S/W Trigger	O	O
Minimum Integration Time		Free run Mode : 7ms S/W Trigger Mode : 1ms	All modes : 7ms
ADC and FIFO update		5ms	6ms
USB Tx Time		Within 3ms	Within 3ms

spSetExtEdgeMode

```
short spSetExtEdgeMode
(
        short sExtTriggerMode = SP_EXTEDGE_FALLING,
        short sChannel = 0
)
```

This function sets the interrupt method in Hardware Triggering Mode.
SP_EXTEDGE_FALLING = 0 (falling edge)
SP_EXTEDGE_RISING = 1 (rising edge) method is used and default is SP_EXTEDGE_FALLING.

RETURN

An error will occur when it is less than 0.

spDevInfo

```
short spDevInfo
(
        char *strModel,
        char *strSerial,
        short sChannel = 0
)
```

This function gets the model name and serial number of the connected device.
Returns the Model name in strModel and Serial Number in strSerial.

RETURN

An error will occur when it is less than 0.

spGetModel

```
short spGetModel
(
        short sChannel = 0
)
```

This function returns the Model Type of the connected device.

RETURN

An error will occur when it is less than 0.
0: SM2XX
1: SM440
2: SM303
3: SM304
4 : SM642

spGetWLTable

```
short spGetWLTable
(
        double *WLTable,
        short sChannel = 0
)
```

This function gets the Wavelength Table of the connected device. Simply use it to import the Wavelength Table. Cal data saved in EEPROM is loaded and polynomial calculation is performed automatically. Returns the Wavelength Table to the WLTable.

RETURN

An error will occur when it is less than 0.

spInsShutter

```
short spInsShutter
(
        short sChannel = 0
)
```

This function checks whether the connected device has a built-in shutter.

RETURN

An error will occur when it is less than 0.
0: Built-in shutter X
1 : Built-in shutter O

spSetShutterPos

```
short spSetShutterPos
(
        short sShutter = 0,
        short sChannel = 0
)
```

This function sets the position of the shutter.
Open shutter is 0, Closed shutter when 1.
Default is 0 (open).

RETURN

An error will occur when it is less than 0.

spGetShutterPos

```
short spGetShutterPos
(
        short sChannel = 0
)
```

This function returns the position of the shutter.

RETURN

An error will occur when it is less than 0.
0: Open
1: Close

spReadNLCDataEx

```
short spReadNLCDataEx
(
        short sChannel = 0
)
```

This function outputs nonlinearly corrected data. When the spectrometer is initialized, it is corrected by matching the lookup table calculated using the NLC coefficients in the EEPROM. Before use, dark data must be collected using the spReadDark Function.

RETURN

An error will occur when it is less than 0.

spBadPixelInfo

```
short spBadPixelInfo
(
        short *sBadPixelsCnt,
        short *sBadPixels,
        short sChannel = 0
)
```

This function outputs bad pixel information from EEPROM. Output the total number and pixel numbering.

sBadPixelCnt is used to get the total number of bad pixels. Due to the limitation in the current state-of-art for manufacturing array InGaAs and/or back thinned CCD, a few (<5) pixels could be bad ones. Even regular CCDs may generate some bad pixels when used for long time. Its default is “0”.

spBadPixels points to a bad pixel information. Each value indicates the location of the memory address, which uses zero-based numbering.

RETURN

An error will occur when it is less than 0.

spSetBaseLineCorrection

```
short spSetBaseLineCorrection
(
        bool bApply
        short sOffset
        double dGain
        short sChannel = 0
)
```

This function flattens the baseline of the detector. When this function is activated, it removes the pattern noise of the signal and makes the baesline flat. Before using this function, the **'spReadDark'** function should be called first to measure the dark signals first. Calling the **'spSetIntEx'** function disables this function and the detector signal data will go back to the raw data.

bApply Activated or Not.
sOffset is the baseline offset setting value. It can be set from “1” to “10000”.
dGain is the setting value of the amplification value of the signal. Only the actual change amount excluding the pattern noise of the signal is amplified by the sGain magnification. It can be set from “1.0” to “5.0”.

RETURN
An error will occur when it is less than 0.

spSetOffsetCorrection

```
short spSetOffsetCorrection
(
        bool bApply,
        short sChannel = 0
)
```

This function adjusts the baseline offset against the reference optical black pixels of the detector. Detector signals will be automatically adjusted to remove the strange offset of the baseline, which could be caused by an ambient electric/electronic sharp noise. Before using this function, the **'spReadDark'** function should be called first to set the dark baseline level. Calling the **'spSetIntEx'** function disables this function.

bApply Activated or Not.

RETURN

An error will occur when it is less than 0.

spReadDark

```
short spReadDark
(
        long *DarkArray,
        short sChannel = 0
)
```

This function collects dark data to set the baseline level at the dark condition. The collected dark baseline will be saved in the **DarkArray**. Calling the **'spSetIntEx'** function disables this function.

RETURN

An error will occur when it is less than 0.

spReadDataExOutTrg

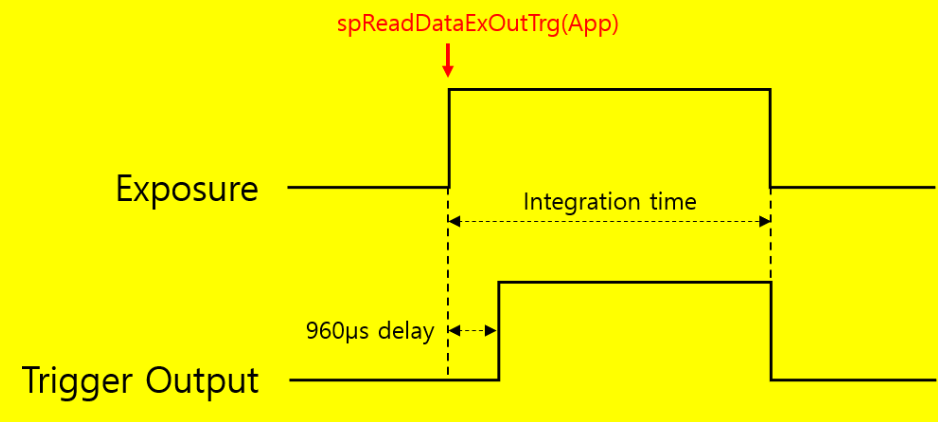
```
short spReadDataExOutTrg
(
        long * pArray;       // The array in which spectral data is stored
        short sChannel = 0      // The channel ID or USB port number
)
```

This function is the same as ‘**spReadDataEx’**, except it generates an additional output trigger signal during the detector exposure time.
Refer to the ‘**spReadDataEx’** function for how to use this function.
When you call this function in the software trigger mode, the external trigger signal will be generated right after 960μs from the start of the integration time. The external signal output pin can be set by calling the **'spSetOutTrgPin'** function before using this function.

RETURN

If there is no error, it will return a positive value (SP_NO_ERROR). If not, a negative one.

※ **Trigger Output Signal Timing**



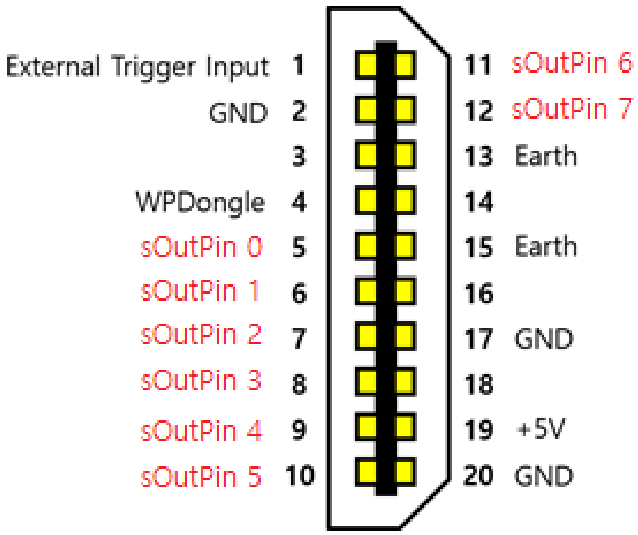
spSetOutTrgPin

```
short spSetOutTrgPin
(
        short sOutPin;          // The trigger output pin
        short sChannel = 0      // The channel ID or USB port number
)
```

This function sets the output pin number for sending out the external trigger signal when the **'spReadDataExOutTrg'** function is called.

sOutPin is the trigger signal output pin number. It can be from "**0**" (pin **#5**) to "**7**" (pin#**12**).

※ Pinout Diagrams for spectrometers


External Trigger Input 1
GND 2
3
WPDongle 4
sOutPin 0 5
sOutPin 1 6
sOutPin 2 7
sOutPin 3 8
sOutPin 4 9
sOutPin 5 10
11 sOutPin 6
12 sOutPin 7
13 Earth
14
15 Earth
16
17 GND
18
19 +5V
20 GND