SDK User Manual for HPS-CF Series Chromatic Confocal Sensors

目录

I. Introduction of SDK

SDK provides application program interfaces for the HPS-CF series chromatic confocal sensors, which are available for the development of Windows x64 platform. This SDK is a secondary development kit tool that provides interfaces containing a vast majority of operating instructions for the chromatic confocal sensors developed by our company. Please read through the user manual for details.

II. Integration of SDK into IDE

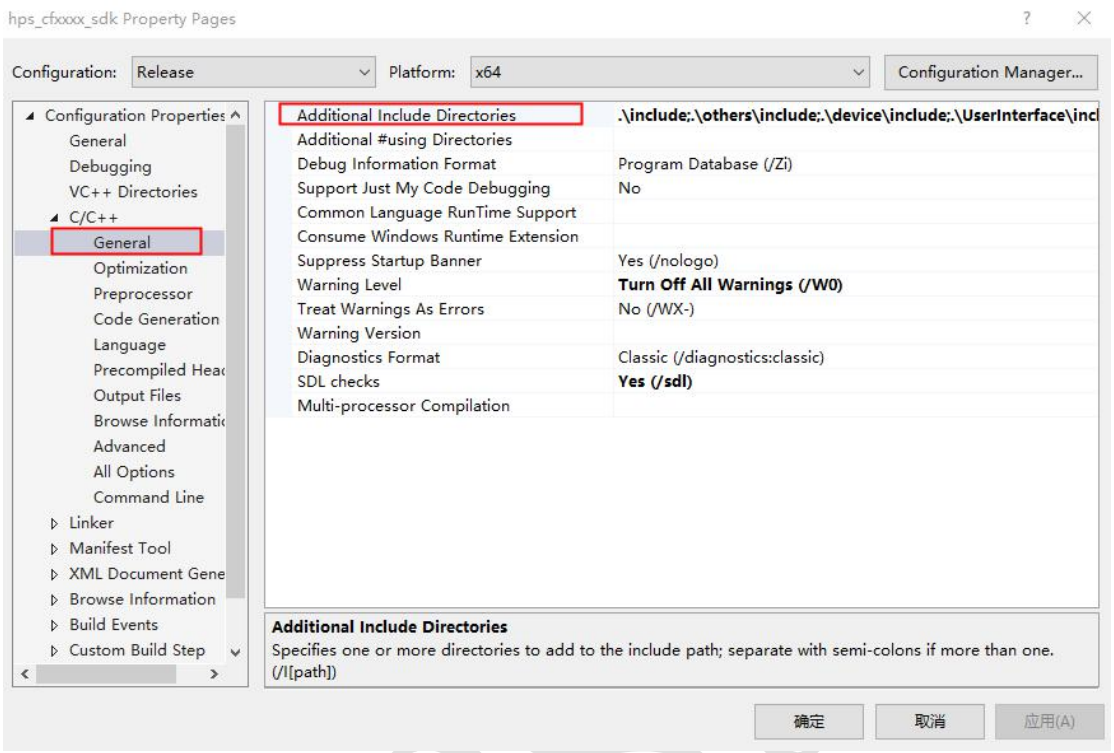
Here below is the integration of SDK in the platform Microsoft Visual Studio

2.1 Environment configuration in the Visual Studio platform and integration into IDE

xxx.lib and xxx.dll are suitable for use in the Windows operating system platform. Here we take the VS2015 environment as an example.

2.1.1 Configuration and integration of engineering environment

1. Add the included path of the header file
   a. Copy Type Define. h and User Interface .h to the path specified by the user.
   b. Click on Item - Attributes - C/C++ - General. Specify the path of the header file in the additional included directory.

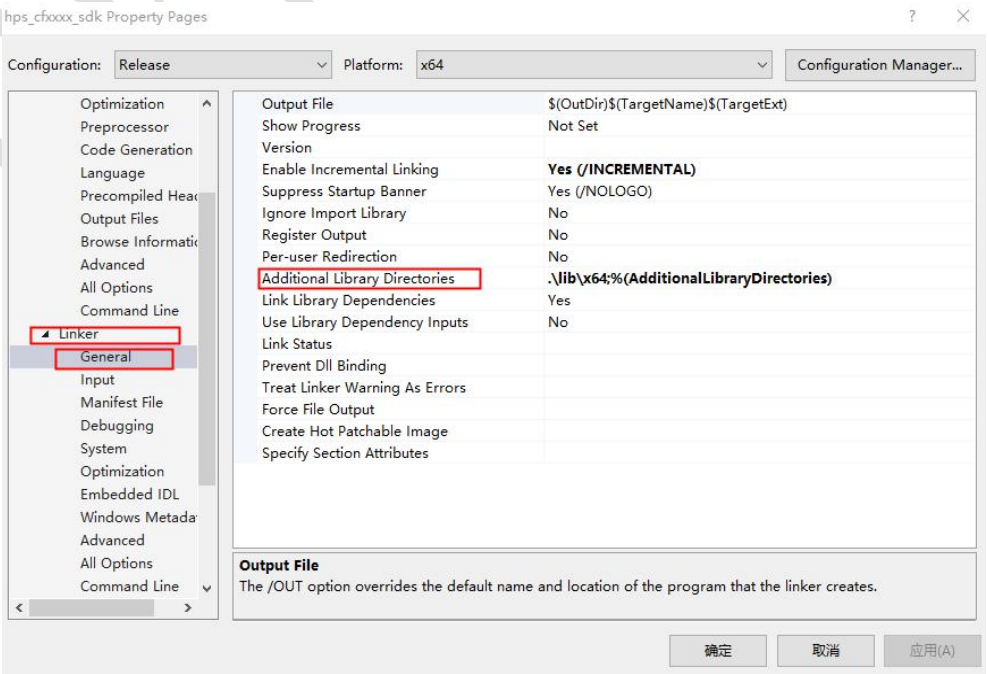


2. Add the library reference and the library path

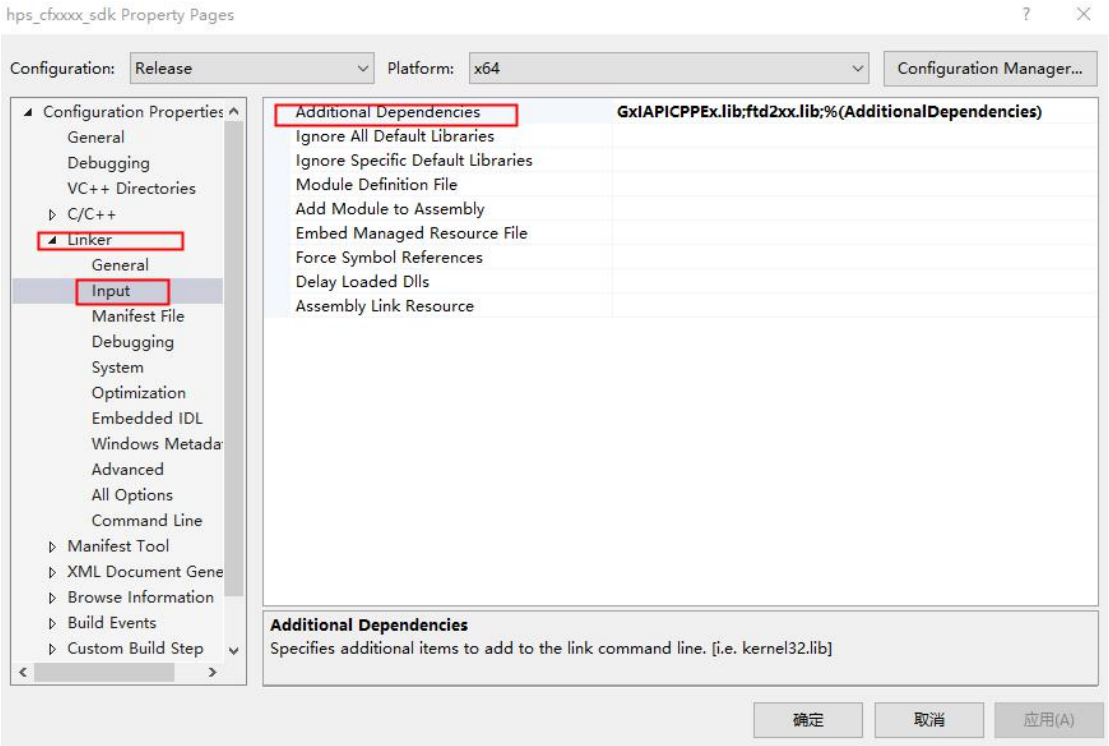
a. Select the SDK file in the x64 or x86 directory according to the user's Visual Studio platform. Here we take x64 as an example.

b. Copy hps_cfxxxx_sdk.dll and cf_windows_3156557.dll under the x64 directory to the directory where the program runs; copy hps_cfxxxx_sdk.lib to the path specified by the user.

c. Click on Item - Attributes - Linker - General. Specify the path of hps_cfxxxx_sdk.lib in the additional library directory



d. Click on Item - Attributes - Linker - Input. Enter hps_cfxxxx_sdk.lib in the additional dependency option.



2.1.2 Use of SDK in the user project

Here below are simple steps to get sensor data. Please refer to the sample codes in SDK Sample for specifics.

1. Connect the device to USB3.x interface at PC end and press the power button on the controller. The chromatic confocal sensor currently support USB3.0 or above interface only. Connection to USB2.0 interface may lead to communication failure.

2. Header files are contained in the project. Example code:

```
#include "UserInterface.h"
```

3. Perform the following operations after setting and scan the present number of devices:

```
DeviceHandle_t g_handle;
DeviceInfo_t* devList = NULL;
StatusTypeDef state;
int deviceNumber = 0;
scanDeviceList(&devList, &deviceNumber);
if (deviceNumber == 0)
{
    printf("no device found\n");
    return -1;
}
```

4. Call and open the device function to initialize the device. This step must be performed after Step 3 is successful. Example code:

```
/*Open the first device in the device list and return the operating handle
of the device*/
```

```
state = openDevice(&devList[0], &g_handle);
if (state != Status_Succeed)
{
    printf("Failed to openDevice%d\n",state);
    return -1;
}
printf("Device opened successfully\n");
```

5. Register event callback function:

```
/*eventCallback is an event callback function and the second Parameters are data passed on by users. If not required, pass in NULL*/
registerEventCallback(eventCallback, NULL);
```

callback function. The sample code is as follows:

```
void eventCallback(DeviceHandle_t handle, EventCallbackArgs_t arg,
void*userPara)
{
    SC_ResultDataTypeDef_t* res;

    if (handle == g_handle) //Judge whether or not the device is a user-designated device
    {
        //Data is received. It is not advisable to conduct complicated operations here, what should be done is to just copy the data away
        if (arg.eventType == EventType_DataRecv)
        {



          // Data type
           if (arg.rid == RID_RESULT)
           {
               //arg.dataLen: data of how many channels res[i]: Data of Channel i. If there is only one channel, arg.dataLen is 1
               res = (SC_ResultDataTypeDef_t*)arg.data;
               //Read measurement results
               //Read the measurement results of the channel. By default, when the multi-distance measurement mode is not enabled, only result[0] data is valid
                g_channel = res[0].channelIndex;   //Channel index corresponding to the result
               g_measureData = res[0].result[0];  //Measurement result
               g_saturation = res[0].saturation;//Saturation
               if (g_measureData == INVALID_VALUE) //No signal and invalid value output
               {
               }
```

```
            }
            else if (arg.rid == RID_DEVICE_DISCONNECT)   //Device disconnected
            {
                printf("Device disconnected\n");
             }
            else if (arg.rid == RID_API_CALL_EXCEPTION)   //API call exception
            {
                const char* err = (const char*)arg.data;
                printf("API call exception, Description%s\n",err);

            }
        }
    }
}
```

6. Configuration of the sensor

```
    //Enable Channel 0 Automatic light intensity adjustment
    state = setAutoLightIntensity(g_handle, 0, true);
    if (state != Status_Succeed)
    {

printf("setAutoLightIntensity->Description:%s\n",getErrorText(g_handle
));
        return -1;
    }
    //Set the exposure time to be 100us
    state = setExposureTime(g_handle, ExposureTime_100);
    if (state != Status_Succeed)
    {

printf("setExposureTime->Description:%s\n",getErrorText(g_handle));

        return -1;
    }
```

7. Start the continuous measurement

```
    //Start the continuous sampling
    state = continueCaptureStart(g_handle);
    if (state != Status_Succeed)
    {

printf("continueCaptureStart->Description:%s\n",getErrorText(g_handle)
);
```

```
        return -1;
    }
```

III. API function interface

3.1 Scan the list of sensor devices

scanDeviceList

Description	Scan the list of sensor devices already connected with PC
Function	StatusTypeDef scanDeviceList(DeviceInfo_t** devList, int* deviceNumber)
Parameters	DevList: Return device list deviceNumber: Return deviceNumber
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

Example:

```
DeviceHandle_t handle;
DeviceInfo_t* devList = NULL;
int deviceNumber = 0;
scanDeviceList(&devList, &deviceNumber);
if (deviceNumber == 0)
{
    printf("No device found\r\n");
    return;
}
```

3.2 Open the designated sensor device

openDevice

Description	Open the designated sensor device
Function	StatusTypeDef openDevice(DeviceInfo_t* handle, DeviceHandle_t*devicehandle);
Parameters	handle: User-designated sensor device

	devicehandle: Return the handle of the device
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

Example:

```
//Open the first device in the device list
StatusTypeDef state = openDevice(&devList[0], &handle);
if (state != Status_Succeed)
{
   //Failed to open the device. Print the error code
    printf("openDevice1 failed Error code: %d",state );
    return;
}
```

3.3 Turn off the device

closeDevice

Description	Turn off the designated sensor and release all resources
Function	void closeDevice(DeviceHandle_t handle);
Parameters	handle: Handle of the user-designated sensor device
Return	None
Note	

3.4 Get the device status

isOpen

Description	Get the status of the designated sensor device
Function	bool isOpen(DeviceHandle_t handle);
Parameters	handle: Handle of the user-designated sensor device
Return	true: the device has been opened false: the device has not been opened
Note	

3.5 Get the error description

getErrorText

Description	Get the description of the most recent error
Function	const char* getErrorText(DeviceHandle_t handle);
Parameters	handle: User-designated sensor device
Return	Success error description
Note	After the API call fails, the user can use this interface to get the description of the failure

Example:

```
if (Status_Succeed != zero(handle, 0))
{
  //Print the error description
   printf(" Error: %s",getErrorText(handle));
    return;
}
```

3.6 Getthe SDK version number

getSDKVersion

Description	Get the SDK version number
Function	const char* getSDKVersion(void);
Parameters	None
Return	Return SDK version number
Note	

3.7 Get the controller version number

getControlerVersion

Description	Get the controller version number
Function	const char* getControlerVersion(DeviceHandle_t handle)
Parameters	handle: User-designated sensor device
Return	Return controller version number
Note	

3.8 Start the sensor to continuously output data

continueCaptureStart

Description	Start the sensor to continuously output data
Function	StatusTypeDef continueCaptureStart(DeviceHandle_t handle)
Parameters	handle: User-designated sensor device
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

Example:

```
//Register event callback function
registerEventCallback(eventCallback, NULL);
if (Status_Succeed != continueCaptureStart(handle1))
{
    //Print the error description
    printf(" Error: %s",getErrorText(handle));
    return;
}
```

3.9 Stop the sensor to continuously output data

continueCaptureStop

Description	Stop the sensor to continuously output data
Function	StatusTypeDef continueCaptureStop(DeviceHandle_t handle)
Parameters	handle: User-designated sensor device
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.10 Register event callback function

registerEventCallback

Description	Register event callback function
Function	void registerEventCallback(

	UserEventCallbackHandle eventHandle, void*userPara)
Parameters	eventHandle: Event callback function UserPara: User data
Return	None
Note	

Example:

```
DeviceHandle_t g_handle;
DeviceInfo_t*  g_devList = NULL;
float g_measureData = 0;
float g_saturation = 0;
float g_channel = 0;
void eventCallback(DeviceHandle_t handle, EventCallbackArgs_t arg,
void*userPara)
{
    SC_ResultDataTypeDef_t* res;
    if (handle == g_handle) //Judge whether or not the device is a user-designated device
    {
        //Data is received. It is not advisable to conduct complicated operations here, what
should be done is to just copy the data away
        if (arg.eventType == EventType_DataRecv)
        {


          // Data type
          if (arg.rid == RID_RESULT)
          {
            //arg.dataLen: data of how many channels res[i]: Data of Channel i. If there
is only one channel, arg.dataLen is 1
              res = (SC_ResultDataTypeDef_t*)arg.data;
              //Read measurement results
              //Read the measurement results of the channel. By default, when the
multi-distance measurement mode is not enabled, only result[0] data is valid
               g_channel = res[0].channelIndex;   //Channel index corresponding to
the result
              g_measureData = res[0].result[0];  //Measurement result
              g_saturation = res[0].saturation;//Saturation
              if (g_measureData == INVALID_VALUE) //No signal and invalid value
output
              {
              }
          }
```

```
            else if (arg.rid == RID_DEVICE_DISCONNECT)   //Device disconnected
            {
                printf("Device disconnected\n");
             }
            else if (arg.rid == RID_API_CALL_EXCEPTION)   //API call exception
            {
                const char* err = (const char*)arg.data;
                printf("API call exception, Description%s\n",err);

            }
        }
    }
}
int main()
{
    //Initialization of device enumeration
    ......

   //Register event callback function
   registerEventCallback(eventCallback, NULL);
   if (Status_Succeed != continueCaptureStart(g_handle))
    {
        //Print the error description
        printf(" Error: %s",getErrorText(g_handle));
        return -1;
    }



   getChar();
   //Turn off the device and release all resources
   closeDevice(g_handle);
    return 0;
}
```

3.11 Unregister event callback function

unregisterEventCallback

Description	Unregister event callback function
Function	void unregisterEventCallback();
Parameters	None
Return	None

Note	

3.12 Single sampling of one frame of data

singleShot

Description	Single sampling of one frame of data
Function	StatusTypeDef singleShot(DeviceHandle_t handle, SC_ResultDataTypeDef_t res[]);
Parameters	handle: User-designated sensor device res: Return the results of all activated channels Avger: Average time
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	This function call is valid only when the sensor is not in the continuous output mode

Example:

```
  SC_ResultDataTypeDef_t res[4];//Used to receive the measurement results of
4 channels. If there is only one channel, the array length can be set to 1

  if(Status_Succeed!=singleShot(handle, res))
  {
       //Print the error description
        printf(" Error: %s",getErrorText(handle));
        return -1;

  }


 //Print the results of Channel 0
  printf("Measurement results:" %f\r\n",res[0].result[0]);
  printf("Signal saturation:" %f\r\n",res[0].saturation);
```

3.13 Perform a zero re-set

zero

Description	Perform a zero re-set
Function	StatusTypeDef zero(DeviceHandle_t handle, int channel)
Parameters	handle: User-designated sensor device channel: User-designated channel

Return	Succeed Return Status_Succeed, Fail Return error description code
Note	When performing the zero re-set, make sure that the sensor is within the range and the signal is good

Example:

```
//Perform a zero re-set for Channel 0
if(Status_Succeed!=zero(handle, 0))
{
     //Print the error description
    printf(" Error: %s",getErrorText(handle));
    return -1;
}
```

3.14 Collect the dark signals of the sensor darkSignal

Description	Collect the dark signals of the sensor
Function	StatusTypeDef darkSignal(DeviceHandle_t handle, int channel, bool presetExpTime)
Parameters	handle: User-designated sensor device channel: User-designated channel. If you enter -1, it corresponds to all channels presetExpTime: true: Dark signals are sampled at all pre-set exposure times, saved as a file and stored in the current directory. The data will be automatically loaded when the sensor is turned on next time false: Dark signals are sampled only at the current exposure time and not saved as a file
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	1、The dark signal of the sensor is generated by the back reflection (noise) of the internal optical surface of the sensor. This signal must be subtracted from the actual measurement. The level of the dark signal depends on the exposure time and the luminance of the light source. 2、Before leaving the factory, the equipment has collected dark signals at each pre-set exposure time and light source luminance and saved them in the internal memory of the controller. These data will be automatically loaded when the software is run. But the user must repeat this operation regularly 3、During sampling of dark signals, the sensor must not be within the measurement range 4、It may take tens of seconds to sample dark signals, because the sensor needs to collect the dark signals at all pre-set exposure times and light source

	luminance 5. For more notes about the collection of dark signals, refer to the document <<How HPS-CF Chromatic Confocal Sensor Gets Dark Signals>>

Example:

```
  int channel;
  channel = 0;     // Sample dark signals for Channel 0
 //channel = -1;  // Sample dark signals for all activated channels
 //Dark signals are sampled only at each current exposure time and saved as
a file
 if(Status_Succeed! =darkSignal(handle, channel, true))
 {
      //Print the error description
     printf(" Error: %s",getErrorText(handle));
     return -1;
 }
```

3.15 Setthe measurement mode to be abs mode

setAbsMode

Description	Set the measurement mode to be abs mode and the output results to be absolute distance
Function	StatusTypeDef setAbsMode(DeviceHandle_t handle, int channel, bool en)
Parameters	handle: User-designated sensor device channel: User-designated channel en: true: enable the abs mode; false: Disable the abs mode
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	The result output by the sensor before the zero command is executed is the absolute distance

Example:

```
 //The measurement mode of Channel 0 is set to the abs mode
 if(Status_Succeed!=setAbsMode(handle, 0, true))
 {
    //Print the error description
     printf(" Error: %s",getErrorText(handle));
     return -1;
 }
```

3.16 Set the exposure time

setExposureTime

Description	Set the exposure time of the sensor
Function	StatusTypeDef setExposureTime(DeviceHandle_t handle, PresetExposureTime_t us)
Parameters	handle: User-designated sensor device us: Exposure time, in us
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

Example:

```
//Set the exposure event to be 400us
if(Status_Succeed!=setExposureTime(handle, ExposureTime_400))
{
    //Print the error description
     printf(" Error: %s",getErrorText(handle));
    return -1;
}
```

3.17 Get the current exposure event

getExposureTime

Description	Get the current exposure event
Function	StatusTypeDef getExposureTime(DeviceHandle_t handle, int* expTime)
Parameters	handle: User-designated sensor device expTime: Return exposure time, in us
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.18 Setthe light source luminance

setLightIntensity

Description	Set the light source luminance
Function	StatusTypeDef setLightIntensity(DeviceHandle_t handle, int channel, uint8_t value

	);
Parameters	handle: User-designated sensor device channel: User-designated channel Value: light source luminance 0~255
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	Valid only when not set in the automatic light adjustment

3.19 Set the automatic light adjustment mode setAutoLightIntensity

Description	Set the automatic light adjustment mode
Function	StatusTypeDef setAutoLightIntensity(DeviceHandle_t handle, int channel, bool en);
Parameters	handle: User-designated sensor device channel: User-designated channel en: true Enable automatic light adjustment, false Disable automatic light adjustment
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.20 Set the negation of the measured value reversePos

Description	Set the negation of the measured value
Function	StatusTypeDef reversePos(DeviceHandle_t handle, int channel, bool reverse)
Parameters	handle: User-designated sensor device channel: User-designated channel reverse: True: the measured value is negated; false: the measured value is not negated
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	By default, the closer to the sensor head it is, the smaller the measured displacement value is

3.21 Set the control mode of frame rate

acquisitionFrameMode

Description	Set the control mode of frame rate
Function	StatusTypeDef acquisitionFrameMode(DeviceHandle_t handle, bool en)
Parameters	handle: User-designated sensor device en: True: enable the control mode of frame rate; false: disable the control mode of frame rate
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.22 Set the frame rate

acquisitionFrameRate

Description	Set the frame rate
Function	StatusTypeDef acquisitionFrameRate(DeviceHandle_t handle, int frameRate)
Parameters	handle: User-designated sensor device frameRate: Designated frame rate
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	This command is valid only when it is in the control mode of the frame rate

3.23 Activate or deactivate the designated channel

Active Channel

Description	Activate or de-activate the designated channel
Function	StatusTypeDef activeChannel(DeviceHandle_t handle, int channel, bool en)
Parameters	handle: User-designated sensor device channel: User-designated channel en: true: Enable; false: disable
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	1、By default, when the device is opened, all detected channels that have been inserted into the optical fiber board will be activated 2、Only channels that have been inserted into the optical fiber board can be activated or de-activated 3、The sensor needs to be re-initialized to activate the inside of the channels. This process takes about a few seconds

3.24 Get currently activated channels

getActiveChannel

Description	Get currently activated channels
Function	StatusTypeDef getActiveChannel(DeviceHandle_t handle, uint8_t channel[])
Parameters	handle: User-designated sensor device channel: The user passes in a 4bytes array. If the channel has been activated, the corresponding member is 1, otherwise it is 0
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.25 Get the activation status of the designated channel

isChannelActive

Description	Get the activation status of the designated channel
Function	StatusTypeDef isChannelActive(DeviceHandle_t handle, int channel, uint8_t*active)
Parameters	handle: User-designated sensor device channel: User-designated channel active: Return the activation status of the channel; 1 Activated; 0 Not activated
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.26 Get the number of currently inserted boards

getExistBoard

Description	Get the number of currently inserted boards
Function	StatusTypeDef getExistBoard(DeviceHandle_t handle, uint8_t channel[])
Parameters	handle: User-designated sensor device channel: The user passes in a 4bytes array. If the channel has been inserted in the board, the corresponding member is 1, otherwise it is 0
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.27 saveUserSettings to Flash

saveUserSetting

Description	Save the user settings into Flash. The settings will be automatically loaded when the device is opened
Function	StatusTypeDef saveUserSetting(DeviceHandle_t handle)
Parameters	handle: User-designated sensor device
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.28 Restore the factory settings

restoreFactorySetting

Description	Get the factory settings
Function	StatusTypeDef restoreFactorySetting(DeviceHandle_t handle)
Parameters	handle: User-designated sensor device
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.29 Set the moving average filtering

setMovingAverageFilter

Description	Set the moving average filtering
Function	StatusTypeDef setMovingAverageFilter(DeviceHandle_t handle, int channle, int depth)
Parameters	handle: User-designated sensor device Channel: Channel depth: Moving average depth
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	The Parameters of this filtering are only valid in the continuous output mode and are not valid for a single output command

Example:

```
//Set the moving average depth of Channel 0 to be 32
if(Status_Succeed!=setMovingAverageFilter(handle,0, 32))
```

```
{
    //Print the error description
     printf(" Error: %s",getErrorText(handle));
    return -1;
}
```

3.30 Set the median filtering

setMedianFilterFilter

Description	Set the median filtering
Function	StatusTypeDef setMedianFilterFilter(DeviceHandle_t handle, int channle, int depth)
Parameters	handle: User-designated sensor device Channel: Channel depth: Set the median filtering depth
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	The Parameters of this filtering are only valid in the continuous output mode and are not valid for a single output command

Example:

```
//Set the median filtering depth of Channel 0 to be 15
if(Status_Succeed!=setMedianFilterFilter(handle,0, 15))
{
    //Print the error description
     printf(" Error: %s",getErrorText(handle));
    return -1;
}
```

3.31 Set the average filtering

setAverageFilter

Description	Set the average filtering
Function	StatusTypeDef setAverageFilter(DeviceHandle_t handle, int avergeNumber)
Parameters	handle: User-designated sensor device avergeNumber: Average time
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	Not used for the moving average filtering and does not affect the frame rate. The average filtering is to take the average of multiple frames of data as one-frame result output, which will reduce the frame rate of output

Example:

```
//Set the average time to be 4
  if(Status_Succeed!=setAverageFilter(handle,4))
  {
      //Print the error description
      printf(" Error: %s",getErrorText(handle));
      return -1;
  }
```

3.32 Set the Kalman filtering

setKalmanFilterPara

Description	Set the Kalman filtering
Function	StatusTypeDef set Kalman Filter Para(DeviceHandle_t handle, int channle, KalmanFilterPara_t para)
Parameters	handle: User-designated sensor device channle: User-designated channel para: Kalman filtering Parameters
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

Example:

```
KalmanFilterPara_t para;
para.kalman_k = 0.1;
para.kalman_threshold = 0.002;
para.num_check = 10;
//Set the Kalman filtering Parameters of Channel 0
if(Status_Succeed!=setKalmanFilterPara(handle,0,para))
{
    //Print the error description
     printf(" Error: %s",getErrorText(handle));
    return -1;
}
```

3.33 Enable the Kalman filtering

enableKalmanFilter

Description	Enable the Kalman filtering
Function	StatusTypeDef enableKalmanFilter(DeviceHandle_t handle, int channel, bool en)
Parameters	handle: User-designated sensor device channle: User-designated channel en:true: enable the Kalman filtering; false: disable the Kalman filtering
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.33 Set the number of abnormal data

setErrorFilterDataCount

Description	Set the number of abnormal data. If the number of continuous abnormal data is greater than this number, the data will be output to the user and if it is less than this number, the last data will be output
Function	StatusTypeDef setErrorFilterDataCount(DeviceHandle_t handle, int channel, int number)
Parameters	handle: User-designated sensor device channle: User-designated channel Number: Number of abnormal data
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.34 Enable abnormal data filtering

enableErrorDataFilter

Description	Enable abnormal data filtering
Function	StatusTypeDef enableErrorDataFilter(DeviceHandle_t handle, int channel, bool en)
Parameters	handle: User-designated sensor device channle: User-designated channel en:true: enable abnormal data filter; false: disable abnormal data filter
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.35 Set the measurement unit

setMearsureUnit

Description	Set the unit of measurement results and the default measurement unit is millimeter
Function	StatusTypeDef setMearsureUnit(DeviceHandle_t handle, int channel, Confocal_MeasuretUnit_t unit)

Parameters	handle: User-designated sensor device
	channle: User-designated channel
	unit: measurement unit
	//measurement unit
	type def enum
	{
	MeasuretUnit_mm = 0, //millimeter
	MeasuretUnit_um, //microsecond
	MeasuretUnit_inch, //inch
	}Confocal_MeasuretUnit_t;

Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.36 Get the measurement range of the sensor head

getMeasureRange

Description	Get the measurement range of the sensor head
Function	StatusTypeDef getMeasureRange(DeviceHandle_t handle, int channel, double*mini_mm, double*max_mm)
Parameters	handle: User-designated sensor device channle: User-designated channel mini_mm: Return the near-end position of the range, in mm max_mm: Return the far-end position of the range, in mm
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.37 Set the measurement mode of the designated channel

setChannelMeasureMode

Description	Get the measurement range of the sensor head

Function	StatusTypeDef setChannelMeasureMode(DeviceHandle_t handle, int channel, Confocal_MeasureMode_t measureMode)

Parameters	handle: User-designated sensor device
	channle: User-designated channel
	Measure Mode: designate the measurement mode
	//Measurement mode
	type def enum
	{
	Measure Mode_Distance = 0, //Distance mode
	Measure Mode_Thickness, //Thickness mode
	}Confocal_Measure Mode_t;

Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.38 Get the signal saturation of the designated channel getSaturation

Description	Get the measurement range of the sensor head
Function	StatusTypeDef getSaturation(DeviceHandle_t handle, int channel, float*saturation)
Parameters	handle: User-designated sensor device channle: User-designated channel saturation: Refer to return the signal saturation of the channel
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.39 Get the range of analog voltage output getAnalogVoltageMaxRange

Description	Get the measurement range of the sensor head
Function	StatusTypeDef getAnalogVoltageMaxRange(DeviceHandle_t handle, double *mini_vol, double *max_vol)
Parameters	handle: User-designated sensor device channle: User-designated channel mini_vol: Return the minimum value of voltage

	max_vol: Return the maximum value of voltage
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.40 Set the mapping relationship between analog voltage and distance

setAnalogVoltageMapping

Description	Get the measurement range of the sensor head
Function	StatusTypeDef setAnalogVoltageMapping(DeviceHandle_t handle, int channel, double mini_vol, double max_vol, double value_mini_mm, double value_max_mm)
Parameters	handle: User-designated sensor device channle: User-designated channel mini_vol: Set the minimum value of the output voltage, in V max_vol: Set the maximum value of the output voltage, in V value_mini_mm: Set the minimum value of the measured displacement, in mm value_max_mm: Set the maximum value of the measured displacement, in mm
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	The value_mini_mm and value_max_mm need to be set according to the user's actual measurement scenario and the measurement range of the sensor

Example:

```
double mini_vol = -10;   //Set the minimum value of the output voltage to be -10V
double max_vol = 10;    //Set the maximum value of the output voltage to be 10V
double mini_dis = -1.5;  //Set the minimum displacement value measured to be -1.5mm
double max_dis = 1.5;   //Set the maximum displacement value measured to be 1.5mm

//Set the voltage-distance mapping relationship of Channel 0
  if(Status_Succeed!=set Analog Voltage Mapping(
     handle,0,mini_vol,max_vol,mini_dis,max_dis)
```

```
)
{
    //Print the error description
    printf(" Error: %s",getErrorText(handle));
    return -1;
}
//Enable the voltage output of Channel 0
if(Status_Succeed!=setAnalogVoltageOutput(handle,0,true))
{
    //Print the error description
    printf(" Error: %s",getErrorText(handle));
    return -1;
}
```

3.41 Enable the analog voltage output

setAnalogVoltageOutput

Description	Enable the analog voltage output
Function	StatusTypeDef setAnalogVoltageOutput(DeviceHandle_t handle, int channel, bool en)
Parameters	handle: User-designated sensor device channle: User-designated channel en: True: enable the analog voltage output; false: disable the analog voltage output
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.42 Set the multi-distance measurement mode

setMultiDistanceMode

Description	Set the multi-distance measurement mode. By default, if it is not enabled, only the distance corresponding to the spot with the highest intensity is calculated
Function	StatusTypeDef setMultiDistanceMode(DeviceHandle_t handle, int channel, bool en)
Parameters	handle: User-designated sensor device channle: User-designated channel

	en: True: enable the multi-distance measurement mode; false: disable the multi-distance measurement mode
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.43 Set the main spot index

setMainSignalIndex

Description	Set the main spot index
Function	StatusTypeDef setMainSignalIndex(DeviceHandle_t handle,int channel, int index)
Parameters	handle: User-designated sensor device channle: User-designated channel index: Spot index
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	This command is valid only when the multi-distance measurement mode is enabled. Used to select a spot as analog output, IO alarm output and filtering data.

3.44 Binding of IO port output and particular events

bindAlarmIOPort

Description	Binding of the IO port and the particular events of the designated channel
Function	StatusTypeDef bindAlarmIOPort(DeviceHandle_t handle, int channel, AlarmType_t type, int alarmPort, IoPortState_t portState)
Parameters	handle: User-designated sensor device channle: User-designated channel type: Alarm type Alarm Port: IO port selection and 0~3 correspond to ERROR1~ERROR4 portState: The on/off state of the IO port when the alarm is triggered

	//Alarm type
	type def enum
	{
	AlarmType_None, //No alarm
	AlarmType_UpperLimit, //Alarm for the upper limit
	AlarmType_LowerLimit, /Alarm for the lower limit
	AlarmTyp_DeviceDisconnect, //Alarm for device disconnection
	AlarmType_SignalWeak, //Alarm for weak signal
	AlarmType_SignalSaturated, //Alarm for signal saturation
	AlarmType_TempError, //Alarm for abnormal temperature
	AlarmType_FanError, //Alarm for fan not rotating
	}AlarmType_t;
	
	//On/off status of false alarm output
	type def enum
	{
	PortState_Off = 0, //IO disconnected
	PortState_On = 1 //IO closed
	}IoPortState_t;

Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

Example:

```
//Set the upper limit of tolerance of Channel 0 to be 0.7mm
setUpperLimit(handler, 0, 0, 0.7);
/*
  Channel: Channel 0
  Alarm type: Alarm for the upper limit of tolerance
  Alarm port: ERROR1
  State of the alarm triggering port: Port closed
*/
if (Status_Succeed != bindAlarmIOPort(
     handler, 0, AlarmType_UpperLimit, 0, PortState_On)
  )
{
    //Print the error description
   printf(" Error: %s",getErrorText(handle));
   return -1;
```

```
}
```

3.45 Unbind the designated alarm IO port

unbindAlarmIOPort

Description	Unbind the designated alarm IO port
Function	StatusTypeDef unbindAlarmIOPort(DeviceHandle_t handle, int alarmPort)
Parameters	handle: User-designated sensor device channle: User-designated channel Alarm Port: The user-designated IO ports 0~3 correspond to ERROR1~ERROR4
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.46 Set the upper limit of tolerance

setUpperLimit

Description	Set the upper limit of tolerance
Function	StatusTypeDef setUpperLimit(DeviceHandle_t handle, int channel, int progSegIndex, double mm)
Parameters	handle: User-designated sensor device channle: User-designated channel progSegIndex: For program segments designated by the user, each channel supports a maximum of 8 program segments mm: Upper limit of tolerance, in mm
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.47 Set the lower limit of tolerance

setLowerLimit

Description	Set the upper limit of tolerance
Function	StatusTypeDef set Lower Limit(DeviceHandle_t handle,

	int channel, int progSegIndex, double mm)
Parameters	handle: User-designated sensor device channle: User-designated channel progSegIndex: For program segments designated by the user, each channel supports a maximum of 8 program segments mm: Lower limit of tolerance, in mm
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.48 Set the program segment for IO alarms for the upper and lower limit of tolerance

setProgramSegmentIndex

Description	Set the upper limit of tolerance
Function	StatusTypeDef setProgramSegmentIndex(DeviceHandle_t handle, int channel, int index)
Parameters	handle: User-designated sensor device channle: User-designated channel index: Designate the currently activated program segments
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.49 Set the spot index of the layers during thickness measurement

setThicknessLayer

Description	Designate the spot index of the layers during thickness measurement. When the thickness of multiple layers are measured, the user can use this command to designate the measurement of the thickness of a certain layer or multiple layers
Function	StatusTypeDef setThicknessLayer(DeviceHandle_t handle, int channel, uint8_t index1, uint8_t index2)

Parameters	
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	By default, the spot index for calculating thickness during thickness measurement is 0, 1, corresponding to the first and second spots. The user can use this command to add the spot index corresponding to the number of thickness layers he wants to measure.

Example:

```
//Set the measurement mode of Channel 0 to the thickness measurement mode
if(Status_Succeed!=setChannelMeasureMode(handle,0,MeasureMode_Thickness
))
{
    //Print the error description
    printf(" Error: %s",getErrorText(handle));
    return -1;
}
//Set the spot index of the first-layer thickness to be 0 and 1
if(Status_Succeed!=setThicknessLayer(handle,0,0,1))
{
    //Print the error description
    printf(" Error: %s",getErrorText(handle));
    return -1;
}
//Set the spot index of the second-layer thickness to be 2 and 3
if(Status_Succeed!=setThicknessLayer(handle,0,2,3))
{
    //Print the error description
    printf(" Error: %s",getErrorText(handle));
    return -1;
}

//Register event callback function
registerEventCallback(eventCallback, NULL);
//Start the continuous measurement of the sensor
if (Status_Succeed != continueCaptureStart(handle1))
{
    //Print the error description
     printf(" Error: %s",getErrorText(handle));
    return;
}
```

3.50 Delete the spot index for thickness measurement

removeThicknessLayer

Description	Delete the spot index for thickness measurement
Function	StatusTypeDef removeThicknessLayer(DeviceHandle_t handle, int channel, int index1, int index2)
Parameters	handle: User-designated sensor device channle: User-designated channel index1: The first spot index. If it is less than 0, clear all settings Index2: The second spot index. If it is less than 0, clear all settings
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.51 Get the number of spot index groups under thickness measurement

getThicknessLayerNumber

Description	Get the number of spot index groups under thickness measurement
Function	StatusTypeDef getThicknessLayerNumber(DeviceHandle_t handle, int channel, int*layerNumber)
Parameters	handle: User-designated sensor device channle: User-designated channel Layer Number: Return the number of spot index groups
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.52 Perform the thickness measurement calibration thicknessModeDoCalibration

Description	Perform the thickness measurement calibration
Function	StatusTypeDef thicknessModeDoCalibration(DeviceHandle_t handle, int channel, float std Thickness)
Parameters	handle: User-designated sensor device channle: User-designated channel Std Thickness: Glass of standard thickness of the same material as the object to be measured
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	1、Due to the internal refraction of the glass in measuring glass thickness, there will be a difference between the measured thickness and the actual thickness, so the user needs to use a piece of glass of standard thickness for calibration 2、In calibration, the user needs to place a piece of glass of standard thickness within the measurement range of the sensor, and ensures that the sensor is set with reasonable Parameters, so that signals are reflected from each surface of the glass back to the sensor

3.53 Set the coefficient of thickness measurement thicknessModeSetRatio

Description	Set the coefficient of thickness measurement
Function	StatusTypeDef thicknessModeSetRatio(DeviceHandle_t handle, int channel, float ratio)
Parameters	handle: User-designated sensor device channle: User-designated channel ratio: After a coefficient is set, the measurement results will be multiplied by the coefficient
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.54 Set the trigger mode of the sensor

setTriggerMode

Description	Set the trigger mode of the sensor
Function	StatusTypeDef setTriggerMode(devicehandle_thandle, Confocal_Trigger Mode_t mode)

Parameters	handle: User-designated sensor device
	channle: User-designated channel
	mode: User-designated trigger mode
	//Selection of the trigger mode
	type def enum
	{
	Trigger_Internal = 0, //Internal Trigger
	Trigger_Extern, //External Trigger
	Trigger_Encoder, //Encoder Trigger
	Trigger_Timing, //Internal Timing Trigger
	}Confocal_TriggerMode_t;

Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

Example:

```
    //Set the trigger mode to be external trigger, which correspondly triggers the IO port SYNC-IN
    if (Status_Succeed != setTriggerMode(handler, Trigger_Extern))
    {
        //Print the error description
       printf(" Error: %s",getErrorText(handle));
       return -1;
    }
    //Register event callback function
    registerEventCallback(eventCallback, NULL);
    //Start collecting data
    if (Status_Succeed != continueCaptureStart(handle1))
    {
        //Print the error description
        printf(" Error: %s",getErrorText(handle));
        Return -1;
    }
```

3.55 Set the encoder channel

setEncoderChannel

Description	Set the encoder channel
Function	StatusTypeDef setEncoderChannel(DeviceHandle_t handle, int encoder Channel)
Parameters	handle: User-designated sensor device Encoder Channel: Designated encoder channel
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.56 Set the division factor for encoder trigger

setEncoderDivision

Description	Set the division factor for encoder trigger
Function	StatusTypeDef setEncoderDivision(DeviceHandle_t handle, uint16_t division)
Parameters	handle: User-designated sensor device division: Division factor of encoder, 64~65535
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.57 Set the encoder counter asynchronous notice

enableEncoderCounterNotice

Description	Set the division factor for encoder trigger
Function	StatusTypeDef enableEncoderCounterNotice(DeviceHandle_t handle, bool en)
Parameters	handle: User-designated sensor device en: True: enable the asynchronous notice; false: disable the asynchronous notice
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

Example:

```
//Event callback function
void eventCallback(DeviceHandle_t handle, EventCallbackArgs_t arg,
void*userPara)
{
    SC_ResultDataTypeDef_t* res;
    if (handle == g_handle) //Judge whether or not the device is a user-designated device
    {
        //Data is received. It is not advisable to conduct complicated operations here, what
should be done is to just copy the data away
        if (arg.eventType == EventType_DataRecv)
        {
           // Data type
            if (arg.rid == RID_RESULT)
            {

            }
             else if (arg.rid == RID_ENCODER_COUNT)  //encoder count value
            {
                int32_t counter = arg.data[0];
            }
            else if (arg.rid == RID_DEVICE_DISCONNECT)   //Device disconnected
            {

             }
            else if (arg.rid == RID_API_CALL_EXCEPTION)   //API call exception
            {

            }
        }
    }
 }
```

3.58 Enable the external synchronous output setTriggerSyncOut

Description	Enable the external synchronous output, which corresponds to IO port SYNC-OUT
Function	StatusTypeDef setTriggerSyncOut(DeviceHandle_t handle, bool en)
Parameters	handle: User-designated sensor device en: True: enable; false: disable
Return	Succeed Return Status_Succeed, Fail Return error description code

Note	When synchronous output is enabled, SYNC-OUT will output pulses with the same frequency as the current measurement frequency of the sensor

3.59 Set the threshold coefficient for signal detection

setSignalDetectThresholdCoef

Description	Set the threshold coefficient for signal detection
Function	StatusTypeDef setSignalDetectThresholdCoef(DeviceHandle_t handle, int channel, double threshold Coef)
Parameters	handle: User-designated sensor device channel: User-designated channel Threshold Coef: coefficient, 3.0 by default
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	In the multi-distance measurement mode or the thickness measurement mode, sometimes the number of measured results is incorrect due to the mismatch between the threshold coefficient for signal detection and the current measurement scenario. The user can set the coefficient manually or use the automatic detection command.

3.60 Automatically set the signal detection threshold

doSignalDetectThresholdCoefAutoFound

Description	Automatically set the signal detection threshold
Function	StatusTypeDef doSignalDetectThresholdCoefAutoFound(DeviceHandle_t handle, int channel, int signal Number, double*threshold)
Parameters	handle: User-designated sensor device channel: User-designated channel Signal Number: Number of spots to be detected threshold: Return the threshold coefficient for signal detection
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	1、Put the object to be detected into the range of the sensor, set the number of signals that the user needs to detect, and perform automatic detection 2、After successful detection, the coefficient will be automatically set inside the sensor and returned to the user

3.61 Set the RSxxx communication protocol

setRSxxxProtocol

Description	Set the RS485/RS422/RS232 communication protocol
Function	setRSxxxProtocol(DeviceHandle_t handle, COMM_Protocol_Control_Enum_t comm_control, COMM_Protocol_Enum_t comm_protocol);

Parameters	handle: Handle of the user-designated sensor device
	comm_control: Select hardware control or software control
	Comm_protocol: select the RSXXX protocol
	//Right of control of the communication protocol
	type def enum
	{
	Hardware = 0,//Default mode
	Software = 1
	}COMM_Protocol_Control_Enum_t;
	
	//Select the communication protocol
	type def enum
	{
	RS422_COMM = 0, //Default mode
	RS485_COMM = 1,
	RS232_COMM = 3 //Both 2 and 3 are RS232 communication protocols
	}COMM_Protocol_Enum_t;

Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.62 Set the RS xxx communication Parameters

setRSxxxCommunicationPara

Description	Set the RS xxx communication Parameters
Function	StatusType Def set RS xxx Communication Para(DeviceHandle_t handle, COMM_BaudRate_Enum_t baudrate, COMM_Parity_Enum_t parity);
Parameters	handle: Handle of the user-designated sensor device comm_control: Select hardware control or software control Comm_protocol: select the RSXXX protocol

	//Right of control of the communication protocol
	type def enum
	{
	Hardware = 0,//Default mode
	Software = 1
	}COMM_Protocol_Control_Enum_t;
	
	//Select the communication protocol
	type def enum
	{
	RS422_COMM = 0, //Default mode
	RS485_COMM = 1,
	RS232_COMM = 3 //Both 2 and 3 are RS232 communication protocols
	}COMM_Protocol_Enum_t;

Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.63 Set the parsing format of RS xxx commands

setRSxxxDataFormat

Description	Set the parsing format of RS xxx commands
Function	StatusTypeDef setRSxxxDataFormat(DeviceHandle_t handle, COMM_Data_Format_Enum_t format);

Parameters	handle: Handle of the user-designated sensor device
	format: Command parsing format
	//Data format of communication
	type def enum
	{
	ASCII = 0,
	Hexadecimal = 1 //Default data format
	
	}COMM_Data_Format_Enum_t;

Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.64 Set or disable the independent mode

setChannelIndependentMode

Description	Set or disable the independent mode; each channel sensor head operates independently in the independent mode and the independent mode needs to be disabled in the double-head thickness measurement model
Function	StatusTypeDef setChannelIndependentMode(DeviceHandle_t handle, bool en);
Parameters	handle: Handle of the user-designated sensor device en: true: enable the independent mode; false: disable the independent mode
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.65 Set the measurement mode under multi-channel cooperation

setMultiChannelMeasureMode

Description	Set the measurement mode under multi-channel cooperation
Function	StatusTypeDef setMultiChannelMeasureMode(DeviceHandle_t handle,

	Confocal_Cooperation Measure Mode_t mode);

Parameters	handle: Handle of the user-designated sensor device
	en: true: enable the independent mode; false: disable the independent mode
	type def enum
	{
	CM_Thickness = 0, //Double-headed thickness measurement
	}Confocal_CooperationMeasureMode_t;

Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.66 Set the offset under multi-channel cooperation setMultiChannelMeasureOffset

Description	Set the offset under multi-channel cooperation
Function	StatusTypeDef setMultiChannelMeasureOffset(DeviceHandle_t handle, int groupIndex, double offset);
Parameters	handle: Handle of the user-designated sensor device Group Index: Group selection offset: Offset
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.67 Get the offset under multi-channel cooperation getMultiChannelMeasureOffset

Description	Get the offset under multi-channel cooperation
Function	StatusTypeDef getMultiChannelMeasureOffset(DeviceHandle_t handle, int groupIndex, double* offset);
Parameters	handle: Handle of the user-designated sensor device Group Index: Group selection offset: Return offset
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.68 Set the group index in the double-headed thickness measurement mode

setDoubleChannelThicknessGroupIndex

Description	Set the group index in the double-head thickness measurement mode. 0 group corresponds to Channels 0, 1; 1 group corresponds to Channels 2, 3
Function	StatusTypeDef setDoubleChannelThicknessGroupIndex(DeviceHandle_t handle, int groupIndex, bool en);
Parameters	handle: Handle of the user-designated sensor device Group Index: Group selection en: True: enable; false: disable
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.69 Transparent or translucent mode of double-headed measurement

doubleChannelALMode

Description	Transparent or translucent mode of double-head measurement
Function	StatusTypeDef doubleChannelALMode(DeviceHandle_t handle, bool en);
Parameters	handle: Handle of the user-designated sensor device Group Index: Group selection
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.70 Enabled ABS mode of double-headed measurement

doubleChannelABSMode

Description	Transparent or translucent mode of double-head measurement
Function	StatusTypeDef doubleChannelABSMode(DeviceHandle_t handle, int groupIndex, bool en)

Parameters	handle: Handle of the user-designated sensor device Group Index: Group selection en: True: enable; false: disable
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.71 Zeroing of the two channels of double-headed measurement

doubleChannelThicknessZero

Description	Transparent or translucent mode of double-head measurement
Function	StatusTypeDef doubleChannelThicknessZero(DeviceHandle_t handle, int group Index)
Parameters	handle: Handle of the user-designated sensor device Group Index: Group selection
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.72 Enable output signal data

setSignalDataOutput

Description	Enable output signal data, which is only used for debugging
Function	StatusTypeDef setSignalDataOutput(DeviceHandle_t handle, bool en)
Parameters	handle: Handle of the user-designated sensor device en: True: enable; false: disable
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.73 Enable automatic detection of multiple signals

setAutoSignalDetect

Description	Enable automatic detection of multiple signals
Function	StatusTypeDef setAutoSignalDetect(DeviceHandle_t handle, int channel, bool en)
Parameters	handle: Handle of the user-designated sensor device channel: Designated channel en: True: enable; false: disable
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.74 Set the number of detected signals

setDetectSignalNumber

Description	Set the number of detected signals. If the number of signals detected is not equal to this number, an invalid value will be output
Function	StatusTypeDef setDetectSignalNumber(DeviceHandle_t handle, int channel, int signal Number)
Parameters	handle: Handle of the user-designated sensor device channel: Designated channel Signal Number: Number of signals
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.75 Set the step for accurately searching signals

setSignalSearchAccurateStep

Description	Set the step for accurately searching signals
Function	StatusTypeDef setSignalSearchAccurateStep(DeviceHandle_t handle, int channel, uint8_t step)
Parameters	handle: Handle of the user-designated sensor device channel: Designated channel step: Number of steps

Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.76 Set the number of steps for the rising edge of the signal

setSignalRisingStepCount

Description	Set the number of steps for the rising edge of the signal
Function	StatusTypeDef setSigna RisingStepCount(DeviceHandle_t handle, int channel, uint8_t count)
Parameters	handle: Handle of the user-designated sensor device channel: Designated channel count: Count
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.77 Set the rising-falling edge ratio of signal detection

setSignalRisingFallRatio

Description	Set the rising-falling edge ratio of signal detection
Function	StatusTypeDef setSignalRisingFallRatio(DeviceHandle_t handle, int channel, double ratio)
Parameters	handle: Handle of the user-designated sensor device channel: Designated channel ratio: Ratio
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.78 Set the data cache size

setDataCacheSize

Description	Set the data cache size. By default, each channel can cache 1 million data; be used to filter out the maximum, minimum, average and standard deviation from these data
Function	StatusTypeDef setDataCacheSize(DeviceHandle_t handle, int cache Index, int dataCount)
Parameters	handle: Handle of the user-designated sensor device Cache Index: The cache index (0~3) corresponds to 4 channels Data Count: Number of cached data
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.79 Get the number of data in the current data cache

getCurDataCacheCount

Description	Get the number of data in the current data cache
Function	StatusTypeDef getCurDataCacheCount(DeviceHandle_t handle, int cache Index, int* data Count)
Parameters	handle: Handle of the user-designated sensor device Cache Index: The cache index (0~3) corresponds to 4 channels Data Count:Return the number of data
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.80 Clear the data cache

clearDataCache

Description	Get the number of data in the current data cache
Function	StatusTypeDef clearDataCache(DeviceHandle_t handle, int cache Index)
Parameters	handle: Handle of the user-designated sensor device Cache Index: The cache index (0~3) corresponds to 4 channels

Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.81 Filter data from the buffer

selectCacheData

Description	Use the data cached to calculate the result according to the data attributes designated by the user
Function	StatusTypeDef selectCacheData(DeviceHandle_t handle, int cache Index, Data Set Attribute_t attribute, double ret[])

Parameters	handle: Handle of the user-designated sensor device
	Cache Index: The cache index (0~3) corresponds to 4 channels
	attribute: Designate the attributes
	ret : Return the calculation result
	//Attributes of data set
	type def enum
	{
	Attribute_MinMax, //Maximum and minimum
	Attribute_Avg, //Average
	Attribute_PtP, //Peak-peak value
	Attribute_STD //Standard deviation
	
	}Data Set Attribute_t

Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.82 Clear the pulse count in the encoder

clearEncoderCount

Description	Clear the pulse count in the encoder
Function	StatusTypeDef clearEncoderCount(DeviceHandle_t handle, int encoder Channel)

Parameters	handle: Handle of the user-designated sensor device Encoder Channel: Designated encoder channel
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.83 selectSignals to be calculated

selectSignal

Description	In the single distance mode, which signal is selected for calculation
Function	StatusTypeDef selectSignal(DeviceHandle_t handle, int channel, Confocal_Signal Select_t signal)

Parameters	handle: Handle of the user-designated sensor device
	channel: Designated channel
	signal: Signal selection
	//In the single distance mode, which signal is selected for calculation
	type def enum
	{
	Signal_MaxIntensity, //Signal with the maximum light intensity
	Signal_NearEnd, //Near-end signal
	Signal_FarEnd //Far-end signal
	}Confocal_SignalSelect_t;

Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.84 Clear moving average data

clearMovingAverageFilter

Description	Clear the data in the moving average filtering
Function	StatusTypeDef clearMovingAverageFilter(DeviceHandle_t handle, int channel)
Parameters	handle: Handle of the user-designated sensor device channel: Designated channel

Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.85 Clear sliding median data

clearMedianFilter

Description	Clear the data in the moving median filtering
Function	StatusTypeDef clearMedianFilter(DeviceHandle_t handle, int channel)
Parameters	handle: Handle of the user-designated sensor device channel: Designated channel
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.86 Get the serial number of the sensor head

getSensorHeadSN

Description	Get the serial number of the sensor head
Function	const char* getSensorHeadSN(DeviceHandle_t handle, int channel)
Parameters	handle: Handle of the user-designated sensor device channel: Designated channel
Return	Return the serial number of the sensor head
Note	

3.87 Get the serial number of the controller

getControlerSN

Description	Get the serial number of the sensor head
Function	const char* getControlerSN(DeviceHandle_t handle)

Parameters	handle: Handle of the user-designated sensor device channel: Designated channel
Return	Return the serial number of the controller
Note	

3.88 Open the specified sensor device(CF2000)

GE_openDevice

Description	Open the specified sensor device
Function	StatusTypeDef GE_openDevice (ControllerGEPara_t* controllerPara, char* localIP, DeviceHandle_t* handle);
Parameters	controllerPara:If NULL is used, the fixed parameters of the controller are used for connection localIP:Local IP. If NULL is used, INADDR_ANY is used to bind all local IP addresses handle:Handle of the user-designated sensor device
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

Example:

```
//Open device
StatusTypeDef state = GE_openDevice(NULL,NULL &handle);
if (state != Status_Succeed)
{
    //failed
    printf("Open devicel failed Error code: %d",state );
    return;
}
```

3.89 Set threshold of the signal calculation

setCalculationThreshold

Description	Set threshold of the signal calculation
Function	StatusTypeDef setCalculationThreshold(DeviceHandle_t handle, int channel, double ratio);
Parameters	handle:Handle of the user-designated sensor device channel:Selected channel ratio:range 0~1,If it is set to 1, all points from the maximum value to the minimum value of the signal will be calculated;,If it is set to 0.5, the points from the Half of the maximum to the minimum value of the signal will be calculated
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.90 Pivot the data of double channel

doubleChannelReversePos

Description	Pivot the data of double channel
Function	StatusTypeDef doubleChannelReversePos(DeviceHandle_t handle, int groupIndex, bool en);
Parameters	handle:Handle of the user-designated sensor device groupIndex:Selected group en: true:enable false:disable
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.91 Export the data from the cache

dumpCacheData

Description	The user should call getCurDataCacheCoun to find out how much data is currently in the Cache, and then make space to fetch the data out
Function	StatusTypeDef dumpCacheData(DeviceHandle_t handle, int cacheIndex, double retData[], int maxCount, int32_t* dataCount);
Parameters	handle:Handle of the user-designated sensor device CacheIndex:The indexes of the cache(0~3) correspond to 4 channels data:the returned data maxDataCount:The maximum length dataCount:Return the actual data
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.92 Export the data from the cache(Single Channel)

dumpCacheData_SC

Description	The user should call getCurDataCacheCoun to find out how much data is currently in the Cache, and then make space to fetch the data out
Function	StatusTypeDef dumpCacheData_SC(DeviceHandle_t handle, int cacheIndex, SC_ResultDataTypeDef_t retData[], int maxCount, int32_t* dataCount);
Parameters	handle:Handle of the user-designated sensor device CacheIndex:The indexes of the cache(0~3) correspond to 4 channels data:the returned data maxDataCount:The maximum length dataCount:Return the actual data
Return	Succeed Return Status_Succeed, Fail Return error description code

Note	

3.93 Export the data from the cache(Multiple Channel)

dumpCacheData_MC

Description	The user should call getCurDataCacheCoun to find out how much data is currently in the Cache, and then make space to fetch the data out
Function	StatusTypeDef dumpCacheData_MC(DeviceHandle_t handle, int cacheIndex, MC_ResultDataTypeDef_t retData[], int maxCount, int32_t* dataCount);
Parameters	handle:Handle of the user-designated sensor device CacheIndex:The indexes of the cache(0~3) correspond to 4 channels data:the returned data maxDataCount:The maximum length dataCount:Return the actual data
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.94 Set the offset of channel

setChxOffset

Description	Set the offset of channel
Function	StatusTypeDef setChxOffset(DeviceHandle_t handle, int channel, float offset);
Parameters	handle:Handle of the user-designated sensor device channel:The selected channel offset:
Return	Succeed Return Status_Succeed, Fail Return error description code

Note	

3.95 Get the offset of channel

GetChxOffset

Description	Get the offset of channel
Function	StatusTypeDef getChxOffset(DeviceHandle_t handle, int channel, float* offset);
Parameters	handle:Handle of the user-designated sensor device channel:The selected channel offset:Return offset
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.96 Set the IP address of controller

setControllerIPAddr

Description	Set the IP address of controller
Function	StatusTypeDef setControllerIPAddr(DeviceHandle_t handle, char* ip_addr);
Parameters	handle:Handle of the user-designated sensor device ip_addr:IP address
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.97 Set the port of controller

setControllerPort

Description	Set the port of controller
Function	StatusTypeDef setControllerPort(DeviceHandle_t handle, uint16_t port);
Parameters	handle:Handle of the user-designated sensor device port: port number
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.98 Bind input ports

bindInputPort

Description	Bind input ports

Function	StatusTypeDef bindInputPort(
	DeviceHandle_t handle,
	int Channel,
	Confocal_InputPortFunc_t func,
	int inputPort
	);
	
	enum Confocal_InputPortFunc_t
	{
	InputPort_None=0,
	InputPort_ExtTrigger,
	InputPort_ExtTriggerCache,
	InputPort_Zero,
	InputPort_StartSample,
	InputPort_StopSample,
	InputPort_SampleToggle,
	InputPort_ClearCache,
	InputPort_EnableCache,
	InputPort_DisableCache,
	};

Parameters	handle:Handle of the user-designated sensor device Confocal_InputPortFunc_t inputPort
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.99 Unbind Input Ports

unbindInputPort

Description	Unbind Input Ports
Function	StatusTypeDef unbindInputPort(DeviceHandle_t handle, int inputPort);
Parameters	handle:Handle of the user-designated sensor device inputPort:
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.100 Get the current frame rate

getCurFrameRate

Description	Get the current frame rate
Function	uint32_t getCurFrameRate(DeviceHandle_t handle);
Parameters	handle:Handle of the user-designated sensor device
Return	Returns the current internal frame rate
Note	

3.101 Enable the data cache

enableDataCache

Description	Enable the data cache (enable by default)
Function	StatusTypeDef enableDataCache(DeviceHandle_t handle, int cacheIndex, bool en);
Parameters	handle:Handle of the user-designated sensor device en: true:enable false:disable
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.102 Set the encoder input mode

setEncoderInputMode

Description	Set the encoder input mode
Function	StatusTypeDef setEncoderInputMode(DeviceHandle_t handle, Confocal_EncoderInputMode_t mode);
Parameters	handle:Handle of the user-designated sensor device Confocal_EncoderInputMode_t: typedefenum { Mode_1_INC_1=0,//One phase one increasing Mode_2_INC_1=1,//Two phase one increasing Mode_2_INC_2=2,//Two phase two increasing Mode_2_INC_4=3,//Two phase four increasing }Confocal_EncoderInputMode_t;
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.103 Set the encoder working mode

setEncoderWorkingMode

Description	Set the encoder working mode
Function	StatusTypeDef setEncoderWorkingMode(DeviceHandle_t handle, Confocal_EncoderWorkingMode_t mode);
Parameters	handle:Handle of the user-designated sensor device Encoder_Working_Mode_Enum_t: typedefenum { Mode_Three_Signal_End=0, Mode_Diff_One_Signal_End=1, }Confocal_EncoderWorkingMode_t;
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.104 clear the counts of encoders

clearEncodersCount

Description	clear the counts of encoders
Function	StatusTypeDef clearEncodersCount(DeviceHandle_t handle);
Parameters	handle:Handle of the user-designated sensor device Confocal_EncoderInputMode_t: the format of mode
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.105 Get the state of capture

isCaptureStart

Description	Get the state of capture
Function	bool isCaptureStart(DeviceHandle_t handle);
Parameters	handle:Handle of the user-designated sensor device
Return	true: the capture is started false:the capture is closed
Note	

3.106 Enable the debug function of trigger pass

setTriggerPassDebug

Description	Enable the debug function of trigger pass
Function	StatusTypeDef setTriggerPassDebug(DeviceHandle_t handle,bool en);

Parameters	handle:Handle of the user-designated sensor device true:enable false:disable
Return	Succeed Return Status_Succeed, Fail Return error description code
Note	

3.107 Clear the flag of trigger pass

clearTriggerPassFlag

Description	Clear the flag of trigger pass
Function	void clearTriggerPassFlag(DeviceHandle_t handle);
Parameters	handle:Handle of the user-designated sensor device
Return	
Note	

3.108 Single shot (Mutiple Channel)

singleShot_MC

Description	Single shot (Mutiple Channel)
Function	StatusTypeDef singleShot_MC(DeviceHandle_t handle, MC_ResultDataTypeDef_t res[]);
Parameters	handle:Handle of the user-designated sensor device res:return the results of channels that have been opened
Return	
Note	

3.109 Get the connect params of CF2000

GE_getControlConnectParam

Description	Get the connect params of CF2000
Function	GE_getControlConnectParam(DeviceHandle_thandle,char *controllerIP, uint16_t *controllerPort,char *controllerMac);
Parameters	handle:Handle of the user-designated sensor device controllerIP controllerPort controllerMac
Return	
Note	

3.110 Set the refractive params of thickness

setThicknessRefractivePara

Description	Set the refractive params of thickness
Function	setThicknessRefractivePara(DeviceHandle_t handle,int channel,int signalIndex1,int signalIndex2,float Nc,float Nd,float Nf);
Parameters	handle:Handle of the user-designated sensor device
Return	
Note	

3.111 Enable the output of RSXXX(Double Channel)

setDoubleChannelRSxxxOutput

Description	Enable the output of RSXXX(Double Channel)
Function	setDoubleChannelRSxxxOutput(DeviceHandle_t handle,bool en);
Parameters	handle:Handle of the user-designated sensor device en: true:enable false:disable
Return	
Note	

IV. Revision history

Date	Revision	Description
8/25/2020	1.1	Update some new APIs.
2/18/2022	1.2	Update APIs GE_openDevice setCalculationThreshold doubleChannelReversePos dumpCacheData dumpCacheData_SC dumpCacheData_MC setChxOffset getChxOffset setControllerIPAddr setControllerPort bindInputPort unbindInputPort getCurFrameRate enableDataCache setEncoderInputMode setEncoderWorkingMode clearEncodersCount isCaptureStart setTriggerPassDebug clearTriggerPassFlag singleShot_MC GE_getControlConnectParam setThicknessRefractivePara setDoubleChannelRSxxxOutput

IMPORTANT NOTICE – PLEASE READ CAREFULLY

Hypersen Technologies Co., Ltd. reserves the right to make changes, corrections, enhancements, modifications and improvements to Hypersen products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on Hypersen products before placing orders. Hypersen products are sold pursuant to Hypersen's terms and conditions of sale in place at the time of order acknowledgment.

Purchasers are solely responsible for the choice, selection and use of Hypersen products and Hypersen assumes no liability for application assistance or the design of Purchasers' products.

No license, express or implied, to any intellectual property right is granted by Hypersen herein.

Re-sale of Hypersen products with provisions different from the information set forth herein shall void any warranty granted by Hypersen for such product.

Hypersen and the Hypersen logo are trademarks of Hypersen. All other products or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.