Sensor API v2.4.0

Copyright

Contact Information
LMI Technologies, Inc.
9200 Glenlyon Parkway
Burnaby BC V5J 5J8
Canada

Telephone: +1 604-636-1011
Fax: +1 604-516-8368

www.lmi3D.com

CONTENTS

1 FocalSpec Sensor API

1.1 Introduction

This document specifies the functions and parameters of the FocalSpec Sensor API (SDK).

1.2 Requirements

1.2.1 Supported Platforms

- Windows 10 Professional 32-bit or 64-bit
- Linux (tested to work in Ubuntu 64-bit 18.04.1 LTS, although there are not any distribution-specific requirements)

1.2.2 Hardware Requirements

- Gigabit Ethernet for LCI401, LCI1200, LCI1201 and LCI1600 sensors
- 10 Gigabit Ethernet for LCI1220 and LCI1620 sensors

1.2.3 Software Compatibility Chart

Open *FW FSSDK Sensor Compatibility Chart.pdf* from *FocalSpec Software Development Kit\API* folder.

FocalSpec sensor Firmware (FW), FocalSpec Software Development Kit (FSSDK) and FocalSpec LCI sensor compatibility chart

Version:

Date: September 21 2020

Modifier: Esa Kalistaja

X Not compatible

X Compatible

X Compatible and recommended combination

	FsVevo FW 4.7.1	FsVevo FW 4.7.4	FsVevo FW 4.8.2	FsVevo FW 4.9.0	FsVevo FW 4.9.1	FsMega FW 1.1.3	FsMega FW 1.2.0	FsMega FW 1.3.0	FsMega FW 1.3.1	FsMega FW 1.3.2	FsMega FW 1.4.0	FSSDK 1.50.0	FSSDK 1.50.2	FSSDK 1.51.0	FSSDK 1.52.0	FSSDK 2.0.0	FSSDK 2.0.1	FSSDK 2.1.0	FSSDK 2.2.0	FSSDK 2.3.0	FSSDK 2.4.0	LCI401	LCI1200	LCI1201	LCI1600	LCI1220	LCI1620
FsVevo FW 4.7.1												X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
FsVevo FW 4.7.4												X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
FsVevo FW 4.8.2												X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
FsVevo FW 4.9.0												X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
FsVevo FW 4.9.1												X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
FsMega FW 1.1.3												X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
FsMega FW 1.2.0												X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
FsMega FW 1.3.0												X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
FsMega FW 1.3.1												X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
FsMega FW 1.3.2												X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
FsMega FW 1.4.0												X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
FSSDK 1.50.0	X	X	X	X	X	X	X	X	X	X	X											X	X	X	X	X	X
FSSDK 1.50.2	X	X	X	X	X	X	X	X	X	X	X											X	X	X	X	X	X
FSSDK 1.51.0	X	X	X	X	X	X	X	X	X	X	X											X	X	X	X	X	X
FSSDK 1.52.0	X	X	X	X	X	X	X	X	X	X	X											X	X	X	X	X	X
FSSDK 2.0.0	X	X	X	X	X	X	X	X	X	X	X											X	X	X	X	X	X
FSSDK 2.0.1	X	X	X	X	X	X	X	X	X	X	X											X	X	X	X	X	X
FSSDK 2.1.0	X	X	X	X	X	X	X	X	X	X	X											X	X	X	X	X	X
FSSDK 2.2.0	X	X	X	X	X	X	X	X	X	X	X											X	X	X	X	X	X
FSSDK 2.3.0	X	X	X	X	X	X	X	X	X	X	X											X	X	X	X	X	X
FSSDK 2.4.0	X	X	X	X	X	X	X	X	X	X	X											X	X	X	X	X	X
LCI401	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X						
LCI1200	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X						
LCI1201	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X						
LCI1600	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X						
LCI1220	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X						
LCI1620	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X						

1.3 Installation

1.3.1 Windows

Run *setup-X.Y.Z.exe* from the USB drive or SharePoint to install FSSDK. X.Y.Z is the FSSDK version number.

1.3.2 Linux

Download Intel IPP library from Intel: https://software.intel.com/en-us/ipp. Unpack the downloaded package and install it by running the included install_GUI.sh or install.sh as root. By default, the library will be installed in /opt/intel/ipp.

Download Intel MKL library from Intel: https://software.intel.com/en-us/mkl. Unpack the downloaded package and install it by running the included

install_GUI.sh or install.sh as root. By default, the library will be installed in /opt/intel/mkl.

Unpack fssdk-X.Y.Z-linux.tar.gz from the USB drive or SharePoint to the installation folder. X.Y.Z is the FSSDK version number. Default target folder: ./fssdk.

```
tar xzvf fssdk-X.Y.Z-linux.tar.gz
```

Installation option 1: installer script.

Run fssdk/install.sh as root. By default, the target installation folder is /opt/fssdk. You can change this by option --target=TARGET. Use option --help for help.

```
# Installation to /opt/fssdk:
cd fssdk
sudo ./install.sh
```

This will:

- Copy files to target folder.
- Generate fssdk-env.sh file which will contain the required environment variables.

Either run fssdk-env.sh from profile script (such as ~/.bashrc), or copy-paste its content to the profile script.

Make sure the Intel IPP and MKL library path variables in the fssdk-env.sh are correct.

Installation option 2: manual installation.

Copy the FSSDK files to a desired location.

Set the following environment variables:

- FSSDK_PATH: path to the top folder of FSSDK installation (e.g. /opt/fssdk).
- IPP_PATH: path to Intel IPP library (e.g. /opt/intel/ipp).
- MKL_PATH: path to Intel MKL library (e.g. /opt/intel/mkl).
- LD_LIBRARY_PATH: add the path to FSSDK API folder (FSSDK_PATH/API), and IPP and MKL libraries (IPP_PATH/lib/intel64, MKL_PATH/lib/intel64).

The environment variables can be set by adding the following lines to ~/.bashrc or other profile script. Modify paths as necessary.

```
# Path to FSSDK
export FSSDK_PATH=/opt/fssdk
# Path to Intel IPP library
export IPP_PATH=/opt/intel/ipp
# Path to Intel MKL library
export MKL_PATH=/opt/intel/mkl
# Library search path
export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$FSSDK_PATH/API:$IPP_PATH/lib/intel64:$MKL_PATH/lib/i
ntel64
```

To apply environment variables immediately, run the following command:

```
source ~/.bashrc
```

1.4 Folder Structure

- API - The API.
- ConsoleExample - Example cross-platform C/C++ project to start with.
- Drivers - Drivers needed to run the API. Windows-only.
- GuiExample - Example C#/.Net application with graphical user interface (Windows Forms) to start with. Windows-only.
- FocalSpec SDK Demo - Simplified example C#/.Net application with graphical user interface (XAML). Windows-only.
- LabViewExample - Example LabView application.
- Manual - This documentation.

1.4.1 API Files

1.4.1.1 Windows

Library	Description
Vevo.dll	FSAPI DLL
Vevo.lib	FSAPI library file for DLL linking
FsApiNet.dll	FSAPI .Net wrapper library.
vevo-win32.dll	Communication library used by Win32 Vevo.dll
vevo-x64.dll	Communication library used by x64 Vevo.dll

1.4.1.2 Linux

Library	Description
libvevo.so	FSAPI shared library
libvevobase.so	Communication library used by libvevo.so

1.4.1.3 Header Files

Header	Description
CameraDll.h	FSAPI library function signatures
Callback.h	Callback function signatures
CameraStatus.h	Return value listing
PeakStructure.h	Peak format description
VevoParameterDefinitions.h	FSAPI parameters

1.5 Creating an Application

In short, the API design follows a common practice:

1. open API
2. initialize a camera
3. start receiving data
4. do something with the data
5. stop receiving data
6. close API

In function level, the flow is usually as follows:

```
int camera_count = 1; // One camera expected
char **camera_ids = NULL;

// Open API and discover cameras that respond within 2000ms. If expected camera count
is not reached an additional 20 seconds timeout is waited.
// As a result, we get number of cameras and their IDs that we need to control them.
_Open(&camera_count, &camera_ids, 2000)

// Connect to a camera of interest. Let it be the first one. Use auto-ip.
_Connect(camera_ids[0], NULL);

int calibrations_in_camera = 0;
_GetIntParameter(camera_ids[0], const_cast<char*>(PARAM_SENSOR_DATA_IN_FLASH),
&calibrations_in_camera);

if(!calibrations_in_camera)
{
   // Assign Z calibration file.
   _SetStringParameter(camera_ids[0], PARAM_SENSOR_CALIBRATION_FILE,
"C:\\temp\\zcalibration.calib");

   // Assign X calibration file.
   _SetStringParameter(camera_ids[0], PARAM_SENSOR_X_CALIBRATION_FILE,
"C:\\temp\\xcalibration.calib");
}

_SetStringParameter(camera_ids[0], PARAM_LOAD_RECIPE, "Recipe"));

// Set sorting order for line callback
_SetLineSortingOrder(camera_ids[0], SurfaceSorting::FROM_TOP_TO_BOTTOM);

// Register a callback function
_SetLineCallback(camera_ids[0], 0, LineCallbackHandler);

// Start data reception, which will automatically fire your corresponding callback
_StartGrabbing(camera_ids[0]);
```

```
// Do some application specific operations at the callback.
// ...

// Time to close application. Stop data reception and close the API.
 StopGrabbing(camera_ids[0]);
 Close();
```

Below is an example of a line callback.

```
void LineCallbackHandler(char *cameraID, int layerId, float *zValues, float
*intensityValues, int line_length, double xStep, HEADER *header)
{
    // Place you application code here.

    // Make this function fast and non-blocking (parameter points_in_queue should not
accumulate over time).
    // One approach to achieve this is to push values into a buffer, which is
processed by another consumer thread.
}
```

1.6 Example Projects

There are four example projects with source codes included:

1. GuiExample: graphical user interface example, written in C#/.Net, Windows only. To open and run it, please open *GuiExample\GuiExample.sln* with VisualStudio.
2. FocalSpec SDK Demo: simplified graphical user interface example, written in C#/.Net, Windows only. To open and run it, please open *FocalSpec SDK Demo\FocalSpec_SDK_Demo.sln* with VisualStudio.
3. ConsoleExample: cross-platform CLI example, written in C/C++.
4. LabViewExample: example GUI application, written in LabView.

It is recommended to start with *GuiExample*, since they provide visual feedback. More information about GuiExample can be found from *manual.pdf*.

1.6.1 Building ConsoleExample

1.6.1.1 Windows

Open *ConsoleExample\ConsoleExample.sln* and read instructions from *ConsoleExample.cpp*.

1.6.1.2 Linux

Install the necessary build tools, such as build-essential and cmake.
Build with CMake:

```
cd ConsoleExample
mkdir build
cd build
cmake ..
make
```

1.7 Change Log

Change log is provided as a PDF file (Changelog.pdf) contained in the *API* folder of the SDK installation folder.

---

2 Data Acquisition

This page describes how to use callback functions to receive data from the sensor.

2.1 Line callback

The recommended way to acquire 3D data is to use line callback configuration (_SetLineCallback). Benefits of this mode are that it provides 3D points with a fixed x-size and it has many preprocessing options available. All preprocessing functions provided by the line callback are available also in batch callback.

If a line callback is configured following preprocessing functions are available:

- Noise removal (PARAM_NOISE_REMOVAL)
- Sorting of points to layers according to the SurfaceSorting mode (SurfaceSorting)
- Detection of missing first layer (Detect Missing First Layer)
- Interpolation of missing surface points (PARAM_FILL_GAP_X_MAX)
- Median filtering (PARAM_MEDIAN_Z_FILTER_SIZE, PARAM_MEDIAN_INTENSITY_FILTER_SIZE)
- Averaging filtering (PARAM_AVERAGE_Z_FILTER_SIZE, PARAM_AVERAGE_INTENSITY_FILTER_SIZE)
- Resampling to the configured x-resolution (PARAM_RESAMPLE_LINE_X_RESOLUTION)

Application shall call _SetLineCallback separately for each layer. The configuration specifies how many layers are expected. For example, if a user wants to have callback for 3 top layers following functions shall be called:

```
_SetLineSortingOrder(camera_ids[0], SurfaceSorting::FROM_TOP_TO_BOTTOM);
_SetLineCallback(camera_ids[0], 0, LineCallbackHandler);
_SetLineCallback(camera_ids[0], 1, LineCallbackHandler);
_SetLineCallback(camera_ids[0], 2, LineCallbackHandler);
```

2.2 Batch callback

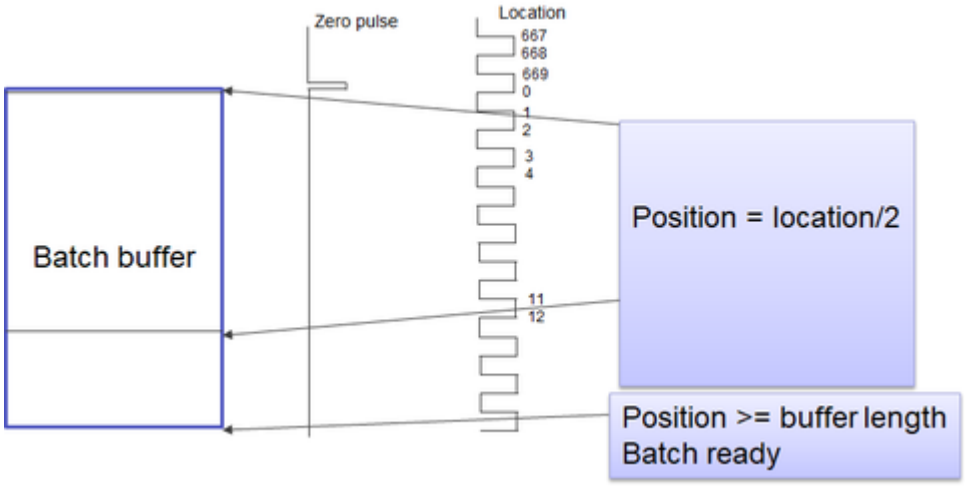
The batch callback option (_SetBatchCallback) can be used if the application does not need to process the profile data when acquired. SDK's batch functionally collects profiles until number of profiles (PARAM_BATCH_LENGTH) is reached. It is also possible to define a timeout for the callback (PARAM_BATCH_TIMEOUT).

```
status = static_cast<CameraStatus>(_SetIntParameter(camera_ids[0],
const_cast<char*>(PARAM_PEAK_ENABLE), 1));
status = static_cast<CameraStatus>(_SetLineSortingOrder(camera_ids[0],
SurfaceSorting::FROM_TOP_TO_BOTTOM));
status = static_cast<CameraStatus>(_SetIntParameter(camera_ids[0],
const_cast<char*>(PARAM_BATCH_LENGTH), 1000));
status = static_cast<CameraStatus>(_SetIntParameter(camera_ids[0],
const_cast<char*>(PARAM_BATCH_TIMEOUT), 5000));
status = static_cast<CameraStatus>(_SetBatchCallback(camera_ids[0], 0,
BatchCallbackHandler));
status = static_cast<CameraStatus>(_StartGrabbing(camera_ids[0]));
```

2.3 Recommendations for buffering by using encoder position

When the line callback is used the application typically collects the data to fixed size batches. Below are recommendations listed for the application software.

1. Preallocate a buffer for the batch. Usually application knows how many profiles are needed.
2. When a profile is received copy it to the proper buffer location.
3. When the last line is received the buffer is ready and in order



For continuous inspections it is recommended to use double buffering. One is used for profile acquisition and other for processing.

2.4 Raw image callback

The raw image callback can be used when the gray scale image for the camera is needed. Note that the imaging frequency shall be low enough to fit data to the Ethernet.

```
status = static_cast<CameraStatus>( SetIntParameter(camera_ids[0], PARAM_PEAK_ENABLE,
0));
status = static_cast<CameraStatus>( SetIntParameter(camera_ids[0],
PARAM_REG_PULSE_FREQ, 50));
status = static_cast<CameraStatus>( SetRawImageCallback(camera_ids[0],
RawImageCallbackHandler));
status = static_cast<CameraStatus>( StartGrabbing(camera_ids[0]));
```

2.5 Profile callback

The profile callback can be used if the application needs all points from the sensor without preprocessing.

Point cloud provided by Profile callback contains all detected peaks up to defined limit (PARAM_MAX_POINT_COUNT)

```
status = static_cast<CameraStatus>( SetIntParameter(camera_ids[0], PARAM_PEAK_ENABLE,
1));
status = static_cast<CameraStatus>( SetIntParameter(camera_ids[0],
PARAM_REG_PULSE_FREQ, SENSOR_FREQUENCY));
status = static_cast<CameraStatus>( SetProfileCallback(camera_ids[0],
ProfileCallbackHandler));
status = static_cast<CameraStatus>( StartGrabbing(camera_ids[0]));
```

See FSDSK source file ConsoleExample.cpp basic callback function implementation ProfileCallback.

2.6 Thickness calculation

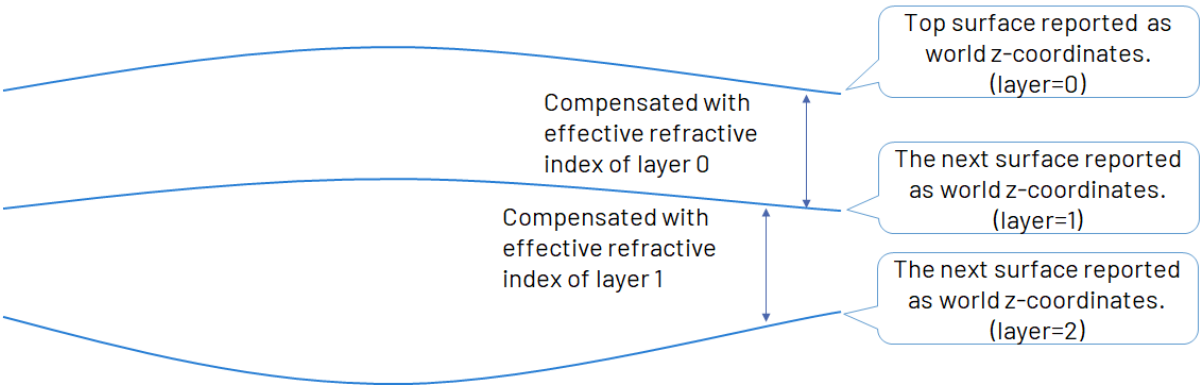
Effective refractive index correction and thickness calculation can be enabled for the profiles given by the lineCallback and batchCallback functions. This is useful for transparent layers where layer thicknesses are needed to be analyzed.

End user needs to configure effective refractive index for each layer separately with _SetLayerFloatParameter() function. This calibration can be done by measuring known layer and calculating ratio of real thickness and measured thickness. This will adjust z-coordinates of layers in the world coordinate system.

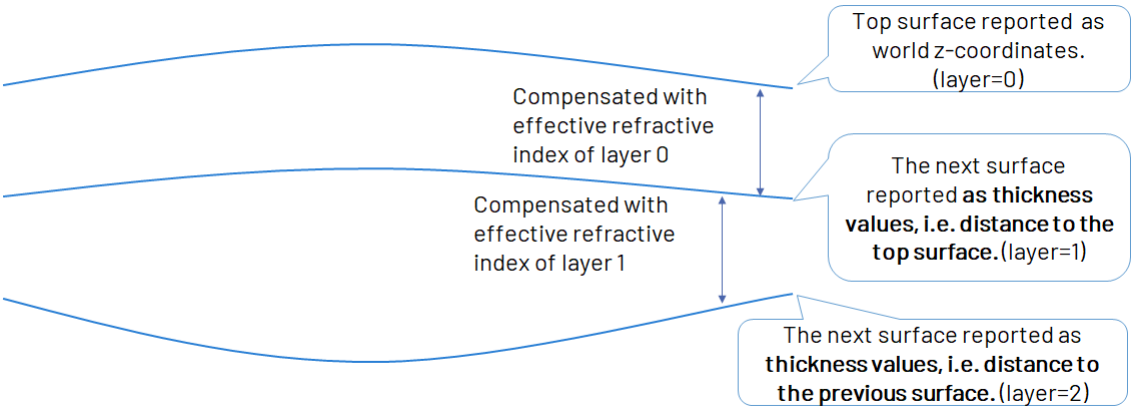
```
_SetLayerFloatParameter(camera_ids[0],0, PARAM_LAYER_EFFECTIVE_REFRACTIVE_INDEX,
1.333)); // Refractive index for the first layer
_SetLayerFloatParameter(camera_ids[0],1, PARAM_LAYER_EFFECTIVE_REFRACTIVE_INDEX,
1.21)); // The second layer
_SetLayerFloatParameter(camera_ids[0],2, PARAM_LAYER_EFFECTIVE_REFRACTIVE_INDEX,
1.30)); // The third layer
```

Additionally, a user can set PARAM_THICKNESS_MODE to enable thickness calculation. In this mode the first layer contains a top surface as world coordinate points, and following layers are precalculated thicknesses in micrometers.

Z-coordinates when Thickness mode is disabled:



Z-coordinates when Thickness mode is enabled:



3 Sensor Configuration

This page clarifies usage of sensor parameters in the SDK.

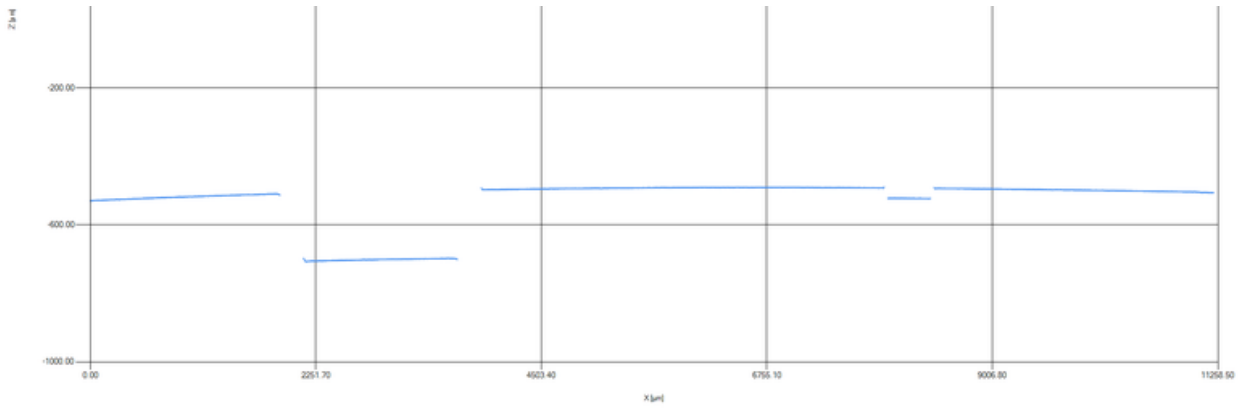
3.1 Sensor Modes

The sensor works either in peak or raw data mode. Changing of mode during grabbing is a slow operation (typically about 1 second).

3.1.1 The Peak Mode (default)

Typically, peak mode is used. In this mode the sensor processes a raw image and provides points to the application. Each point output contains x coordinate, height and intensity values.

Example of received 3D points when two grooves are measured:



3.1.2 Raw Mode

In a raw mode gray scale image from the sensor is captured. This mode is useful when sensor capabilities are studied.

Example of raw image where two grooves are measured:

C++:

```
CameraStatus status = static_cast<CameraStatus>( SetIntParameter(sensor_ids[0],
PARAM_PEAK_ENABLE, 1));
if (status != CameraStatus::CAMERA_OK)
{
        printf("FSAPI error: %d\n", status);
}
```

C#:

```
var cameraStatus = _sensor.SetParameter(_sensorId, SensorParameter.PeakEnabled, 1);
```

Maximum size of the raw image equals sensor size. Sensor size for LCI401, LCI1200, LCI1201 and LCI1600 sensors is 1088 (vertical) x 2048 (horizontal) pixels. For LCI2020 and LCI1620 sensors size is 1400 (vertical) x 1728 (horizontal) pixels.

Maximum frequency of raw image is limited by the Ethernet interface. Required throughput can be calculated by formula image_height x 2048 x imaging_frequency.

Disable peak mode:

C++:

```
CameraStatus status = static_cast<CameraStatus>( SetIntParameter(sensor_ids[0],
PARAM_PEAK_ENABLE, 0));
if (status != CameraStatus::CAMERA_OK)
{
        printf("FSAPI error: %d\n", status);
}
```

C#:

```
var cameraStatus = _sensor.SetParameter(_sensorId, SensorParameter.PeakEnabled, 0);
```

3.2 Calibration Files

The sensor is calibrated in the FocalSpec's production to produce exact micrometer coordinates. Calibration files are stored in the sensors for those which are manufactured after 02/2019. If files are not in the sensor those must be set before image grabbing.

The sensor must be in the peak mode when calibration files are used. After files are set sensor unit must be set to use micrometers. If unit is not changed pixel units without calibration are used.

3.2.1 Use calibration files from the sensor

Following examples check whether calibration files exist in the sensor or not. If files exist only peak unit change is needed.

```
CameraStatus status = static_cast<CameraStatus>(_SetIntParameter(sensor_ids[0],
PARAM_PEAK_ENABLE, 1));

int calibrations_in_sensor = 0;
_GetIntParameter(sensor_ids[0], const_cast<char*>(PARAM_SENSOR_DATA_IN_FLASH),
&calibrations_in_sensor);

if (!calibrations_in_sensor)
{
   _SetStringParameter(sensor_ids[0], PARAM_SENSOR_CALIBRATION_FILE,
"C:\\temp\\zcalibration.calib");
   _SetStringParameter(sensor_ids[0], PARAM_SENSOR_X_CALIBRATION_FILE,
"C:\\temp\\xcalibration.calib");
}

_SetIntParameter(sensor_ids[0], PARAM_PEAK_Y_UNIT, 1);
_SetIntParameter(sensor_ids[0], PARAM_PEAK_X_UNIT, 1);
```

C#:

```
var cameraStatus = _sensor.SetParameter(_sensorId, SensorParameter.PeakEnabled, 1);
if (cameraStatus != CameraStatusCode.Ok)
   return cameraStatus;

_sensor.GetParameter(_sensorId, SensorParameter.SensorDataInFlash, out int
calibrationsInCamera);
if (calibrationsInCamera == 1)
{
    cameraStatus = _sensor.SetParameter(_sensorId, SensorParameter.ZCalibrationFile,
"C:\\temp\\lci1200z.dat");
    if (cameraStatus != CameraStatusCode.Ok)
       return cameraStatus;
```

```
    cameraStatus = _sensor.SetParameter(_sensorId, SensorParameter.XCalibrationFile,
"C:\\temp\\lci1200x.dat");
    if (cameraStatus != CameraStatusCode.Ok)
       return cameraStatus;
}

cameraStatus = _sensor.SetParameter(_sensorId, SensorParameter.PeakXUnit, 1);
if (cameraStatus != CameraStatusCode.Ok)
   return cameraStatus;
cameraStatus = _sensor.SetParameter(_sensorId, SensorParameter.PeakYUnit, 1);
if (cameraStatus != CameraStatusCode.Ok)
   return cameraStatus;
```

3.2.2 Set calibration files manually

C++:

```
CameraStatus status = static_cast<CameraStatus>( SetIntParameter(sensor_ids[0],
PARAM_PEAK_ENABLE, 1));
if (status != CameraStatus::CAMERA_OK)
{
     printf("FSAPI error: %d\n", status);
     return;
}
status = static_cast<CameraStatus>( SetStringParameter(sensor_ids[0],
PARAM_SENSOR_CALIBRATION_FILE, "C:\\temp\\lci1200z.dat"));
if (status != CameraStatus::CAMERA_OK)
{
     printf("FSAPI error: %d\n", status);
     return;
}
status = static_cast<CameraStatus>( SetStringParameter(sensor_ids[0],
PARAM_SENSOR_X_CALIBRATION_FILE, "C:\\temp\\lci1200x.dat"));
if (status != CameraStatus::CAMERA_OK)
{
     printf("FSAPI error: %d\n", status);
     return;
}
status = static_cast<CameraStatus>( SetIntParameter(sensor_ids[0], PARAM_PEAK_Y_UNIT,
1));
if (status != CameraStatus::CAMERA_OK)
{
     printf("FSAPI error: %d\n", status);
     return;
}
status = static_cast<CameraStatus>( SetIntParameter(sensor_ids[0], PARAM_PEAK_X_UNIT,
1));
if (status != CameraStatus::CAMERA_OK)
{
```

```
    printf("FSAPI error: %d\n", status);
    return;
}
```

C#:

```
var cameraStatus = _sensor.SetParameter(_sensorId, SensorParameter.PeakEnabled, 1);
if (cameraStatus != CameraStatusCode.Ok)
   return cameraStatus;
cameraStatus = _sensor.SetParameter(_sensorId, SensorParameter.ZCalibrationFile,
"C:\\temp\\lci1200z.dat");
if (cameraStatus != CameraStatusCode.Ok)
   return cameraStatus;
cameraStatus = _sensor.SetParameter(_sensorId, SensorParameter.XCalibrationFile,
"C:\\temp\\lci1200x.dat");
if (cameraStatus != CameraStatusCode.Ok)
   return cameraStatus;
cameraStatus = _sensor.SetParameter(_sensorId, SensorParameter.PeakXUnit, 1);
if (cameraStatus != CameraStatusCode.Ok)
   return cameraStatus;
cameraStatus = _sensor.SetParameter(_sensorId, SensorParameter.PeakYUnit, 1);
if (cameraStatus != CameraStatusCode.Ok)
   return cameraStatus;
```

3.3 Imaging Frequency

Camera works either in internal (free run) or external pulsing mode. In free run mode user can set imaging frequency. In external pulsing mode frequency is set to 0 and the frequency is defined by an external triggering to the camera I/O. Maximum imaging frequency is limited by

1. sensor's capability,
2. LED pulse duration or
3. Ethernet throughput in raw image mode.

3.3.1 Sensor capability

Maximum camera frequency is increased when depth of field is decreased. Following formulas can be used to calculate maximum window height in pixels when a frequency is known.

Recommended way to calculate maximum frequency is to use _AdjustRoiAndFps() function.

```
_sensor.AdjustRoiAndFps(_sensorId, (int)(RoiOperation.GetFpsForHeight), height, out
var frequency);
```

For LCI401, LCI1200, LCI1201 and LCI1600 sensors maximum frequency is 5000 Hz and height can be calculate with the following equation: int height = (int)(47.5 / 129 * (1e6 / frequency - 1548 / 47.5)); Safety margin. height -= (int)(0.2 * height);

See Depth of Field chapter to map height into micrometers.

3.3.2 LED Pulse Duration

For LCI401, LCI1200, LCI1201 and LCI1600 sensors LED pulse duration must be at least 100 us less than minimum interval between two image captures. For example, if maximum imaging frequency is 500 Hz maximum LED pulse duration is (1/500 Hz) * 1000 * 1000-100 us = 1900 us.

For LCI1220 and LCI1620 sensors LED pulse duration must be at least 31 us less than minimum interval between two image captures.

Recommended way to calculate maximum led pulse duration is to use _AdjustRoiAndFps() function.

```
_sensor.AdjustRoiAndFps(_sensorId, (int)(RoiOperation.GetLedPulseForFps), frequency,
out var pulseLength);
```

3.3.3 Ethernet Throughput

In peak mode required ethernet throughput can be calculated by following formula: imaging_frequency * 64 bits/points * points_per_frame

For example, if we have two surfaces and imaging frequency is 500 Hz resulting Ethernet throughput is

1. LCI401, LCI1200, LCI1201 and LCI1600 sensors: 500 Hz * 64 * 2048 *2 = 128 Mbps.
2. LCI1220 and LCI1620 sensors: 500 Hz * 64 * 1728 *2 = 111 Mbps.

In raw mode required throughput is:

1. LCI401, LCI1200, LCI1201 and LCI1600 sensors: image_height x 2048 x imaging_frequency
2. LCI1220 and LCI1620 sensors: image_height x 1728 x imaging_frequency

3.4 Depth of Field (Height and Offset of the Image)

Sensor's maximum imaging frequency can be increased by decreasing the image height.

Recommended way to adjust PARAM_IMAGE_HEIGHT and PARAM_IMAGE_OFFSETY parameters is to use _AdjustRoiAndFps() function with SetRoiForFps or RoiOperation.SetFpsForRoi attribute.

Example to adjust imaging height and offset when maximum imaging frequency is known:

```
_sensor.AdjustRoiAndFps(_sensorId, (int)(RoiOperation.SetRoiForFps), freq, out var
zHeight);
```

Example to adjust imaging height and offset when needed depth of field is known. The maximum imaging frequency is returned.

```
_sensor.AdjustRoiAndFps(_sensorId, (int)(RoiOperation.SetFpsForRoi), height, out var
frequency);
```

3.5 Exposure Control

Amount of light in the sensor can adjusted in three ways

1. changing LED pulse length,
2. changing LED current or
3. changing gain of the sensor.

3.5.1 LED Pulse Width

Usually exposure is controlled by setting proper value for the LED pulse width. Width is given as microseconds typically varying between 1 us (high gloss, such as a mirror surface) and 100 us, but it can be also higher in low gloss surfaces.

C++:

```
float pulseWidth = 15.5;

CameraStatus
status = static_cast<CameraStatus>( SetFloatParameter(sensor_ids[0],
PARAM_REG_PULSE_WIDTH_FLOAT, pulseWidth ));
if (status != CameraStatus::CAMERA_OK)
{
     printf("FSAPI error: %d\n", status);
     return;
}
```

C#:

```
float pulseWidth = 15.5;

CameraStatusCode status = _sensor.SetParameter(_sensorId,
SensorParameter.PulseWidthFloat, pulseWidth );

if (status != CameraStatusCode.Ok)
     return status;
```

AGC is a camera function to adjust LED pulse width automatically to a proper value. This functionality can be used if the surface is continuous and no rapid changed are expected. The controller continuously calculates average intensity value of measured 3D points and compares this value to predefined value. LED

pulse width is adjusted to minimize the error between measured and predefined value. Adjusted LED pulse width can be read from Header.PulseWidth of each received profile.

C++:

```
CameraStatus status;
int agcEnabled = 1; // Enable AGC
float agcTarget = 80.0; // Average intensity of measured points
int agcPulseWidthLimit = 300; // in microseconds
int agcMinPulseWidthLimit =10;// in microseconds
float agcGain = 1.2; // Gain for AGC controller

status = static_cast<CameraStatus>( SetIntParameter(sensor_ids[0], PARAM_AGC_ENABLED,
agcEnabled));
if (status != CameraStatus::CAMERA_OK)
    return 1;

if (agcEnabled)
{
    status = static_cast<CameraStatus>( SetFloatParameter(sensor_ids[0],
PARAM_AGC_TARGET, agcTarget));
    if (status != CameraStatus::CAMERA_OK)
        return 1;
    status = static_cast<CameraStatus>( SetIntParameter(sensor_ids[0],
PARAM_AGC_PULSE_WIDTH_LIMIT, agcPulseWidthLimit));
    if (status != CameraStatus::CAMERA_OK)
        return 1;
    status = static_cast<CameraStatus>( SetIntParameter(sensor_ids[0],
PARAM_AGC_MIN_PULSE_WIDTH_LIMIT, agcMinPulseWidthLimit));
    if (status != CameraStatus::CAMERA_OK)
        return 1;
    status = static_cast<CameraStatus>( SetFloatParameter(sensor_ids[0],
PARAM_AGC_GAIN, agcGain));
    if (status != CameraStatus::CAMERA_OK)
        return 1;
}
```

C#:

```
CameraStatusCode status;
int agcEnabled = 1; // Enable AGC
float agcTarget = 80.0; // Average intensity of measured points
int agcPulseWidthLimit = 300; // in microseconds
int agcMinPulseWidthLimit =10;// in microseconds
float agcGain = 1.2; // Gain for AGC controller

status = _sensor.SetParameter(_sensorId, SensorParameter.AgcEnabled, agcEnabled);
if (status != CameraStatusCode.Ok)
    return status;
```

```
if (agcEnabled == 1)
{
    _sensor.SetParameter(_sensorId, SensorParameter.AgcWiLimit, agcPulseWidthLimit);
    _sensor.SetParameter(_sensorId, SensorParameter.AgcMinPulseWidthLimit,
agcMinPulseWidthLimit);
    _sensor.SetParameter(_sensorId, SensorParameter.AgcTarget, agcTarget);
    _sensor.SetParameter(_sensorId, SensorParameter.AgcGain, agcGain);
}
```

3.5.2 LED Current

Used current for the LED lighting can be tuned. There are separate values for edge and center areas of the illuminator which can be used to smooth the illumination profile. Typically 1 ampere current is used.

C++:

```
CameraStatus fsapi_status;

float pulseCurrentMiddle = 1.0;
float pulseCurrentEdges = 1.0;

fsapi_status = m_fsapi->SetDoubleParameter(head_id_cstr, PARAM_PULSE_CURRENT_MIDDLE,
pulseCurrentMiddle);
if (fsapi_status != CAMERA_OK)
        return fsapi_status;

fsapi_status = m_fsapi->SetDoubleParameter(head_id_cstr, PARAM_PULSE_CURRENT_EDGES,
pulseCurrentEdges);
if (fsapi_status != CAMERA_OK)
        return fsapi_status;
```

C#:

```
float pulseCurrentMiddle = 1.0;
float pulseCurrentEdges = 1.0;
CameraStatusCode status;

status = _sensor.SetParameter(_sensorId, SensorParameter.PulseCurrentMiddle,
pulseCurrentMiddle);
if (status != CameraStatusCode.Ok)
    return status;
status = _sensor.SetParameter(_sensorId, SensorParameter.PulseCurrentEdges,
pulseCurrentEdges);
if (status != CameraStatusCode.Ok)
    return status;
```

3.5.3 Gain of the Sensor

The gain controls the amplification of the signal from the camera sensor. When the gain is increased also background noise is increased. Typically there is no need to change default gain value.

For LCI401, LCI1200, LCI1201 and LCI1600 sensors the default gain value is 1 and the range is 1.0 - 3.2. For LCI1220 and LCI1620 sensors the default gain value is 2 and the range is 2.0 - 8.0.

C++:

```
CameraStatus status;
float gain = 1.0;
status = static_cast<CameraStatus>( SetIntParameter(sensor_ids[0], PARAM_GAIN, gain));
if (status != CameraStatus::CAMERA_OK)
     return 1;
```

C#:

```
float gain = 1.0;
CameraStatusCode status;

status = _sensor.SetParameter(_sensorId, SensorParameter.Gain, gain);
if (status != CameraStatusCode.Ok)
  return status;
```

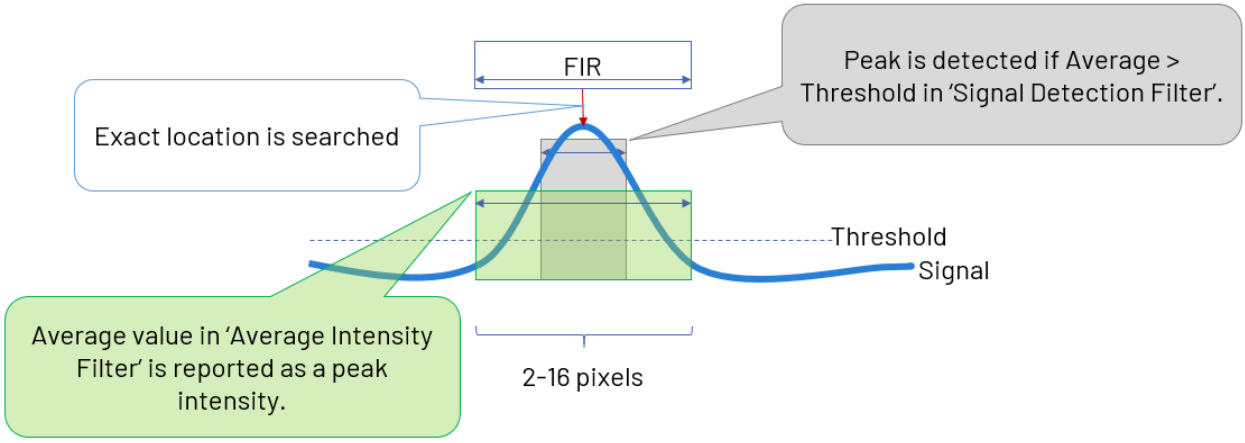
3.6 3D Point Detection

3D points detection starts with applying Signal Detection Filter to the signal. If the averaged value is greater than a specified threshold a peak is detected. After that exact location of the peak is calculated with FIR. Finally, an average peak intensity is calculated with Average Intensity Filter. Length of filters and threshold are user configurable. Usually filter length 16 gives the best results if material type is opaque. A shorter value may be needed if highly glossy surface (like mirror) is measured. Also, transparent surfaces may need lower values since layers needs to be separated from each other.

User has a possibility to select which intensity value is returned per layer. This is useful in multilayer surfaces where some layers might be thinner.

Parameters:

1. PARAM_SIGNAL_DETECTION_FILTER_LENGTH
2. PARAM_PEAK_FIR_LENGTH
3. PARAM_PEAK_AVERAGE_INTENSITY_FILTER_LENGTH
4. PARAM_PEAK_THRESHOLD
5. PARAM_LAYER_INTENSITY_TYPE



Recommended way to set all peak detection parameters is to use _SetPeakDetectionParameters() function.

C++:

```
_SetFloatParameter(sensor_ids[0], const_cast<char*>(PARAM_LAYER_MIN_THICKNESS), 80);
_SetPeakDetectionParameters(sensor_ids[0], MaterialType::Transparent,
DetectionSensitivity::BestAccuracy);
```

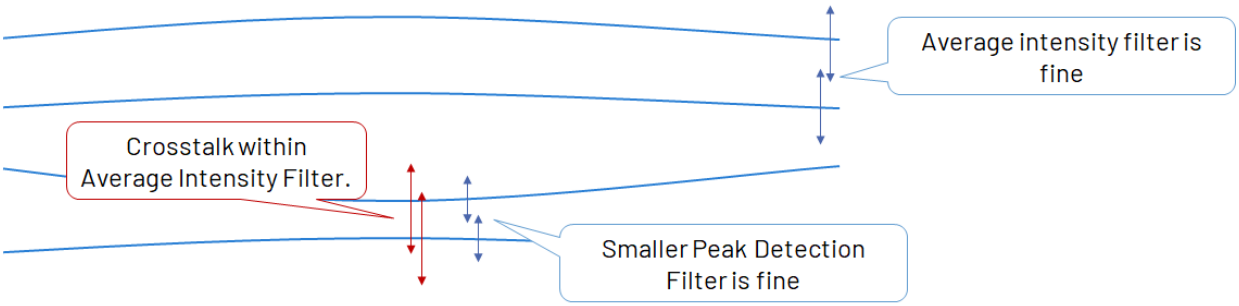
C#:

```
_sensor.SetParameter(_sensorId, SensorParameter.LayerMinThickness, 80);
_sensor.SetPeakDetectionParameters(_sensorId, MaterialType.Transparent,
DetectionSensitivity.BestAccuracy);
```

After _SetPeakDetectionParameters() function threshold and filter parameters can be tuned.

3.7 Intensity Type Selection

By default, intensity is calculated with Average Intensity Filter. If transparent material has a thin layer the filter might be too large. In order to reduce crosstalk user can select filter for each layer separately.



3.8 Reordering of Profiles

This chapter applies only for LCI401, LCI1200, LCI1201 and LCI1600 sensors. In LCI1220 and LCI1620 sensors reordering is not needed.

Received profiles are not guaranteed to arrive in order without reordering. All profiles have a header where order is in header->index attribute. Normally index increases linearly but sometimes there are out of order profiles caused by the Ethernet buffering.

You have two options to do reordering. One is to check the index in the application and do necessary processing, or you can enable reordering in the FS SDK.

FS SDK's reordering functionality buffers profiles until all are in order. Downside of reordering is that if packets are out of order jitter of the profiles will be increase.

Reordering uses 500 ms timeout for waiting pending packets. All received profiles are delivered to the application even though the timeout is expired. This is very unlikely but may happen if the PC is stalled longer than the timeout.

Reordering could be useful to activate on the cased when higher reception frequency ~5000Hz are used and frames are constantly being delivered not in order.

C++:

```
// Enable line reordering
status = static_cast<CameraStatus>(_SetIntParameter(sensor_ids[0], PARAM_REORDERING,
1));
if (status != CameraStatus::CAMERA_OK)
{
    printf("FSAPI error: %d\n", status);
    return 1;
}
// defines reordering span time window in milliseconds
status = static_cast<CameraStatus>(_SetIntParameter(*sensor_ids[0],
PARAM_REORDERING_SPAN, 300));
if (status != CameraStatus::CAMERA_OK)
{
    printf("FSAPI error: %d\n", status);
    return 1;
}

// Define maximum index deviation unless reordering buffer is cleared.
status = static_cast<CameraStatus>(_SetIntParameter(*sensor_ids[0],
PARAM_REORDERING_DEVIATION, 3000));
if (status != CameraStatus::CAMERA_OK)
{
    printf("FSAPI error: %d\n", status);
    return 1;
}
```

C#:

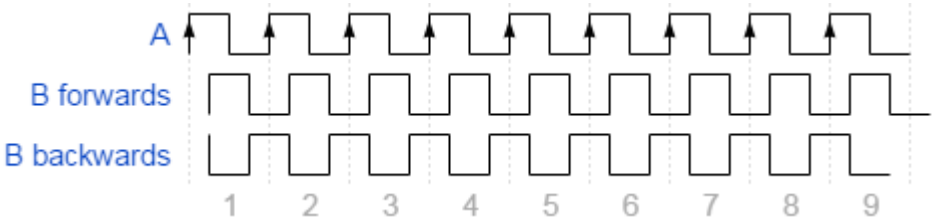
```
cameraStatus = _sensor.SetParameter(_sensorId, SensorParameter.Reordering, 1);
if (cameraStatus != CameraStatusCode.Ok)
    return cameraStatus;

cameraStatus = _sensor.SetParameter(_sensorId, SensorParameter.
SensorParameter.ReorderingSpan, 500);
if (cameraStatus != CameraStatusCode.Ok)
  return cameraStatus;

cameraStatus = _sensor.SetParameter(_sensorId, SensorParameter.ReorderingDeviation,
5000);
if (cameraStatus != CameraStatusCode.Ok)
   return cameraStatus;
```

3.9 External Trigger Configurations

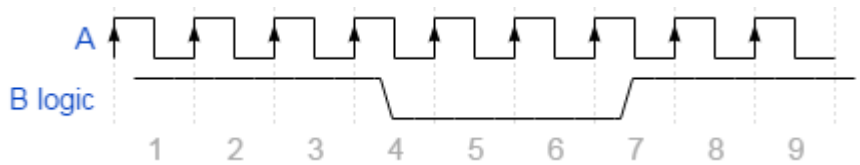
The trigger inputs can be used to synchronize the sensor with an external device. Application examples include: synchronizing the profile measurement into a certain signal of a production process or combining profiles into 3D point clouds based on the trigger index.

To measure the profile at each trigger event, input A should be used (input B is unconnected and is pulled down internally). This way every rising edge of the A signal triggers a measurement. The figure below shows the timing diagram of A and B signals when connected to a quadrature encoder. Only the B forwards triggers the measurement, because the B channel is on low state at the moment of a rising edge of the A channel.

Timing diagram of a quadrature encoder:



External logic B enables trigger A at edges 5-7: Another way of configuring the encoder inputs is shown in the figure below. B input of the encoder can be used as a Â“disableÂ” signal. On every rising edge of the encoder input signal A, the edge is accepted as a trigger if the encoder input signal B is simultaneously at a low state. An external logic can be used to generate B signal. B signal can be used, for example, to disable measurement during a relocation of the sensor.



Any of inputs 1-4 can be configured as A, B and Zero input. Input state is also transferred to PC embedded in each peak frame and can be used as general-

purpose input that way. The sensor keeps count of the trigger location internally and embeds this information for each frame. Every rising edge of the A input, while the B input is in low state, increases location count. Every rising edge of the A input, while the B input is in high state, decreases location count. A rising edge of the Zero input.

FS API default input source definitions:

PARAM_TRIGGER_SOURCE default is 1.

PARAM_TRIGGER_DISABLE_SOURCE default is 2.

PARAM_TRIGGER_ZERO_SOURCE default is 3.

PARAM_REG_PULSE_DIVIDER default is 1.

C++:

```
// defines input 2 used as triggering source
status = static_cast<CameraStatus>( SetIntParameter(sensor_ids[0], REG_TRIGGER_SOURCE,
2));
if (status != CameraStatus::CAMERA_OK)
{
    printf("FSAPI error: %d\n", status);
    return 1;
}
// sets triggering disable source as unused -> 0
status = static_cast<CameraStatus>( SetIntParameter(*sensor_ids[0],
REG_TRIGGER_DISABLE_SOURCE, 0));
if (status != CameraStatus::CAMERA_OK)
{
    printf("FSAPI error: %d\n", status);
    return 1;
}

// Every twentieth pulse triggers sensors
status = static_cast<CameraStatus>( SetIntParameter(*sensor_ids[0],
PARAM_REG_PULSE_DIVIDER, 20));
if (status != CameraStatus::CAMERA_OK)
{
    printf("FSAPI error: %d\n", status);
    return 1;
}
```

C#:

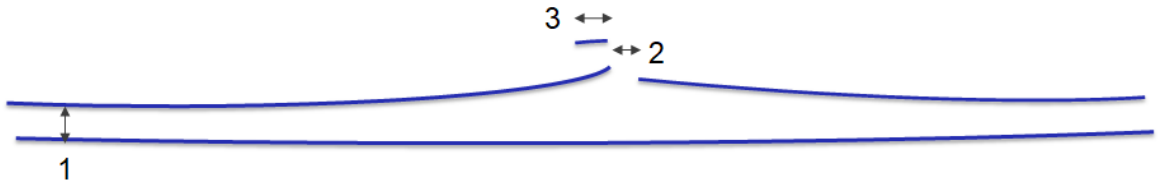
```
cameraStatus = _sensor.SetParameter(_sensorId, SensorParameter.TriggerSource, 1);
if (cameraStatus != CameraStatusCode.Ok)
  return cameraStatus;

cameraStatus = _sensor.SetParameter(_sensorId, SensorParameter.TriggerDisableSource,
2);
if (cameraStatus != CameraStatusCode.Ok)
 return cameraStatus;
```

```
cameraStatus = _sensor.SetParameter(_sensorId, SensorParameter.PulseDivider, 10);
if (cameraStatus != CameraStatusCode.Ok)
   return cameraStatus;
```

3.10 Detect Missing First Layer

Points in transparent surfaces are sorted to different layers according to z-coordinate. Problem is that if the first layer point is missing second layer point is detected as a first layer point. This feature improves classification by using prior information about the surface.

Missing first layer is detected by setting following parameters:

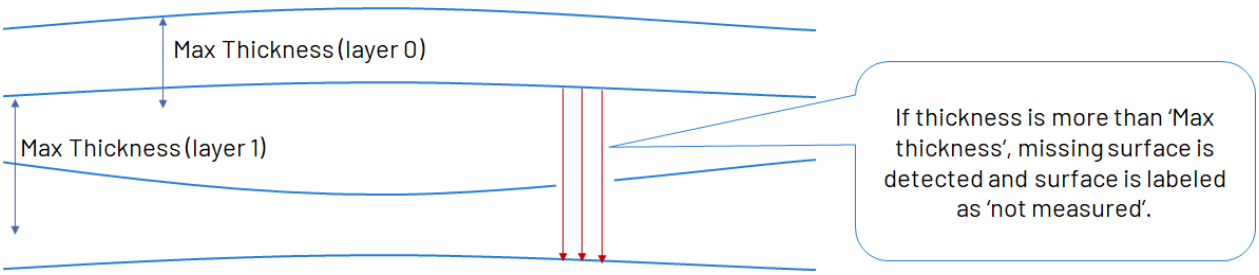


- PARAM_DETECT_MISSING_FIRST_LAYER Defines minimum height distance between two layers in micrometers. Typical value is 30 -100 um.
- PARAM_DETECT_MISSING_FIRST_LAYER Defines maximum x-coordinate distance between two points in micrometers. Typical value is 100 um.
- PARAM_DETECT_MISSING_FIRST_LAYER_MIN_LENGTH Defines minimum length in micrometers for surface which is detected as the first layer. This is useful to filter small defects (like dust) from the first layer classification. Value can be relatively long i.e. 1000 um for flat surfaces.

3.11 Thickness Filtering

FSSDK provides possibility to use a priori information to enhance missing layer detection.

Maximum layer thickness (PARAM_LAYER_MAX_THICKNESS) can be set for each layer



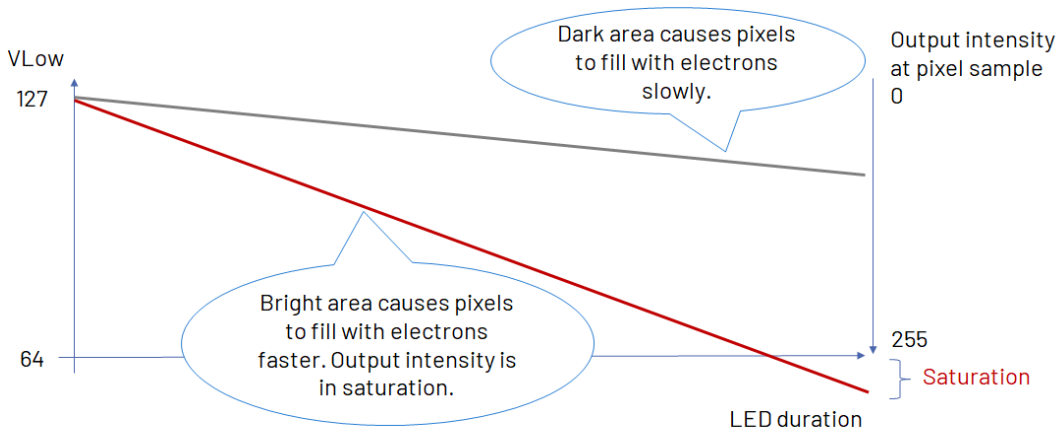
3.12 High Dynamic Range (HDR)

HDR imaging is supported by the sensor and it might be useful in situations where the target has high level of contrast.

If HDR is not used in high contrast cases, exposuring dark areas with a proper LED length saturates bright areas. This might cause reduced accuracy of detected peaks.

If HDR is used, no bright pixels are saturating and the relative order of brightness is maintained between all pixels. This improves accuracy of detected peaks.

Pixel intensity without HDR:

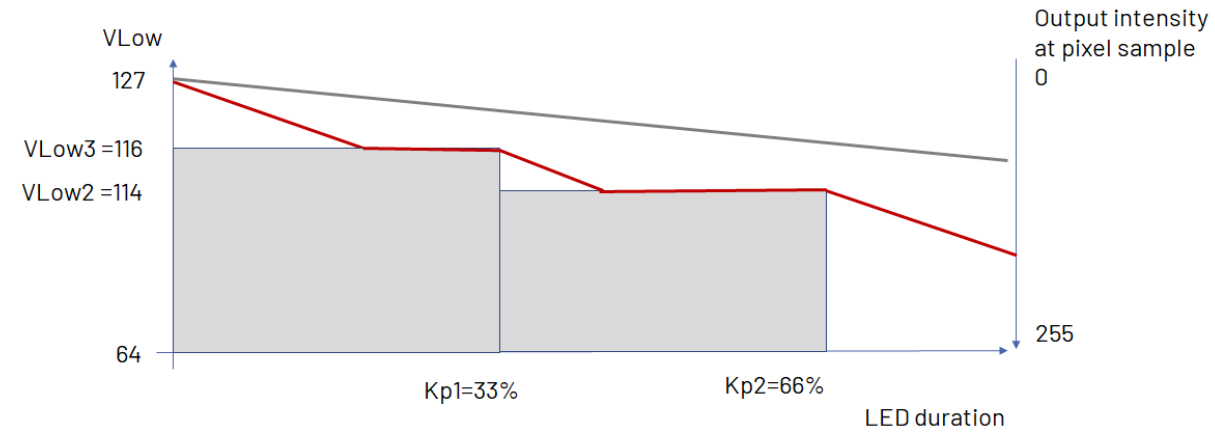


The slopes of the lines indicate the brightness of the pixels.

3.12.1 HDR with Sensors LCI401, LCI1200, LCI1201 and LCI1600

If HDR is enabled sensor sensitivity is dropped approximately 20%. Only values below are supported. Behavior of other values is not tested.

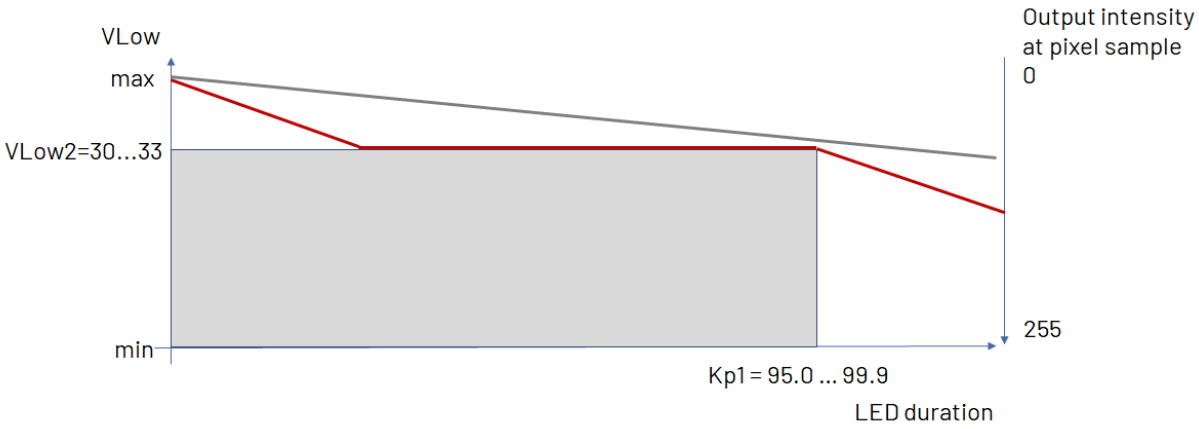
- Kp1=33
- Kp2=66
- VLow2=114
- VLow3=116



The slopes of the lines indicate the brightness of the pixels.

3.12.2 HDR with Sensors LCI1220 and LCI1620

- Only the first knee point is supported
- Float versions of HDR parameters are used
- PARAM_HDR_KP1_POS_FLOAT, range 95.00 - 99.99 is supported
- PARAM_HDR_VLOW2_FLOAT , range 30.0 - 33.0 is supported
- Maximum imaging frequency is different than without HDR. Use _AdjustRoiAndFps() with GetFpsForRoi attribute to read actual maximum frequency.



The slopes of the lines indicate the brightness of the pixels.

3.13 Interpolating Missing Measurement Points

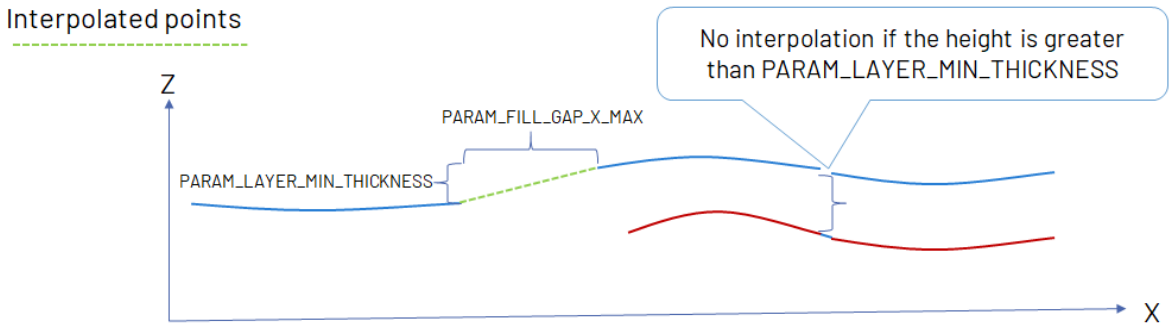
This functionality fills areas which are not measured with interpolated points.

PARAM_FILL_GAP_X_MAX Defines maximum gap of missing measurement points filled by interpolation. If 0, interpolation feature is disabled.

Additionally a user can configure z difference of the gap area:

PARAM_LAYER_MIN_THICKNESS parameter defines maximum Z difference for interpolated points. For opaque materials, a user can set parameter for the first layer.

If PARAM_LAYER_MIN_THICKNESS is 0 (default behavior), the missing area is interpolated except areas where the gap is less than 4 pixels wide. In those cases interpolation is not used when the Z difference is big.



3.14 Trim Edges Filter

Trim Edges filter is indented to remove artificial points detected at the end of the surfaces. The filter can be disabled/enabled at runtime in the line callback function and it takes effect on the next processed profiles

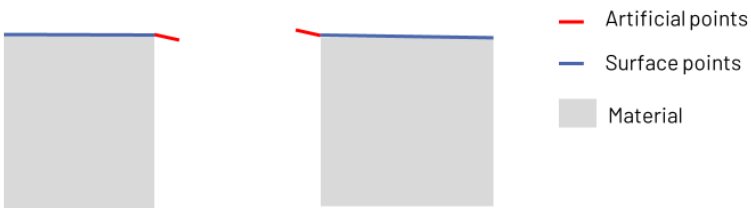
The filter works optimally for those edges which are clearly visible. It may remove real points if edges are gradually disappearing. Therefore, it is recommended to use the filter only for profiles which have clear edges. Additionally, it might be useful to use PARAM_FILL_GAP_X_MAX parameter to fill not measured areas on the edge.

Filtering is done for each profile separately which means that only horizontal edges are trimmed.

Following issues are handled with the filter:

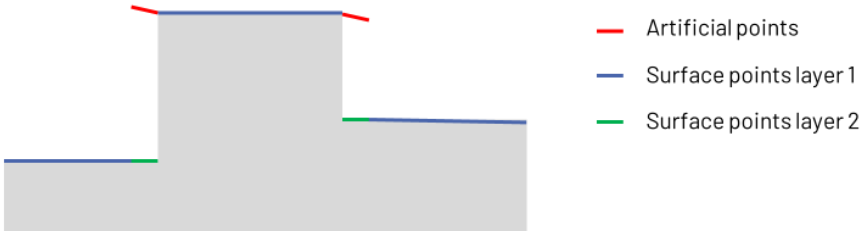
ISSUE#1, MISSING POINTS AT PROFILE

Due to the point spread function (PSF) top surface is continued after real edge. This effect would produce artificial points to the profile.



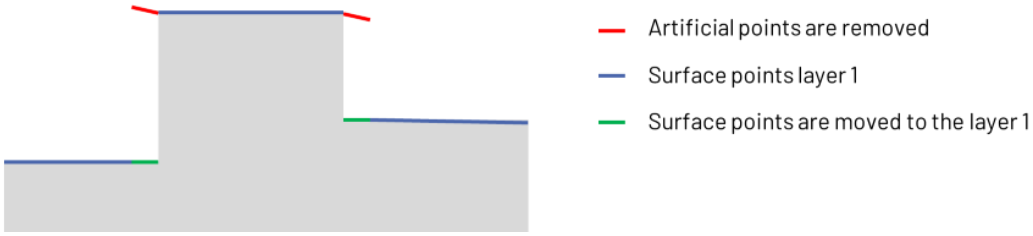
ISSUE#2, SHARP EDGE AT Z-DIRECTION

Due to the point spread function (PSF) top surface is continued after real edge. This effect would produce artificial points to the higher step and missing points at the lower step.



ALGORITHM

The filter removes tails of profiles if no neighboring 3D points are not found. Removing algorithm uses intensity values of 3D points.
Layer reordering is done if points are removed.
The filter can be enabled with PARAM_TRIM_EDGES parameter.



3.15 Recipes

Loading a recipe is the recommended way to configure the sensor at application startup and during the parameter set change. For online changes, user shall use the SetParameter/GetParameter functions.

A typical use case for the recipe is following (this is how FSSDK GuiExample works):

- The application initializes the sensor
- The application loads a recipe and starts acquisition
- The application makes changes to the camera parameters with SetParameter functions
- The application saves the recipe

another use case the application only uses the recipe (this is how FSSDK ConsoleExample works):

- The application initializes the sensor
- The application loads the recipe and starts acquisition

steps are needed for the sensor initialization with a recipe.

1) Open()-function to discover sensors and Connect()-function to set IP addresses

```
int camera_count = 1;
char **camera_ids = nullptr;
_Open(&camera_count, &camera_ids, 5000);
_Connect(camera_ids[0], 0);
```

2) Check is the calibration available in the sensor.

```
int calibrations_in_camera = 0;
GetIntParameter(camera_ids[0], const_cast<char*>(PARAM_SENSOR_DATA_IN_FLASH),
&calibrations_in_camera);
// if calibrations not found from camera, needs to be defined.
if(!calibrations_in_camera)
{
    printf("Calibration files not in camera, must be provided by user\n");
    status = static_cast<CameraStatus>(_SetStringParameter(camera_ids[0],
const_cast<char*>(PARAM_SENSOR_CALIBRATION_FILE), z_calib_file));

    if (status != CameraStatus::CAMERA_OK && !(status ==
CAMERA_ERROR_CALIB_FILE_ATTRIBUTE_NOT_FOUND || status ==
CAMERA_ERROR_CALIB_FILE_ATTRIBUTE_INVALID))
    {
        printf("FSAPI error: %d. Failed to set Z calibration file.\n", status);
        return 1;
    }
    status = static_cast<CameraStatus>(_SetStringParameter(camera_ids[0],
const_cast<char*>(PARAM_SENSOR_X_CALIBRATION_FILE), x_calib_file));
```

```
    if (status != CameraStatus::CAMERA_OK && !(status ==
CAMERA_ERROR_CALIB_FILE_ATTRIBUTE_NOT_FOUND || status ==
CAMERA_ERROR_CALIB_FILE_ATTRIBUTE_INVALID))
    {
        printf("FSAPI error: %d Failed to set X calibration file.\n", status);
        return 1;
}
    }
```

3) Load a recipe

```
status = static_cast<CameraStatus>(_SetStringParameter(camera_ids[0],
const_cast<char*>(PARAM_LOAD_RECIPE), recipeName));
if (status != CameraStatus::CAMERA_OK)
{
    printf("Recipe error");
    return;
}
```

4) Set image acquisition parameters which are not defined in the recipe. For example:

```
_SetIntParameter(camera_ids[0], PARAM_THICKNESS_MODE, 0);
_SetLineSortingOrder(camera_ids[0],
SurfaceSorting::FROM_MAX_INTENSITY_TO_LOWER_AND_TOP_TO_BOTTOM);
                _SetLineCallback(camera_ids[0], 0, LineCallbackHandlerLayer1);
                _SetLineCallback(camera_ids[0], 1, LineCallbackHandlerLayer2);
                _SetLineCallback(camera_ids[0], 2, LineCallbackHandlerLayer3);

_StartGrabbing(camera_ids[0]);
```

3.16 Z-Compensation

3.16.1 Calibration

A strong surface texture may cause edge artifacts in Z values. In order to reduce edge artifacts, the sensor can be calibrated for Z-Compensation.

An example of calibration procedure is implemented in FieldCalibrationTool. The tool can be used as is or it can be customized by using provided source codes.

3.16.2 Calculation

Z-Compensation can be enabled separately for X or Y direction. Y direction means moving direction acquisition with external triggering. Typically, both X and Y direction are needed for compensation calculation.

Note that Z-Compensation in Y direction is available only for BatchCallback data and there is no effect on LineCallback data. Additionally PARAM_MOVING_DIRECTION and PARAM_ENCODER_PULSE_WIDTH parameters need to be set properly when Y direction compensation is used.

A calibration file is read when the user enables either compensation direction. By default, calibration file location is the same as other calibration files (C:\FocalSpec\Calibration). If calibration used this path there is no need to set the file manually. Optionally users can set full path with paramter PARAM_SENSOR_Z_COMPENSATION_FILE .

In console example application X and Y compensations are enabled for batch call back function:

```
status = static_cast<CameraStatus>( SetIntParameter(camera_ids[0],
const_cast<char*>(PARAM_Z_COMPENSATION_X_DIRECTION), 1));
if (status == CAMERA_OK)
    printf("Z Compensation X direction enabled\n");

status = static_cast<CameraStatus>( SetIntParameter(camera_ids[0],
const_cast<char*>(PARAM_Z_COMPENSATION_Y_DIRECTION), 1));
if (status == CAMERA_OK)
    printf("Z Compensation Y direction enabled\n");
```

For some materials it might be beneficial to adjust Z-compensation algorithm. User can increase or decrease Z-compensation effect by following multipliers: PARAM_Z_COMPENSATION_X_BRIGHT_TO_DARK_MUL, PARAM_Z_COMPENSATION_X_DARK_TO_BRIGHT_MUL, PARAM_Z_COMPENSATION_Y_BRIGHT_TO_DARK_MUL, PARAM_Z_COMPENSATION_Y_DARK_TO_BRIGHT_MUL.

User has an option remove points where calculated Z-Compensation is big. This is useful feature if the material has a strong texture and compensation is suboptimal for those points. The feature can be used with parameter PARAM_Z_COMPENSATION_THRESHOLD.

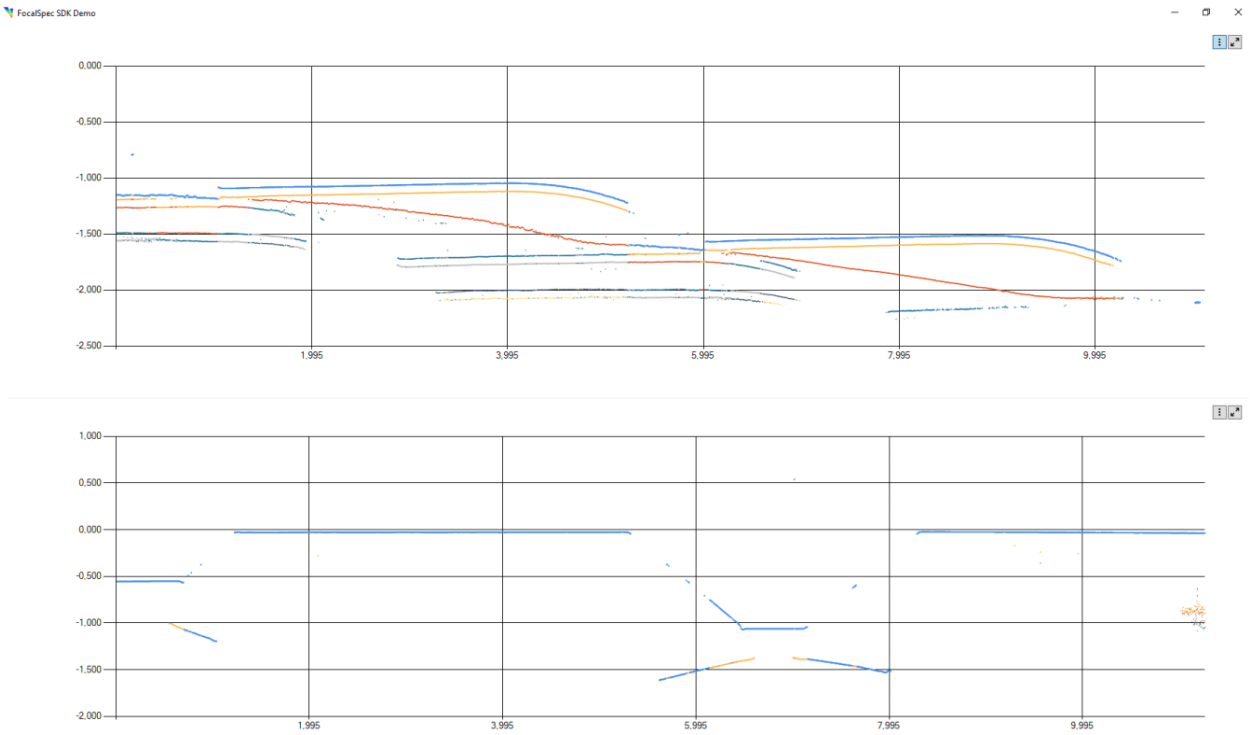
If points are removed by PARAM_Z_COMPENSATION_THRESHOLD feature, empty areas can be interpolated by using neighboring points. Interpolation can be enabled with parameter PARAM_Z_COMPENSATION_INTERPOLATE.

4 FocalSpec SDK Demo Example Application

4.1 Overview

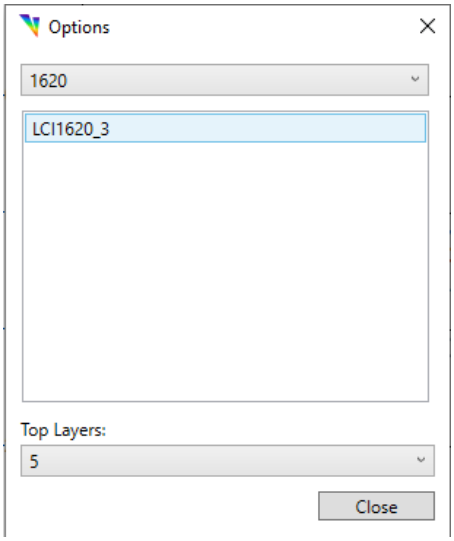
FocalSpec SDK Demo is a simplified example for SDK usage.

The demo application demonstrates support for multiple sensors. User interface displays maximum of 4 sensors. The picture below shows an example with 2 sensors.



The demo application is based on recipe usage. The recipes must be saved first using the GuiExample application.

When the demo application starts, it connects to all available sensors. The recipe selection for each sensor must be done from the options (top-right corner of the sensor display).



5 FocalSpec Graphical User Interface Example Application

5.1 Overview

The pictures below explain the usage of the example application.

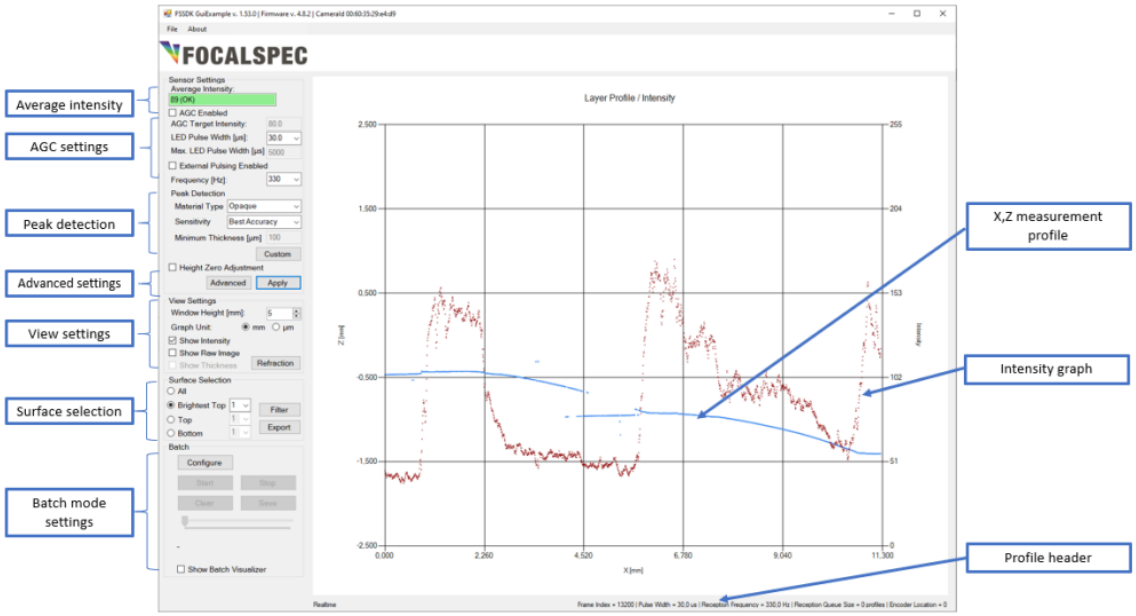


Figure 1. Example Application explained.

5.1.1 Menus

- **File - Calibration Setup** - Opens a setup window where user can change calibration files.
- **File - Load Recipe** - Opens a window where user select available recipes.
- **File - Save Recipe** - Saves currently used parameters to a recipe file. Asks a recipe name on the first time.
- **File - Save Recipe As** - Saves currently used parameters to a different recipe file.
- **About** - Displays software information.

5.1.2 Recipes

The camera parameters, which are currently used in GuiExample, can be saved via menus to a named recipe file. The recipe files are sensor type specific. When the GuiExample application starts, it asks user to select a recipe. GuiExample modifies the camera parameters and the user interface based on the recipe settings.

5.1.3 Zooming

To zoom into an area of interest, use the mouse to select a rectangle, which covers the area. The picture below demonstrates zoom in.

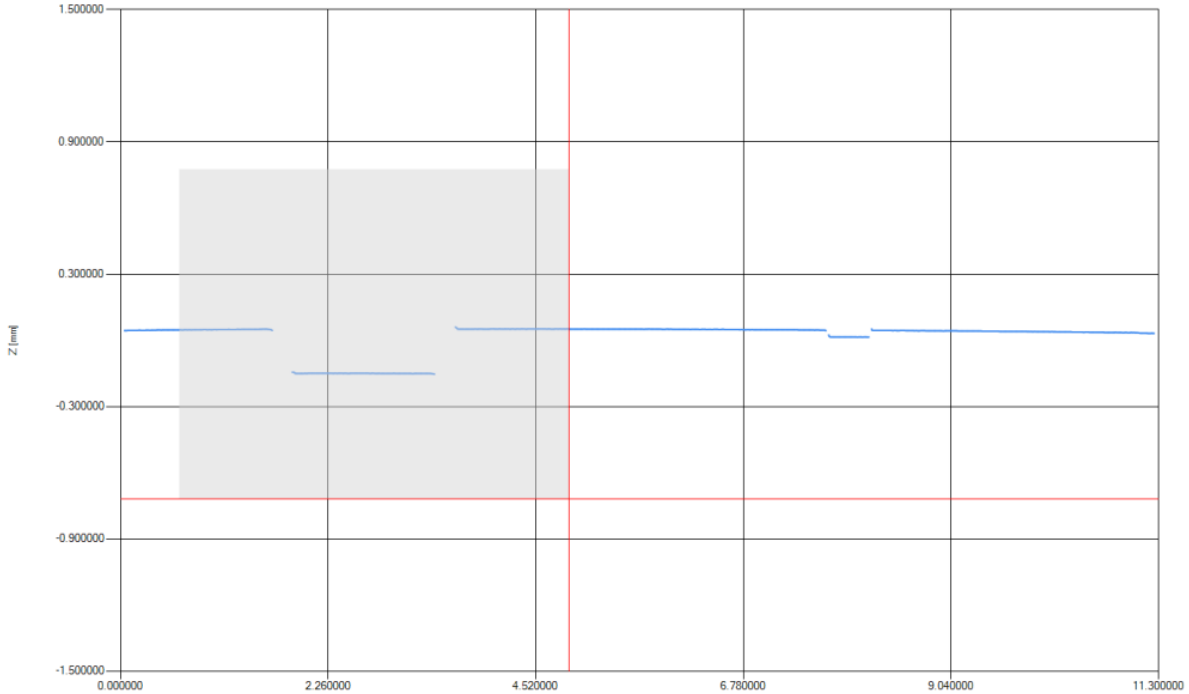


Figure 2. Zoom into an area of interest.

To zoom out, click the buttons on the axes as depicted below.

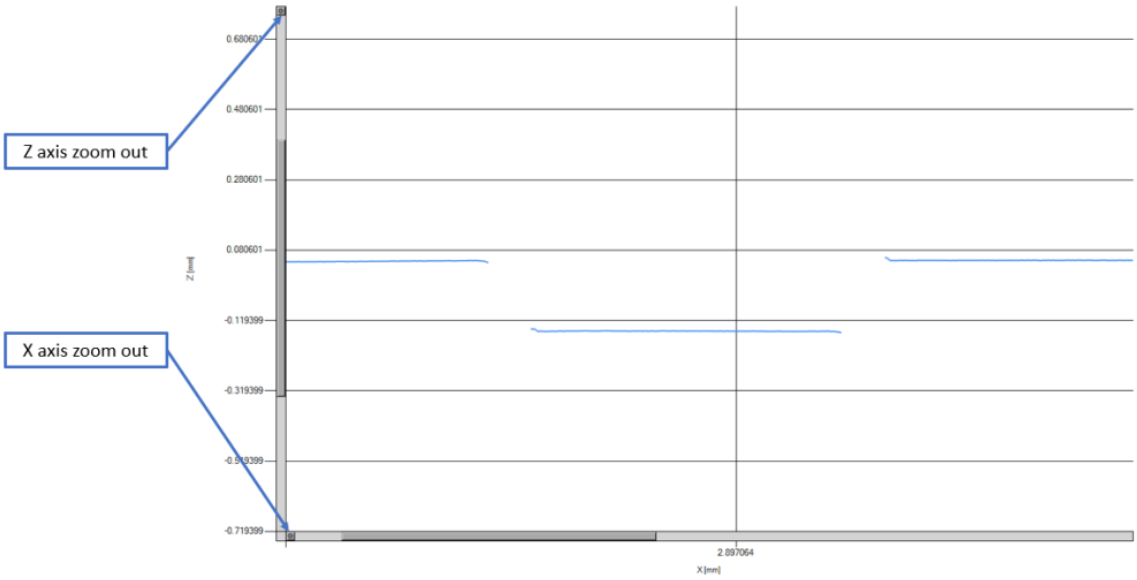


Figure 3. Zoom out.

5.2 Sensor Settings

5.2.1 Sensor attributes

- **Average Intensity** - Average intensity of the profile (read-only).
- **External Pulsing Enabled** - External triggering input selection. Disabled in batch mode.
- **Frequency [Hz]** - Sampling frequency the sensor is using. Disabled in batch mode.
- **Height Zero Adjustment** - Centers image around calibrated center. Can be useful on higher frequencies.

Click **Apply** to make (sensor) settings effective.

5.2.2 Automatic Gain Control

The application provides an example of how to use Automatic Gain Control (AGC) in measurement applications. When AGC is supported by the sensor, *"AGC Enabled"* checkbox is adjustable. When AGC is not supported by the sensor, *"AGC Enabled"* checkbox, *"AGC Target Intensity"* value field and *"Max. LED Pulse Width [µs]"* value field are disabled.

In the example application, the required parameters for the AGC are *"AGC Target Intensity"* and *"Max. LED Pulse Width [µs]"*, which is the upper limit. *"LED Pulse Width [µs]"* is the starting reference. The AGC will automatically adjust *"LED Pulse Width [µs]"* to achieve optimal intensity for the profile. When AGC is adjusting, the *"Average Intensity"* value is changing until optimal intensity is achieved or *"LED Pulse Width [µs]"* is equal to *"Max. LED Pulse Width [µs]"*.

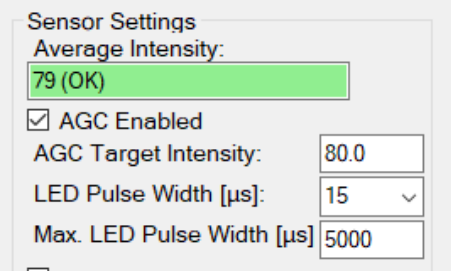


Figure 4. LED pulse width automatically optimized by the AGC.

5.2.3 Peak detection

Peak detection enables an easy selection from pre-selected sensor attributes by changing *"Material Type"* and light *"Sensitivity"*. For transparent materials one must set minimum thickness of the layer (µm).

Click **Custom** to open customization settings for peak detection. Following settings are available:

- **Peak Threshold** - Filter for low intensity noise.
- **Fir Length** - Length of the Finite Impulse Response (FIR) filter.PARAM_PEAK_FIR_LENGTH
- **Averaging Fir Length** - Length of the Averaging FIR filter. Available, if detection filter is not supported by the firmware.
- **Detection Filter** - Detection filter length PARAM_SIGNAL_DETECTION_FILTER_LENGTH. Available, if it is supported by the firmware.
- **Average Intensity Filter** - Average intensity filter length PARAM_PEAK_AVERAGE_INTENSITY_FILTER_LENGTH. Available, if it is supported by the firmware.

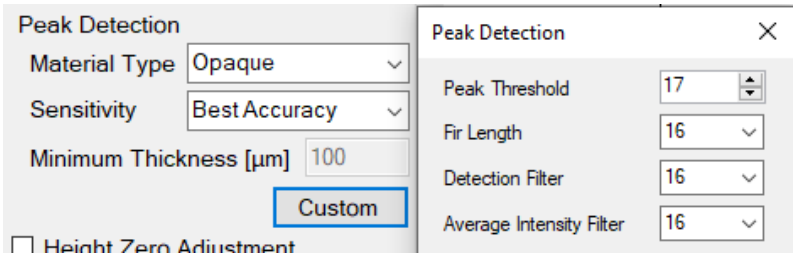


Figure 5. Peak detection settings.

5.2.4 Advanced settings

If HDR (High Dynamic Range) is supported by the firmware, HDR parameters Vlow2, VLow3, Kp1Pos and Kp2Pos can be modified to enhance the image quality. VLow3 and Kp2Pos are not available for LCI1220 and LCI1620 sensors.

If Layer specific parameters are supported by the firmware, *"Max Thickness"* and *"Intensity Type"* can be set for each layer separately.

5.3 View Settings

- **Window Height [mm]** - Scales graph height.
- **Graph Unit** - mm or µm.
- **Show Intensity** - Displays intensity diagram of the received data. The value scale for intensity is displayed on the right graph axis. The intensity selection is disabled in batch mode and when the raw image is displayed.
- **Show Raw Image** - Displays raw image from the camera. Disabled in batch mode.
- **Show Thickness** - Displays thickness profile graph for each layer. Available for *"Brightest Top"*, *"Top"* and *"Bottom"* surface selections. If thickness profile is selected, upper graph displays the height of the first layer. Thickness selection is disabled in batch mode.

Click **Refraction** to modify effective refractive index for each layer.

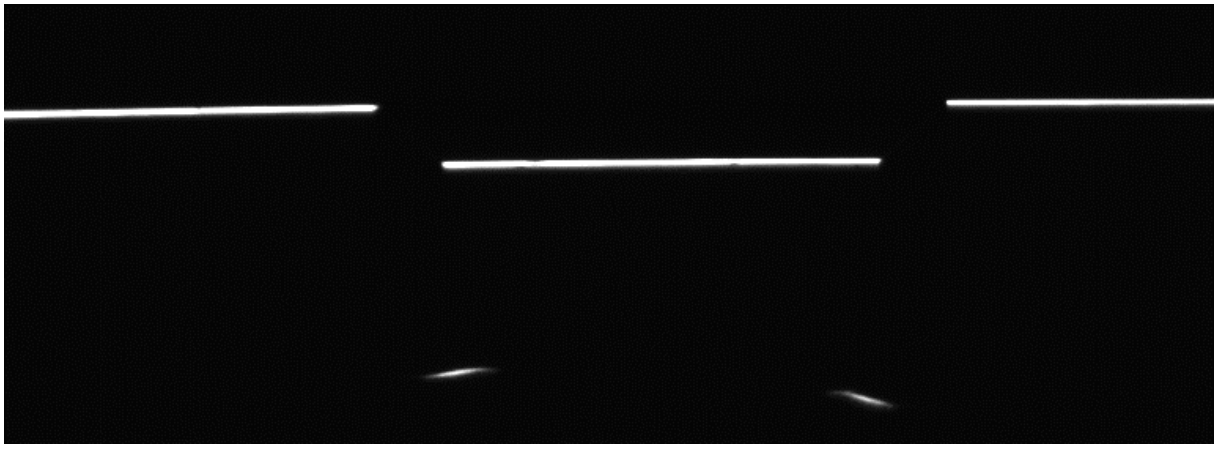


Figure 6. Raw image from camera.

5.4 Surface Selection

The profile graph can display several layers from transparent surfaces. User may select which surface to explore.

- **All** - All surfaces are displayed.
- **Brightest Top** - First points are sorted according to intensity values and then according to z-coordinates from the top to the bottom. Method works well if number of layers is known.
- **Top** - Sorting of displayed surface starting from the topmost.
- **Bottom** - Sorting of displayed surface starting from the bottommost.

Export - Data from the selected layers is exported to a CSV file. If raw image is displayed, the image is exported to a PNG file.

5.5 Batch Mode

Batch mode provides a way to generate 3D point cloud data from a sample on the moving measurement table or assembly line. Sensor sampling frequency is

controlled either internally, i.e. triggered by application itself, or externally, pulsing generated by a moving step motor, for example.

5.5.1 Batch controls

- **Configure** - Opens *"Batch Configuration"* window.
- **Start** - Sets batch in running mode. On external triggering sensor is waiting for pulses from control. On internal triggering batch starts running immediately.
- **Stop** - Allows user abort a running batch, which is not yet completed.
- **Clear** - Clears batch data from buffer.
- **Save** - Saves batch data result either as 3D point cloud data or in 2D bitmap format.

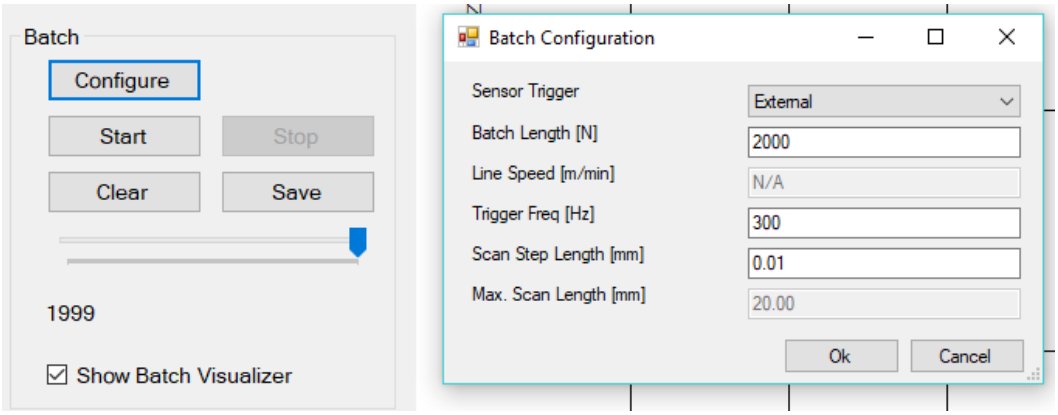


Figure 7. Batch controls and configuration settings.

5.5.2 Batch Configuration settings

- **Sensor Trigger** - Select either external or internal triggering.
- **Batch Length** - Number of lines acquired.
- **Line Speed [m/min]** - Moving speed of measured sample. Not applicable for external trigger.
- **Trigger Freq [Hz]** - Internal trigger defines sampling frequency. External trigger, insert expected average frequency. Information is used to adjust depth of field if frequencies > 300 Hz are used.
- **Scan Step Length [mm]** - Step between each acquired line profile. Not applicable for internal trigger.
- **Max. Scan Length [mm]** - Total scan length of completed batch, which is the result of line count times scan step length.

When the batch is started, the program changes to the recording mode and the real-time mode is stopped. When the recording is either stopped or finished, the program returns to batch mode. By clicking **Clear** program returns to the real-time mode.

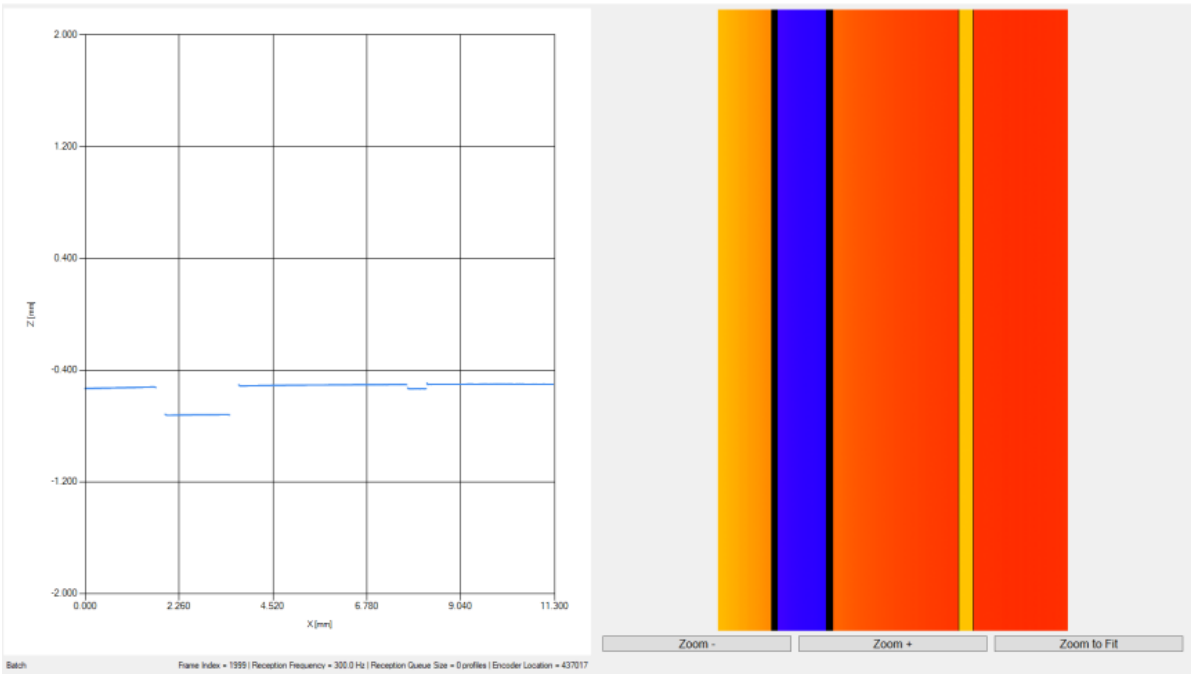


Figure 8. Batch Visualizer view after a completed batch run.

In completed batch job view, select **Save** to store measurement data results for further analysis. Supported file formats are

- ASC - ASCII text file format. 3D point cloud data. X, Y and Z coordinates separated by space.
- PCD - Binary 3D point cloud data format. Contains X, Y and Z coordinates and intensity value for each measurement point.
- BMP - Bitmap, 2D image file format.

6 SDK Troubleshooting

This page describes how to enable logging with FSAPI DLL.

6.1 Logging

Please copy following configuration file to the same folder where the application is started.

Log files will be generated to the log subfolder. Note that the program shall have write access to the folder. Please change FILENAME if the program is located in the protected directory.

File name: vevolog.conf

```
GLOBAL:       ## Generic level that represents all the levels. Useful when setting
global configuration for all levels.
```

```
  FORMAT                              =  "[%datetime] [%level] [%fbase] %msg"
  FILENAME                            =  "logs\vevo_%datetime{%Y%M%d}.log"
  ENABLED                             =  true
  TO_FILE                             =  true
  TO_STANDARD_OUTPUT      =  false
  MILLISECONDS_WIDTH      =  6
  PERFORMANCE_TRACKING    =  false
  MAX_LOG_FILE_SIZE       =  10485760     ## 10 MB
  LOG_FLUSH_THRESHOLD     =  1            ## Flush after every N logs
  KEEP_LOGFILES_DAYS      =  0            ## Log files older than value in days are
deleted automatically. Auto deleting is disabled when set to '0'.

TRACE:       ## Information that can be useful to back-trace certain events - mostly
useful than debug logs.
  ENABLED                             =  true
  TO_FILE                             =  true

DEBUG:       ## Informational events most useful for developers to debug application.
  ENABLED                             =  false
  TO_FILE                             =  false

INFO:        ## Useful to represent additional information about current
configuration.
  ENABLED                             =  true
  TO FILE                             =  true

FATAL:       ## Information representing errors in application but application will
keep running.
  ENABLED                             =  true
  TO_FILE                             =  true

ERROR:       ## Useful when application has potentially harmful situations.
  ENABLED                             =  true
  TO_FILE                             =  true

WARNING:     ## Information that can be highly useful and vary with verbose logging
level.
  ENABLED                             =  true
  TO_FILE                             =  true

VERBOSE:     ## Information that can be highly useful and vary with verbose logging
level.
  ENABLED                             =  false
  TO_FILE                             =  false
```

Above configuration enables all logging but DEBUG level. For troubleshooting you can enable debug logs by following settings:

```
DEBUG:       ## Informational events most useful for developers to debug application.
  ENABLED                             =  true
```

```
TO_FILE                         =  true
```

For advanced configuration more details are found from https://github.com/zuhd-org/easyloggingpp.

7 Module Index

7.1 Functions and Parameters

Here is a list of all modules:

8 Class Index

8.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

9 Module Documentation

9.1 Callback definitions

9.1.1 Typedefs

- typedef void(__cdecl * RawImageCallback)(char *cameraID, unsigned char *image, int *bytes, HEADER *header)
- typedef void(__cdecl * ProfileCallback)(char *cameraID, STRUCT_POINT *points, int *point_count, HEADER *header)
- typedef void(__cdecl * LineCallback)(char *cameraID, int layerId, float *zValues, float *intensityValues, int line_length, double xStep, HEADER *header)
- typedef void(__cdecl * BatchCallback)(char *cameraID, int layerId, float *zValues, float *intensityValues, int line_length, int batch_length, double xStep, HEADER *lineHeaders)

9.1.2 Detailed Description

9.1.3 Typedef Documentation

typedef void(__cdecl * BatchCallback)(char *cameraID, int layerId, float *zValues, float *intensityValues, int line_length, int batch_length, double xStep, HEADER *lineHeaders)

```
C# function:  public delegate void BatchCallback(int layerId,float[]
zValues,float[] intensityValues,int lineLength,int batchLength,double xStep,Header
lineHeaders);
```

Callback signature for batch results per configured layer.

Parameters

cameraID	[in] Identifier for the camera.
layerId	[in] Layer identifier (0 is the first layer).
intensityValues	[in] Array of intensity values.
line_length	[in] Length of the array.
batch_length	[in] Total count of line profiles in the array.
xStep	[in] X resolution in micrometers.
lineHeaders	[in] Collected header information of each line profile belonging for the batch run.

typedef void(__cdecl * LineCallback)(char *cameraID, int layerId, float *zValues, float *intensityValues, int line_length, double xStep, HEADER *header)

```
C# function:  delegate void LineCallback(int layerId,float[] zValues,float[]
intensityValues,int lineLength,double xStep,Header header);
```

Callback signature for line reception per configured layer.

Parameters

cameraID	[in] Identifier for the camera.
layerId	[in] Layer identifier (0 is the first layer).
zValues	[in] Array of heights.
intensityValues	[in] Array of intensity values.
line_length	[in] Length of the array.
xStep	[in] X resolution in micrometers.
header	[in] Additional information related to the line.

typedef void(__cdecl * ProfileCallback)(char *cameraID, STRUCT_POINT *points, int *point_count, HEADER *header)

```
C# function:  delegate void ProfileCallback(IList<Point> profile, Header header);
```

Callback signature for profile reception.

Parameters

cameraID	[in] Identifier for the camera.
points	[in] Pointer to the received profile data.
point_count	[in] Number of points in the profile.
header	[in] Additional information related to the profile.

typedef void(__cdecl * RawImageCallback)(char *cameraID, unsigned char *image, int *bytes, HEADER *header)

```
C# function:  delegate void RawImageCallback(byte[] image, int bytes, Header
header);
```

Callback signature for raw image reception.

Parameters

cameraID	[in] Identifier of the camera.
image	[in] Pointer to the received 8-bit grayscale image. Each pixel is represented by a 8-bit element. The array is composed so that first W elements are the top row, second W elements are next row etc. W is sensor image height that can be found from the sensor data sheet.
bytes	[in] Number of bytes in the image.
header	[in] Additional information related to the raw image.

9.2 Functions

9.2.1 Functions

- void _Close()
- int _GetSerialNumber(char *cameraId, char *serialNumber)
- int _GetDeviceVersion(char *cameraId, char *deviceVersion)
- int _Open(int *cameraCount, char ***cameraIds, int timeout)
- int _Discover(int *cameraCount, char ***cameraIds, int timeout)
- int _Connect(char *cameraId, char *ipAddress)
- int _StartGrabbing(char *cameraID)
- int _StopGrabbing(char *cameraID)
- int _SetRawImageCallback(char *cameraID, RawImageCallback callback)
- int _SetProfileCallback(char *cameraID, ProfileCallback callback)
- int _SetLineCallback(char *cameraID, int layerId, LineCallback callback)
- int _SetBatchCallback(char *cameraID, int layerId, BatchCallback callback)
- int _SetLineSortingOrder(char *cameraID, int sortingOrder)
- int _IsConnected(char *cameraID)
- int _SetStringParameter(char *cameraID, char *paramName, char *value)
- int _SetIntParameter(char *cameraID, char *paramName, int value)
- int _SetLayerIntParameter(char *cameraID, int layerId, char *paramName, int value)
- int _SetLayerFloatParameter(char *cameraID, int layerId, char *paramName, float value)
- int _GetLayerIntParameter(char *cameraID, int layerId, char *paramName, int *value)
- int _GetLayerFloatParameter(char *cameraID, int layerId, char *paramName, float *value)
- int _SetFloatParameter(char *cameraID, char *paramName, float value)
- int _SetDoubleParameter(char *cameraID, char *paramName, double value)
- int _GetStringParameter(char *cameraID, char *paramName, char *value)
- int _GetIntParameter(char *cameraID, char *paramName, int *value)
- int _GetFloatParameter(char *cameraID, char *paramName, float *value)
- int _GetDoubleParameter(char *cameraID, char *paramName, double *value)
- int _FieldCalibrationStart(char *cameraID, char *xCalibFile, char *zCalibFile, double *grooveHeightsExpected, int numOfGrooveHeightsExpected, FieldCalibrationResults *results)
- int _IntensityCalibrationStart(char *cameraID, char *xCalibrationFile, char *zCalibrationFile, char *intensityCalibrationFile, bool smoothHorizontal, double gain)
- double _IntensityCalibrationProgress(char *cameraID, char *progressBitmapFile)
- void _IntensityCalibrationStop(char *cameraID, char *correctedBitmapFile)
- bool _IntensityCalibrationSave(char *cameraId, char *outputFile)
- int _FieldCalibrationSave(char *cameraId, char *fileName)

- int _DynamicSensorCtrlParameters (char *cameraID, int param_set_count, DynSensorControl *control_parameters)
- int _GetParameterSupport (char *cameraID, char *paramName, bool *is_supported)
- int _AdjustRoiAndFps (char *cameraID, const int operation, int const inValue, int *outValue)
- int _SetPeakDetectionParameters (char *cameraID, MaterialType type, DetectionSensitivity sensitivity)

9.2.2 Detailed Description

9.2.3 Function Documentation

int _AdjustRoiAndFps (char * *cameraID*, const int *operation*, int const *inValue*, int * *outValue*)

```
C# function:   CameraStatusCode AdjustRoiAndFps(string cameraId, int roiOperation,
int inValue, out int outValue)
```

Adjust imaging height, offset or frequency according to operation enumeration.

Parameters

cameraID	[in] Identifier of the camera.
operation	[in] ROI and frequency operation mode. SetRoiForFps = 0, Adjust imaging height and offset based on estimated maximum frequency.GetRoiForFps = 1, Returns imaging height based on estimated maximum frequency. Does not set any sensor parameters.SetFpsForRoi = 2, Adjust imaging height and offset based on desired z-range on micrometers.GetFpsForRoi = 3, Returns imaging frequency based on desired z-range on micrometers. Does not set any sensor parameters.GetRoiOffsetZero = 4, Returns imaging zero offset in pixels for desired image height in pixels. Does not set any sensor parameters.GetRoiOffsetCentered = 5, Returns imaging centered offset in pixels for desired image height in pixels. Does not set any sensor parameters.SetRoiForFpsCentered = 6, Adjust imaging height and offset based on estimated maximum frequency. Centers image around middle of sensor.GetFpsForHeight = 7, Returns imaging frequency based on imaging height on pixels. Does not set any sensor parameters.GetLedPulseForFps = 8, Returns max led pulse width time in microseconds based on desired frequency. Does not set any sensor parameters.

inValue	[in] SetRoiForFps, SetRoiForFpsCentered, GetExposureForFps and GetRoiForFps: frequency in Hz.SetFpsForRoi and GetFpsForRoi: z-range in micrometers.GetRoiOffsetZero, GetFpsForHeight and GetRoiOffsetCentered: image height in pixels.

outValue	[out] SetRoiForFps, SetRoiForFpsCentered and GetRoiForFps: image height in pixels.SetFpsForRoi, GetFpsForHeight and GetFpsForRoi: frequency in Hz.GetRoiOffsetZero and GetRoiOffsetCentered: image offset in pixels.GetLedPulseForFps: LED pulse width in microseconds..

Returns

CAMERA_OK or error code

void _Close ()

C# function: CameraStatusCode Close()

Close the API and release all resources. This function is usually called in the end of a client application life cycle.

int _Connect (char * *cameraId*, char * *ipAddress*)

C# function: CameraStatusCode Connect(string cameraId, string ipv4)

Connects to a camera making it ready for use.

Parameters

cameraId	[in] Identifier of the camera.
ipAddress	[in] IPv4 address to assign. This must be in the same address space as the client PC is. For example, if client PC's address is "10.10.40.1", this could be "10.10.40.2" provided that it is free. Set this NULL (0) if auto-ip should be used.

Returns

CAMERA_OK or error code as defined in CameraError.h.

int _Discover (int * *cameraCount*, char *** *cameraIds*, int *timeout*)

C# function: CameraStatusCode Discover(ref int cameraCount, out List<string> cameraIds, int timeout)

Allows rediscover cameras after API has already been loaded. If expected camera count is not reached within given timeout, an additional timeout is waited until all expected cameras are found.

Parameters

cameraCount	[in] Expected camera count as an input. [out] Returns count of connected cameras.
cameraIds	[out] Pointer to a C-string array that will be populated with IDs of the discovered cameras. API will allocate and deallocate memory.
timeout	[int] Timeout to wait for replies in milliseconds.

Returns

CAMERA_OK or error code as defined in CameraStatus.h. (See Return values for full list)

int _DynamicSensorCtrlParameters(char * *cameraID*, int *param_set_count*, DynSensorControl * *control_parameters*)

```
C# function:  CameraStatusCode DynamicSensorCtrlParameters(string cameraId, int
paramSetCount, DynSensorControl[] controlParameters)
```

Set group of dynamic sensor control parameters to register.

In case you are using a variable ROI size in dynamic sensor control parameters, set the initial ROI (the one before dynamic sensor control begins) to the largest size you have in the dynamic sensor control parameter list. For example, if you have ROI heights 100, 500, and 1000 in the list, set the initial ROI height to the highest value 1000. This must be done because the size of internal data buffers is determined by the initial ROI size. Therefore, setting the initial ROI size to the largest value will ensure that the data from the largest ROI will fit in the internal data buffers.

Parameters

cameraID	[in] Identifier of the camera.
param_set_count	[in] Number of control parameter structures.
control_parameters	[in] Group set of control parameter for sensor.

Returns

CAMERA_OK or error code

int _FieldCalibrationSave(char * *cameraId*, char * *fileName*)

```
C# function:  CameraStatusCode FieldCalibrationSave(string cameraId, string
fileName)
```

Save accepted field calibration results to the calibration file

Parameters

cameraId	[in] Identifier of the camera.
fileName	[in] Full path for the saved z-calibration file.

Returns

CAMERA_OK or error code

int _FieldCalibrationStart(char * *cameraID*, char * *xCalibFile*, char * *zCalibFile*, double * *grooveHeightsExpected*, int *numOfGrooveHeightsExpected*, FieldCalibrationResults * *results*)

```
C# function:  CameraStatusCode FieldCalibrationStart(string cameraId, string
xCalibFile, string zCalibFile, List<double> grooveHeightsExpected, ref
FieldCalibrationResults results)
```

Start calibration procedure. Type of the calibration block is detected and calculate calibration correction is written to the results. The output file is not saved yet. Function overrides existing camera parameters.

Parameters

cameraID	[in] Identifier of the camera.
xCalibFile	[in] Full path of the x-calibration.
zCalibFile	[in] Full path of the z-calibration.
grooveHeightsExpected	[in] Array of expected groove heights in um.
numOfGrooveHeightsExpected	[in] Number of values in array. 0 if default groove heights are used.
results	[out] Measured calibration. (FieldCalibrationResults)

Returns

CAMERA_OK or error code as defined in CameraStatus.h. (See Return values for full list)

int _GetDeviceVersion(char * *cameraId*, char * *deviceVersion*)

```
C# function:  CameraStatusCode GetDeviceVersion(string cameraId, out string
deviceVersion)
```

Get the device version of a camera.

Parameters

cameraId	[in] Identifier of the camera.
deviceVersion	[out] Device version read from the camera.

Returns

CAMERA_OK or error code as defined in CameraStatus.h. (See Return values for full list)

int _GetDoubleParameter(char * *cameraID*, char * *paramName*, double * *value*)

```
C# function:  CameraStatusCode GetParameter(string cameraId, string name, out
double value)
```

Get a double parameter from a camera.

Parameters

cameraID	[in] Identifier of the camera.
paramName	[in] Name of the parameter as string. Parameters are listed in Parameters for Camera
value	[out] Value of the parameter.

Returns

CAMERA_OK or error code as defined in CameraStatus.h. (See Return values for full list)

int _GetFloatParameter(char * *cameraID*, char * *paramName*, float * *value*)

```
C# function:   CameraStatusCode GetParameter(string cameraId, string name, out
float value)
```

Get a float parameter from a camera.

Parameters

cameraID	[in] Identifier of the camera.
paramName	[in] Name of the parameter as string. Parameters are listed in Parameters for Camera
value	[out] Value of the parameter.

Returns

CAMERA_OK or error code as defined in CameraStatus.h. (See Return values for full list)

int _GetIntParameter(char * *cameraID*, char * *paramName*, int * *value*)

```
C# function:   CameraStatusCode GetParameter(string cameraId, string name, out int
value)
```

Get an integer parameter from a camera.

Parameters

cameraID	[in] Identifier of the camera.
paramName	[in] Name of the parameter as string. Parameters are listed in Parameters for Camera
value	[out] Value of the parameter.

Returns

CAMERA_OK or error code as defined in CameraStatus.h. (See Return values for full list)

int _GetLayerFloatParameter(char * *cameraID*, int *layerId*, char * *paramName*, float * *value*)

```
C# function:   CameraStatusCode GetLayerParameter(string cameraId, int layerId,
string name, out float value)
```

Get a float parameter from a specific layer.

Parameters

cameraID	[in] Identifier for the camera, may be NULL for one camera systems.
layerId	[in] Id of the layer (0 = first layer).
paramName	[in] Name of the parameter as string.
value	The value.

Returns

CAMERA_OK or error code as defined in CameraError.h

int _GetLayerIntParameter(char * *cameraID*, int *layerId*, char * *paramName*, int * *value*)

```
C# function:  CameraStatusCode GetLayerParameter(string cameraId, int layerId,
string name, out int value)
```

Get an integer parameter from a specific layer.

Parameters

cameraID	[in] Identifier for the camera, may be NULL for one camera systems.
layerId	[in] Id of the layer (0 = first layer).
paramName	[in] Name of the parameter as string.
value	The value.

Returns

CAMERA_OK or error code as defined in CameraError.h

int _GetParameterSupport(char * *cameraID*, char * *paramName*, bool * *is_supported*)

```
C# function:  CameraStatusCode GetParameterSupport(string cameraId, string name,
out bool value)
```

Check is the parameter supported or not with the specified sensor

Parameters

cameraID	[in] Identifier of the camera.
paramName	[in] Name of checked parameter.
is_supported	[out] false if not supported. true if supported.

Returns

CAMERA_OK or error code

int _GetSerialNumber(char * *cameraId*, char * *serialNumber*)

```
C# function:  CameraStatusCode GetSerialNumber(string cameraId, out string
serialNumber)
```

Get the serial number of a camera.

Parameters

cameraId	[in] >Identifier of the camera.
serialNumber	[out] Serial number read from the sensor.

Returns

CAMERA_OK or error code as defined in CameraStatus.h. (See Return values for full list)

int _GetStringParameter(char * *cameraID*, char * *paramName*, char * *value*)

```
C# function:  CameraStatusCode GetParameter(string cameraId, string name, out
string value)
```

Get a string parameter from a camera.

Parameters

cameraID	[in] Identifier of the camera.
paramName	[in] Name of the parameter as string. Parameters are listed in Parameters for Camera
value	[out] Value of the parameter.

Returns

CAMERA_OK or error code as defined in CameraStatus.h. (See Return values for full list)

double _IntensityCalibrationProgress(char * *cameraID*, char * *progressBitmapFile*)
bool _IntensityCalibrationSave(char * *cameraId*, char * *outputFile*)
int _IntensityCalibrationStart(char * *cameraID*, char * *xCalibrationFile*, char * *zCalibrationFile*, char * *intensityCalibrationFile*, bool *smoothHorizontal*, double *gain*)
void _IntensityCalibrationStop(char * *cameraID*, char * *correctedBitmapFile*)
int _IsConnected(char * *cameraID*)

```
C# function:  CameraStatusCode IsConnected(string cameraId)
```

Check whether a camera is connected.

Parameters

cameraID	[in] Identifier of the camera.

Returns

CAMERA_OK or error code as defined in CameraStatus.h. (See Return values for full list)

int _Open(int * *cameraCount*, char *** *cameraIds*, int *timeout*)

```
C# function:  CameraStatusCode Open(ref int cameraCount, out List<string>
cameraIds, int timeout)
```

Opens the API and discovers cameras from the connected network. If expected camera count is not reached within given timeout, an additional timeout is waited until all expected cameras are found.

Parameters

cameraCount	[in] Expected camera count as an input. [out] Returns count of connected cameras.
cameraIds	[out] Pointer to a C-string array that will be populated with IDs of the discovered cameras. API will allocate and deallocate memory.
timeout	[in] Timeout to wait for replies in milliseconds.

Returns

CAMERA_OK or error code as defined in CameraStatus.h. (See Return values for full list)

int _SetBatchCallback(char * cameraID, int layerId, BatchCallback *callback*)

```
C# function:  CameraStatusCode SetBatchCallback(string cameraId, int layerId,
BatchCallback callback)
```

Set a Batch line callback for a camera.

Parameters

cameraID	[in] Identifier of the camera.
layerId	[in] Layer id. 0 is the first layer in defined sorting order. Multiple layers can be configured by calling this function multiple times. Range 0-9.
callback	[in] Callback method that is called whenever a line is received from the camera. 0 if callback is removed.

Returns

CAMERA_OK or error code as defined in CameraStatus.h. (See Return values for full list)

int _SetDoubleParameter(char * *cameraID*, char * *paramName*, double *value*)

```
C# function:  CameraStatusCode SetParameter(string cameraId, string name, double
value)
```

Set a double parameter to a camera.

Parameters

cameraID	[in] Identifier of the camera.
paramName	[in] Name of the parameter as string. Parameters are listed in Parameters for Camera
value	[in] Value of the parameter.

Returns

CAMERA_OK or error code as defined in CameraStatus.h. (See Return values for full list)

int _SetFloatParameter(char * *cameraID*, char * *paramName*, float *value*)

```
C# function:  CameraStatusCode SetParameter(string cameraId, string name, float
value)
```

Set a float parameter to a camera.

Parameters

cameraID	[in] Identifier of the camera.
paramName	[in] Name of the parameter as string. Parameters are listed in Parameters for Camera
value	[in] Value of the parameter.

Returns

CAMERA_OK or error code as defined in CameraStatus.h. (See Return values for full list)

int _SetIntParameter(char * *cameraID*, char * *paramName*, int *value*)

```
C# function:  CameraStatusCode SetParameter(string cameraId, string name, int
value)
```

Set an integer parameter to a camera.

Parameters

cameraID	[in] Identifier of the camera.
paramName	[in] Name of the parameter as string. Parameters are listed in Parameters for Camera
value	[in] Value of the parameter.

Returns

CAMERA_OK or error code as defined in CameraStatus.h. (See Return values for full list)

int _SetLayerFloatParameter(char * *cameraID*, int *layerId*, char * *paramName*, float *value*)

```
C# function:   CameraStatusCode SetLayerParameter(string cameraId,int layerId,
string name, float value)
```

Set a float parameter for a specific layer.

Parameters

cameraID	[in] Identifier of the camera.
layerId	[in] Id of the layer (0 = first layer).
paramName	[in] Name of the parameter as string. Parameters are listed in Parameters for Camera
value	[in] Value of the parameter.

Returns

CAMERA_OK or error code as defined in CameraStatus.h. (See Return values for full list)

int _SetLayerIntParameter(char * *cameraID*, int *layerId*, char * *paramName*, int *value*)

```
C# function:   CameraStatusCode SetLayerParameter(string cameraId, int layerId,
string name, int value)
```

Set an integer parameter for a specific layer.

Parameters

cameraID	[in] Identifier of the camera.
layerId	[in] Id of the layer (0 = first layer).
paramName	[in] Name of the parameter as string. Parameters are listed in Parameters for Camera
value	[in] Value of the parameter.

Returns

CAMERA_OK or error code as defined in CameraStatus.h. (See Return values for full list)

int _SetLineCallback(char * *cameraID*, int *layerId*, LineCallback *callback*)

```
C# function:   CameraStatusCode SetLineCallback(string cameraId, int layerId,
LineCallback callback)
```

Set a line reception callback for a camera.

Parameters

cameraID	[in] Identifier of the camera.
layerId	[in] Layer id. 0 is the first layer in defined sorting order. Multiple layers can be configured by calling this function multiple times. Range 0-3.
callback	[in] Callback method that is called whenever a line is received from the camera. 0 if callback is removed.

Returns

CAMERA_OK or error code as defined in CameraStatus.h. (See Return values for full list)

int _SetLineSortingOrder(char * *cameraID*, int *sortingOrder*)

```
C# function:  CameraStatusCode SetLineSortingOrder(string cameraId, SortingOrder
sortingOrder)
```

Set sorting order for layers in the line callback functions.

Parameters

cameraID	[in] Identifier of the camera.
sortingOrder	[in] enum SurfaceSorting defines values

Returns

CAMERA_OK or error code as defined in CameraStatus.h. (See Return values for full list)

int _SetPeakDetectionParameters(char * *cameraID*, MaterialType *type*, DetectionSensitivity *sensitivity*)

```
C# function:  CameraStatusCode SetPeakDetectionParameters(string cameraId,
MaterialType type, DetectionSensitivity sensitivity)
```

Simplified peak detection configuration based on material type. Note! Set PARAM_LAYER_MIN_THICKNESS before calling this function for transparent materials. Following parameters are set automatically PARAM_PEAK_FIR_LENGTH, PARAM_PEAK_AVERAGE_INTENSITY_FILTER_LENGTH, PARAM_SIGNAL_DETECTION_FILTER_LENGTH, PARAM_PEAK_AVER_FIR_LENGTH and PARAM_PEAK_THRESHOLD.

Parameters

cameraID	[in] Identifier for the camera, may be NULL for one camera systems.
type	[in] Inspected material type (MaterialType) .
sensitivity	[in] Sensitivity for the peak detection (DetectionSensitivity).

Returns

CAMERA_OK or error code as defined in CameraError.h

int _SetProfileCallback(char * *cameraID*, ProfileCallback *callback*)

```
C# function:  CameraStatusCode SetProfileCallback(string cameraId,
ProfileCallback callback)
```

Set a profile reception callback for a camera.

Parameters

cameraID	[in] Identifier of the camera.
callback	[in] Callback method that is called whenever a profile is received from the camera.

Returns

CAMERA_OK or error code as defined in CameraStatus.h. (See Return values for full list)

int _SetRawImageCallback(char * *cameraID*, RawImageCallback *callback*)

```
C# function:  CameraStatusCode SetRawImageCallback(string cameraId,
RawImageCallback callback)
```

Set a raw image reception callback for a camera.

Parameters

cameraID	[in] Identifier of the camera.
callback	[in] Callback method that is called whenever a raw image is received from the camera.

Returns

CAMERA_OK or error code as defined in CameraStatus.h. (See Return values for full list)

int _SetStringParameter(char * *cameraID*, char * *paramName*, char * *value*)

```
C# function:  CameraStatusCode SetParameter(string cameraId, string name, string
value)
```

Set a string parameter to a camera.

Parameters

cameraID	[in] Identifier of the camera.
paramName	[in] Name of the parameter as string. Parameters are listed in Parameters for Camera
value	[in] Value of the parameter.

Returns

CAMERA_OK or error code as defined in CameraStatus.h. (See Return values for full list)

int _StartGrabbing(char * *cameraID*)

```
C# function:  CameraStatusCode StartGrabbing(string cameraId)
```

Start data receiving data from a camera.

Parameters

cameraID	[in] Identifier of the camera.

Returns

CAMERA_OK or error code as defined in CameraStatus.h. (See Return values for full list)

int _StopGrabbing(char * *cameraID*)

```
C# function:   CameraStatusCode StopGrabbing(string cameraId)
```

Stop receiving data from a camera.

Parameters

cameraID	[in] Identifier of the camera.

Returns

CAMERA_OK or error code as defined in CameraStatus.h. (See Return values for full list)

9.3 Return values

9.3.1 Enumerations

- enum CameraStatus { CAMERA_OK =0, CAMERA_ERROR_INSTATION_FAILED =1, CAMERA_ERROR_NOT_CONNECTED =2, CAMERA_ERROR_IMAGE_RECEPTION_NOT_INSTALLED =3, CAMERA_ERROR_GET_IMAGE_FAILED =4, CAMERA_ERROR_START_GRABBING_FAILED =5, CAMERA_ERROR_STOP_GRABBING_FAILED =6, CAMERA_ERROR_GET_PEAKS_FAILED =7, CAMERA_ERROR_FAILED_TO_SET_FRAME_GRABBED_CALLBACK =8, CAMERA_ERROR_SET_PARAMETER_FAILED =9, CAMERA_ERROR_SET_UNKNOWN_PARAMETER =10, CAMERA_ERROR_CAMERA_NOT_FOUND =11, CAMERA_ERROR_PARAM_NOT_FOUND =12, CAMERA_ERROR_RECIPE_NOT_FOUND =13, CAMERA_ERROR_PARAM_TYPE_INVALID =14, CAMERA_ERROR_LOAD_DLL_FAILED =15, CAMERA_ERROR_FILE_NOT_FOUND = 16, CAMERA_ERROR_PARAM_VALUE_INVALID = 17, CAMERA_ERROR_SENSOR_CALIBRATION_FILE_NOT_SET = 18, CAMERA_ERROR_INVALID_FIR_LENGTH = 19, CAMERA_ERROR_HW_NOT_SUPPORTED = 20, CAMERA_ERROR_API_ALREADY_LOADED = 21, CAMERA_ERROR_MISSING_LIBRARY_FILES = 22, CAMERA_ERROR_INVALID_CALIBRATION_FILE = 23, CAMERA_ERROR_FLASH_DATA_LENGTH_EXCEEDED = 24, CAMERA_ERROR_FLASH_WRITE = 25, CAMERA_ERROR_FLASH_WRITE_FINISH = 26, CAMERA_ERROR_FLASH_READ = 27, CAMERA_ERROR_FILE_WRITE = 28, CAMERA_ERROR_UNINITIALIZED_FLASH = 29, CAMERA_ERROR_PARAMETER_READ_ONLY = 30, CAMERA_ERROR_SELFTEST_GRABBING_FAILED = 31, CAMERA_ERROR_CAMERA_NO_LISENCE = 32, CAMERA_ERROR_CALIB_FILE_ATTRIBUTE_NOT_FOUND = 33, CAMERA_ERROR_CALIB_FILE_ATTRIBUTE_INVALID = 34,

CAMERA_ERROR_INVALID_SOFTWARE_VERSION = 35,
CAMERA_ERROR_FIRMWARE_NOT_SUPPORTING = 36,
CAMERA_ERROR_UNKNOWN =100 }
This enum defines the possible return values for the Vevo.dll.

9.3.2 Detailed Description

9.3.3 Enumeration Type Documentation

enum CameraStatus

This enum defines the possible return values for the Vevo.dll.

Enumerator:

CAMERA_OK	The operation was executed successfully.
CAMERA_ERROR_INSTATION_FAILED	Could not instantiate the image receiver.
CAMERA_ERROR_NOT_CONNECTED	Camera is not connected
CAMERA_ERROR_IMAGE_RECEPTION_NOT_INSTALLED	Image reception is not installed.
CAMERA_ERROR_GET_IMAGE_FAILED	Call to get image method failed.
CAMERA_ERROR_START_GRABBING_FAILED	Couldn't start image grabbing.
CAMERA_ERROR_STOP_GRABBING_FAILED	Couldn't stop image grabbing.
CAMERA_ERROR_GET_PEAKS_FAILED	Error occured when trying to get image peaks.

CAMERA_ERROR_FAILED_TO_SET_FRAME_GRABBED_CALLBACK	Error occured when trying to set the frame grabbed callback.
CAMERA_ERROR_SET_PARAMETER_FAILED	Could not set a parameter.
CAMERA_ERROR_SET_UNKNOWN_PARAMETER	Trying to access unknown parameter.
CAMERA_ERROR_CAMERA_NOT_FOUND	The requested camera was not found.
CAMERA_ERROR_PARAM_NOT_FOUND	The requested parameter was not found.
CAMERA_ERROR_RECIPE_NOT_FOUND	The requested recipe could not be found.
CAMERA_ERROR_PARAM_TYPE_INVALID	The parameter type is not correct.
CAMERA_ERROR_LOAD_DLL_FAILED	Failed to load required DLL.
CAMERA_ERROR_FILE_NOT_FOUND	Provided file does not exist.
CAMERA_ERROR_PARAM_VALUE_INVALID	The parameter value is not correct.
CAMERA_ERROR_SENSOR_CALIBRATION_FILE_NOT_SET	Operation requires that sensor calibration file has been set.
CAMERA_ERROR_INVALID_FIR_LENGTH	FIR length is not supported.
CAMERA_ERROR_HW_NOT_SUPPORTED	Sensor hardware does not support the action.
CAMERA_ERROR_API_ALREADY_LOADED	API is already loaded.

CAMERA_ERROR_MISSING_LIBRARY_FILES	A library file is missing.
CAMERA_ERROR_INVALID_CALIBRATION_FILE	Calibration file is invalid.
CAMERA_ERROR_FLASH_DATA_LENGTH_EXCEEDED	Internal error code.
CAMERA_ERROR_FLASH_WRITE	Internal error code.
CAMERA_ERROR_FLASH_WRITE_FINISH	Internal error code.
CAMERA_ERROR_FLASH_READ	Internal error code.
CAMERA_ERROR_FILE_WRITE	Internal error code.
CAMERA_ERROR_UNINITIALIZED_FLASH	Internal error code.
CAMERA_ERROR_PARAMETER_READ_ONLY	Parameter is read-only.
CAMERA_ERROR_SELFTEST_GRABBING_FAILED	Selftest failed at the program start
CAMERA_ERROR_CAMERA_NO_LISENCE	Missing license for the camera
CAMERA_ERROR_CALIB_FILE_ATTRIBUTE_NOT_FOUND	Identifying attributes like sensor id or mac address, not available in calibration file.
CAMERA_ERROR_CALIB_FILE_ATTRIBUTE_INVALID	Calibration file attribute or value is incorrect.
CAMERA_ERROR_INVALID_SOFTWARE_VERSION	Software component version not valid for current product.

CAMERA_ERROR_FIRMWARE_NOT_SUPPORTING	Not supported by current firmware version.
CAMERA_ERROR_UNKNOWN	Any other error.

9.4 Data structures

9.4.1 Classes

- struct STRUCT_POINT
  Defines a point received by a callback handler for (ProfileCallback).
- struct HEADER
  Header associated to each raw image or profile received from the camera.
- struct FieldCalibrationResults
- struct DynSensorControl
  Stucture hold dynamic sensor control parameters.

9.4.2 Macros

- #define FS_HEADER_INPUT_STATE_BIT (1 << 0)
  Position of input state enable bit inside HEADER::enabled_fields.

- #define FS_HEADER_PULSE_WIDTH_BIT (1 << 1)
  Position of pulse width enable bit inside HEADER::enabled_fields.

- #define FS_HEADER_LOCATION_BIT (1 << 2)
  Position of location enable bit inside HEADER::enabled_fields.

- #define BIT_SET(field, bit) (field |= bit)
  Sets a bit inside a bitfield.

- #define BIT_CLEAR(field, bit) (field &= ~bit)
  Clears a bit inside a bitfield.

- #define BIT_IS_SET(field, bit) (field & bit)
  Checks if a bit inside field is set.

- #define NO_MEASUREMENT (float)(9999999)
  Not measured point in line and batch callback functions

- #define MAX_LAYERS 10
  Maximum number of layers

- #define NOT_MEASURED_POINT(x) ((x)>9999998)
  Macro to check whether the point is measured or not

- #define MAX_GROOVE_MEASUREMENTS 5
  Results of field calibration function. Measurements in micrometers.

9.4.3 Enumerations

- enum InterleavePhase { Long =0, Short = 1 }
  Enumeration for the interleaving phase.
- enum SurfaceSorting { FROM_TOP_TO_BOTTOM =0, FROM_BOTTOM_TO_TOP =1, FROM_MAX_INTENSITY_TO_LOWER =2, FROM_MAX_INTENSITY_TO_LOWER_AND_TOP_TO_BOTTOM = 3 }
- enum MaterialType { Opaque = 0, Mirror = 1, Transparent = 2 }
  Enumeration for the material type.
- enum DetectionSensitivity { BestAccuracy = 0, BestSensitivity = 1 }
  Enumeration for sensitivity used in peak detection configuration.

9.4.4 Detailed Description

9.4.5 Macro Definition Documentation

#define BIT_CLEAR(field, bit) (field &= ~bit)

Clears a bit inside a bitfield.

#define BIT_IS_SET(field, bit) (field & bit)

Checks if a bit inside field is set.

#define BIT_SET(field, bit) (field |= bit)

Sets a bit inside a bitfield.

#define FS_HEADER_INPUT_STATE_BIT (1 << 0)

Position of input state enable bit inside HEADER::enabled_fields.

#define FS_HEADER_LOCATION_BIT (1 << 2)

Position of location enable bit inside HEADER::enabled_fields.

#define FS_HEADER_PULSE_WIDTH_BIT (1 << 1)

Position of pulse width enable bit inside HEADER::enabled_fields.

#define MAX_GROOVE_MEASUREMENTS 5

Results of field calibration function. Measurements in micrometers.

#define MAX_LAYERS 10

Maximum number of layers

#define NO_MEASUREMENT (float)(9999999)

Not measured point in line and batch callback functions

#define NOT_MEASURED_POINT(x) ((x)>9999998)

Macro to check whether the point is measured or not

9.4.6 Enumeration Type Documentation

enum DetectionSensitivity

Enumeration for sensitivity used in peak detection configuration.

Enumerator:

BestAccuracy	BestAccuracy is used when when the accuracy is prioritized.
BestSensitivity	BestSensitivity used when dark areas need to be detected.

enum InterleavePhase

Enumeration for the interleaving phase.

Enumerator:

Long	Peak is taken with long exposure time.
Short	Peak is taken with short exposure time.

enum MaterialType

Enumeration for the material type.

Enumerator:

Opaque	Opaque is used when a top surface has opaque areas.
Mirror	Mirror is used for highly reflective surfaces.
Transparent	Transparent is used for materials having transparent surfaces.

enum SurfaceSorting

Enumerator:

FROM_TOP_TO_BOTTOM	Sorting layers from top to bottom
FROM_BOTTOM_TO_TOP	Sorting layers from bottom to top
FROM_MAX_INTENSITY_TO_LOWER	Sorting layers according to intensity value. Highest intensity is layer 0.
FROM_MAX_INTENSITY_TO_L	Sorting layers according to intensity value and then from top to bottom.

OWER_AND_TO P_TO_BOTTOM	

9.5 Parameters for Camera

9.5.1 Macros

- #define PARAM_LOAD_RECIPE "load_recipe"
- #define PARAM_SAVE_RECIPE "save_recipe"
- #define PARAM_PEAK_ENABLE "peak_enable"
- #define PARAM_PEAK_THRESHOLD "peak_treshold"
- #define PARAM_PEAK_FIR_LENGTH "peak_fir_length"
- #define PARAM_SENSOR_DATA_IN_FLASH "sensor_data_in_flash"
- #define PARAM_SENSOR_CALIBRATION_FILE "sensor_calibration_file"
- #define PARAM_SENSOR_X_CALIBRATION_FILE "sensor_x_calibration_file"
- #define PARAM_SENSOR_INTENSITY_CALIBRATION_FILE "sensor_intensity_calibration_file"
- #define PARAM_PEAK_Y_UNIT "peak_y_unit"
- #define PARAM_PEAK_X_UNIT "peak_x_unit"
- #define PARAM_INTENSITY_CORRECTION_ENABLE "intensity_correction_enable"
- #define PARAM_SENSOR_TYPE "sensor_type"
- #define PARAM_GAIN "gain"
- #define PARAM_IMAGE_HEIGHT "image_height"
- #define PARAM_IMAGE_OFFSETY "image_offsety"
- #define PARAM_SURFACE_MODE "surface_mode"
- #define PARAM_HDR_ENABLED "hdr_enable"
- #define PARAM_HDR_KP1_POS "hdr_kp1_pos"
- #define PARAM_HDR_KP2_POS "hdr_kp2_pos"
- #define PARAM_HDR_VLOW2 "hdr_vlow2"
- #define PARAM_HDR_VLOW3 "hdr_vlow3"
- #define PARAM_HDR_KP1_POS_FLOAT "hdr_kp1_pos_float"
- #define PARAM_HDR_KP2_POS_FLOAT "hdr_kp2_pos_float"
- #define PARAM_HDR_VLOW2_FLOAT "hdr_vlow2_float"
- #define PARAM_HDR_VLOW3_FLOAT "hdr_vlow3_float"
- #define PARAM_X_LIMIT_MIN "x_limit_min"
- #define PARAM_X_LIMIT_MAX "x_limit_max"
- #define PARAM_SATURATION_COUNT "saturation_count"
- #define PARAM_AVERAGE_FRAME_INTENSITY "average_frame_intensity"
- #define PARAM_DYN_SENSOR_CTRL_ENABLE "dyn_sensor_control_enable"
- #define PARAM_MAX_POINT_COUNT "max_point_count"
- #define PARAM_MTU "mtu"
- #define PARAM_IFG "ifg"

- #define PARAM_HEARTBEAT "heartbeat"
- #define PARAM_IMAGE_HEIGHT_ZERO_POSITION "image_height_zero_position"
- #define PARAM_DEVICE_SERIAL_NUMBER "device_serial_number"
- #define PARAM_USER_FLASH_WRITE_TIME "user_flash_write_time"
- #define PARAM_SIGNAL_DETECTION_FILTER_LENGTH "signal_detection_filter_length"
- #define PARAM_PEAK_AVERAGE_INTENSITY_FILTER_LENGTH "peak_average_intensity_filter_length"
- #define PARAM_LAYER_INTENSITY_TYPE "layer_intensity_type"
- #define PARAM_LAYER_MAX_THICKNESS "layer_max_thickness"
- #define PARAM_LAYER_MIN_THICKNESS "layer_min_thickness"
- #define PARAM_LAYER_EFFECTIVE_REFRACTIVE_INDEX "layer_refractive_index"
- #define PARAM_THICKNESS_MODE "thickness_mode"

9.5.2 Detailed Description

9.5.3 Macro Definition Documentation

#define PARAM_AVERAGE_FRAME_INTENSITY "average_frame_intensity"

C# name: SensorParameter.AvgFrameIntensity

Average intensity level of a frame.

- **Parameter type:** int
- **Unit** : 8-bit grayscale intensity

#define PARAM_DEVICE_SERIAL_NUMBER "device_serial_number"

C# name: SensorParameter.DeviceSerialNumber

Device Serial Number, defined by production.

- **Parameter type:** string
- **Values** : Null-terminated string of product serial number. "" if the serial number is not written (products manufactured before 02/2019).

#define PARAM_DYN_SENSOR_CTRL_ENABLE "dyn_sensor_control_enable"

C# name: SensorParameter.DynSensorCtrlEnable

Determines if dynamic sensor control is enabled or not. Dynamic control parameters given by interface function xxx to set register parameters REG_DYN_SENSOR_CTRL_N_LOCATION,
REG_DYN_SENSOR_CTRL_N_HEIGHT,

REG_DYN_SENSOR_CTRL_N_HEIGHT_OFFSET,
REG_DYN_SENSOR_CTRL_N_PULSE_WIDTH N = 1, 2, ..., 10.

- **Parameter type:** int
- **Unit** : 1 enabled, 0 disabled.

#define PARAM_GAIN "gain"

C# name: SensorParameter.Gain

Gain sets the sensitivity of the sensor. The higher the gain, the more sensitive the sensor. Image grabbing must be stopped before setting the gain in LCI1220 and LCI1620 sensors. The default setting is suitable for the most of the applications. Use higher Gain setting only when the signal is very weak or short LED Pulse Width is required due to high measurement frequency. Increasing Gain increases also noise and background signal level, therefore Gain increase might also require increasing the Threshold. Optimal Gain value for specific measurement must be found experimentally.

- **Parameter type:** double
- **Range** : 1.0 (default) - 3.2 for LCI401, LCI1200, LCI1201 and LCI1600 sensors
- 2.0 (default) - 8.0 for LCI1220 and LCI1620 sensors

#define PARAM_HDR_ENABLED "hdr_enable"

C# name: SensorParameter.HdrEnabled

Enables or disables the HDR mode, where the dynamic range of luminosity is greater than normally.

- **Parameter type:** int
- **Values** : 0 = disabled, 1 = enabled.

#define PARAM_HDR_KP1_POS "hdr_kp1_pos"

C# name: SensorParameter.Kp1Pos

The knee point 1 position for the HDR. This parameter is used only when the HDR is enabled.

- **Parameter type:** int
- **Valid range:** 1 – 99

#define PARAM_HDR_KP1_POS_FLOAT "hdr_kp1_pos_float"

C# name: SensorParameter.Kp1PosFloat

The knee point 1 position in % for the HDR. This parameter is used only when the HDR is enabled.

- **Parameter type:** float
- **Valid range:** 0.02 - 99.99

#define PARAM_HDR_KP2_POS "hdr_kp2_pos"

C# name: SensorParameter.Kp2Pos

The knee point 2 position for the HDR. This parameter is used only when the HDR is enabled.

- **Parameter type:** int

#define PARAM_HDR_KP2_POS_FLOAT "hdr_kp2_pos_float"

C# name: SensorParameter.Kp2PosFloat

The knee point 2 position in % for the HDR. This parameter is used only when the HDR is enabled.

- **Parameter type:** float

#define PARAM_HDR_VLOW2 "hdr_vlow2"

C# name: SensorParameter.VLow2

The VLow2 (sensitivity, reference voltage) value for the HDR. This parameter is used only when the HDR is enabled.

- **Parameter type:** int

#define PARAM_HDR_VLOW2_FLOAT "hdr_vlow2_float"

C# name: SensorParameter.VLow2Float

The VLow2 (sensitivity, reference voltage) value for the HDR. This parameter is used only when the HDR is enabled.

- **Parameter type:** float

#define PARAM_HDR_VLOW3 "hdr_vlow3"

C# name: SensorParameter.VLow3

The VLow3 (sensitivity, reference voltage) value for the HDR. This parameter is used only when the HDR is enabled.

- **Parameter type:** int

#define PARAM_HDR_VLOW3_FLOAT "hdr_vlow3_float"

C# name: SensorParameter.VLow3Float

The VLow3 (sensitivity, reference voltage) value for the HDR. This parameter is used only when the HDR is enabled.

- **Parameter type:** float

#define PARAM_HEARTBEAT "heartbeat"

C# name: SensorParameter.Heartbeat

Communication heartbeat value. Heartbeat is the interval in milliseconds to check if the connection is alive.

- **Parameter type:** int
- **Values:** In production use the heartbeat should be set to ~5000. For application debugging purposes, greater values, e.g. 99999, are often needed.

#define PARAM_IFG "ifg"

C# name: SensorParameter.Ifg

Sets Interpacket Delay. Min. value is 20. Available only with LCI401, LCI1200, LCI1201 and LCI1600 sensors.

- **Parameter type:** int
- **Unit** : bit times

#define PARAM_IMAGE_HEIGHT "image_height"

C# name: SensorParameter.Height

Height of the sensor image in pixel unit. Small values can be used to achieve higher frequency. Used in conjunction with PARAM_IMAGE_OFFSETY. Sum of image height and image offset y shall not exceed the maximum sensor height.

- **Parameter type:** int
- **Unit** : pixels
- **Range** : 8 - 1088 for LCI401, LCI1200, LCI1201 and LCI1600 sensors
- 8 - 1400 for LCI1220 and LCI1620 sensors

#define PARAM_IMAGE_HEIGHT_ZERO_POSITION "image_height_zero_position"

C# name: SensorParameter.ImageHeightZeroPosition

Returns image row z coordinate where height has been tuned nearest to '0' by calibration file on use.

- **Parameter type:** int
- **Unit** : pixel location

#define PARAM_IMAGE_OFFSETY "image_offsety"

C# name: SensorParameter.OffsetY

Set the Y offset for the sensor. Used in conjunction with PARAM_IMAGE_HEIGHT.

- **Parameter type:** int
- **Unit** : pixels
- **Range** : -1 - 1080 for LCI401, LCI1200, LCI1201 and LCI1600 sensors. -1 - 1392 for LCI1220 and LCI1620 sensors. If value is greater than 0, it should be noticed that the sum of PARAM_IMAGE_OFFSETY and PARAM_IMAGE_HEIGHT should not exceed the maximum sensor height. If value is -1 offset is calculated automatically from the calibration zero height (PARAM_IMAGE_HEIGHT must be set first).

#define PARAM_INTENSITY_CORRECTION_ENABLE "intensity_correction_enable"

C# name: SensorParameter.IntensityCorrectionEnable

Enables or disables the intensity correction for the peak data.

- **Parameter type:** int
- **Values** : 0 = intensity correction is disabled (default), 1 = intensity correction is disabled.

#define PARAM_LAYER_EFFECTIVE_REFRACTIVE_INDEX "layer_refractive_index"

C# name: SensorParameter.LayerRefractiveIndex

Specifies the effective refractive index of a material. _SetLayerFloatParameter function shall be used for this parameter.

- **Parameter type:** float
- **Layer:** layer id (0=first layer)
- **Values** : The effective refractive index (1.0 by default)

#define PARAM_LAYER_INTENSITY_TYPE "layer_intensity_type"

C# name: SensorParameter.LayerIntensityType

Specifies which filter is used for intensity values in linecallback and batchcallback functions. _SetLayerIntParameter function shall be used for this parameter.

- **Parameter type:** int
- **Layer:** layer id (0=first layer)
- **Values** : 0 = Average intensity filter, 1 = Signal detection filter

#define PARAM_LAYER_MAX_THICKNESS "layer_max_thickness"

C# name: SensorParameter.LayerMaxThickness

Specifies maximum thickness for the layer. Used for missing surface detection. _SetLayerFloatParameter function shall be used for this parameter.

- **Parameter type:** float
- **Layer:** layer id (0=first layer)
- **Values** : Thickness in um (0, if not used)

#define PARAM_LAYER_MIN_THICKNESS "layer_min_thickness"

C# name: SensorParameter.LayerMinThickness

Specifies minimum thickness for the layer. _SetFloatParameter function sets same value for all layers. _SetLayerFloatParameter function shall be used if layer specific value is needed.

- **Parameter type:** float
- **Layer:** layer id (0=first layer)
- **Values** : Thickness in um (0, if not used)

#define PARAM_LOAD_RECIPE "load_recipe"

C# name: SensorParameter.LoadRecipe

Loads a sensor configuration from the specified recipe.

- **Parameter type:** string
- **Values** : Name of the recipe. Loads from the default folder (C:\FocalSpec\SDK Recipes) if path is not given.

#define PARAM_MAX_POINT_COUNT "max_point_count"

C# name: SensorParameter.MaxPointCount

Maximum point count per profile. Change this before calling _StartGrabbing(). Default value is fine if number of surfaces is less than 5.

- **Parameter type:** int
- **Unit** : Max number of points. Default value is 10000.

#define PARAM_MTU "mtu"

C# name: SensorParameter.Mtu

Sets Maximum Transmission Unit (MTU). Available only with LCI401, LCI1200, LCI1201 and LCI1600 sensors.

- **Parameter type:** int
- **Unit** : bytes

#define PARAM_PEAK_AVERAGE_INTENSITY_FILTER_LENGTH "peak_average_intensity_filter_length"

C# name: `SensorParameter.PeakAverageIntensityFilterLength`

The filter is used for averaging intensity values of the detected peaks. With high average intensity filter length, the best possible intensity image is produced.

- **Parameter type:** int
- **Values** : Eligible values are 2, 4, 6, 8, 10, 12, 14 or 16.

#define PARAM_PEAK_ENABLE "peak_enable"

C# name: `SensorParameter.PeakEnabled`

Controls the output mode of the camera. If enabled, peaks are output to ProfileCallback, LineCallback and BatchCallback functions. If disabled, raw images are output to RawImageCallback.

- **Parameter type:** int
- **Values** : 0 = raw image output is active, 1 = peak cloud output is active.

#define PARAM_PEAK_FIR_LENGTH "peak_fir_length"

C# name: `SensorParameter.FirLength`

FIR parameter is used to find the detected peak position in the z-direction. Higher the FIR, more accurate the peak position is. With lower FIR, you can detect thinner layers in transparent layered material.

- **Parameter type:** int
- **Values** : Eligible values are 2, 4, 6, 8, 10, 12, 14 or 16.

#define PARAM_PEAK_THRESHOLD "peak_treshold"

C# name: `SensorParameter.PeakThreshold`

The intensity threshold value used in peak detection. If the peak intensity is below this limit, the peak gets discarded

- **Parameter type:** int
- **Unit:** 8-bit grayscale intensity value (2-200). Default value is 20.

#define PARAM_PEAK_X_UNIT "peak_x_unit"

C# name: `SensorParameter.PeakXUnit`

Controls the unit of measure of X of the peaks. Note that this parameter has effect only if peak cloud output is active (PARAM_PEAK_ENABLE is set to 1).

- **Parameter type:** int
- **Values** : 0 = peak X is in sensor pixel unit, 1 = peak X is in micrometers.

#define PARAM_PEAK_Y_UNIT "peak_y_unit"

C# name: `SensorParameter.PeakYUnit`

Controls the unit of measure of Y of the peaks. Note that this parameter has effect only if peak cloud output is active (PARAM_PEAK_ENABLE is set to 1).

- **Parameter type:** int
- **Values** : 0 = peak Y is in sensor pixel unit, 1 = peak Y is in micrometers.

#define PARAM_SATURATION_COUNT "saturation_count"

C# name: SensorParameter.SaturationCount

The number of sensor pixels in saturation state.

- **Parameter type:** int
- **Unit** : Count

#define PARAM_SAVE_RECIPE "save_recipe"

C# name: SensorParameter.SaveRecipe

Saves the current sensor configuration to the specified recipe.

- **Parameter type:** string
- **Values** : Name of the recipe. Saves to the default folder (C:\FocalSpec\SDK Recipes) if path is not given.

#define PARAM_SENSOR_CALIBRATION_FILE "sensor_calibration_file"

C# name: SensorParameter.ZCalibrationFile

Path of the sensor Z calibration file. No need to set if calibration files are in the sensor.

- **Parameter type:** string
- **Values** : Null-terminated string path to the file.

#define PARAM_SENSOR_DATA_IN_FLASH "sensor_data_in_flash"

C# name: SensorParameter.SensorDataInFlash

Read flag if e.g. Sensor type and calibration data is written to sensor flash memory.

- **Parameter type:** int
- **Unit** : 1 if supported, 0 if not.

#define PARAM_SENSOR_INTENSITY_CALIBRATION_FILE "sensor_intensity_calibration_file"

C# name: SensorParameter.IntensityCalibrationFile

Path of the sensor intensity calibration file.

Parameter type: string

- **Values** : Null-terminated string path to the file.

#define PARAM_SENSOR_TYPE "sensor_type"

C# name: SensorParameter.SensorType

Returns sensor type identifier without LCI prefix.

- **Parameter type:** int
- **Values** :

```
            400 = LCI400            1200 = LCI1200            1600 = LCI1600
401 = LCI401            1201 = LCI1201          1220 = LCI1220          1620
= LCI1620              -1 if not set
```

#define PARAM_SENSOR_X_CALIBRATION_FILE "sensor_x_calibration_file"

C# name: SensorParameter.XCalibrationFile

Path of the sensor X calibration file. No need to set if calibration files are in the sensor.

- **Parameter type:** string

- **Values** : Null-terminated string path to the file.

#define PARAM_SIGNAL_DETECTION_FILTER_LENGTH "signal_detection_filter_length"

C# name: SensorParameter.SignalDetectionFilterLength

Detection filter is used to average the signal to reduce noise before threshold (PARAM_PEAK_THRESHOLD). Low detection filter length is useful when the detected signal is rather faint.

- **Parameter type:** int
- **Values** : Eligible values are 2, 4, 6, 8, 10, 12, 14 or 16.

#define PARAM_SURFACE_MODE "surface_mode"

C# name: SensorParameter.SurfaceMode

Sort peaks and filter top, bottom or brightest profile.

- **Parameter type:** int
- **Values** :
- Sensors LCI401, LCI1200, LCI1220 and LCI1620: 0 = disabled (default), 1 = bottom, 2 = top, 3 = brightest, 4 = two two top-most, 5 = two top-most.
- Sensors LCI1201 and LCI1600: 0 = disabled (default), 1 = top, 2 = bottom, 3 = brightest, 4 = two two top-most, 5 = two two top-most.

#define PARAM_THICKNESS_MODE "thickness_mode"

C# name: SensorParameter.ThicknessMode

Calculates thicknesses for layers in line and batch callback functions. The first surface is returned as a world z-coordinate and

following layers are calculated as thickness values. Note that thickness mode parameter is not saved to the recipe.

- **Parameter type:** int
- **Values** : 1 - thickness mode enabled, 0 - disabled (default)

#define PARAM_USER_FLASH_WRITE_TIME "user_flash_write_time"

C# name: SensorParameter.UserFlashWriteTime

UTC timestamp when user data was written when available.

- **Parameter type:** string
- **Values** : Null-terminated formatted string from time of date value.

#define PARAM_X_LIMIT_MAX "x_limit_max"

C# name: SensorParameter.XLimitMax

Upper X threshold for statistics collection (PARAM_SATURATION_COUNT, PARAM_AVERAGE_FRAME_INTENSITY) and AGC input profile.

- **Parameter type:** int
- **Unit** : Pixels
- **Range** : Default value is 2047.

#define PARAM_X_LIMIT_MIN "x_limit_min"

C# name: SensorParameter.XLimitMin

Lower X threshold for statistics collection (PARAM_SATURATION_COUNT, PARAM_AVERAGE_FRAME_INTENSITY) and AGC input profile.

- **Parameter type:** int
- **Unit** : Pixels
- **Range** : Default value is 0.

9.6 Parameters for Acquisition

9.6.1 Macros

- #define PARAM_REG_PULSE_DIVIDER "reg_pulse_divider"
- #define PARAM_REG_PULSE_FREQ "reg_pulse_freq"
- #define PARAM_REORDERING "reordering"
- #define PARAM_REORDERING_SPAN "reordering_span"
- #define PARAM_REORDERING_DEVIATION "reordering_deviation"
- #define PARAM_FREQUENCY_CALCULATION "frequency_calculation"
- #define PARAM_LINE_CALLBACK_MAX_LENGTH "line_callback_max_length"
- #define PARAM_LINE_CALLBACK_XRES "line_callback_xres"
- #define PARAM_BATCH_LENGTH "batch_length"
- #define PARAM_BATCH_TIMEOUT "batch_timeout"
- #define PARAM_GRABBING "grabbing"

9.6.2 Detailed Description

9.6.3 Macro Definition Documentation

#define PARAM_BATCH_LENGTH "batch_length"

C# name: SensorParameter.BatchLength

Defines batch length (number of lines) on the batch run.

- **Parameter type:** int
- **Unit** : Line count

#define PARAM_BATCH_TIMEOUT "batch_timeout"

C# name: SensorParameter.BatchTimeout

Defines batch timeout timer. Interrupts collecting batch if total count lines not received before time expires. Incomplete batch is delivered for callback.

- **Parameter type:** int
- **Unit** : Milliseconds

#define PARAM_FREQUENCY_CALCULATION "frequency_calculation"

C# name: SensorParameter.FrequencyCalculation

Enable or disable frequency calculation in header of the profile. 0 = disabled, 1 = enabled (default)

- **Parameter type:** int

#define PARAM_GRABBING "grabbing"

C# name: SensorParameter.Grabbing

Read only value indicating the camera grabbing status.

- **Parameter type:** int
- **Values** : 0 = stopped, 1 = grabbing is on.

#define PARAM_LINE_CALLBACK_MAX_LENGTH "line_callback_max_length"

C# name: SensorParameter.LineCallbackMaxLength

Number of pixels expected from the line callback function. Can be called before grabbing but after initialization. Parameter is read-only.

- **Parameter type:** int
- **Unit** : number of pixels

#define PARAM_LINE_CALLBACK_XRES "line_callback_xres"

C# name: SensorParameter.LineCallbackXRes

Pixel size expected from the line callback function. Can be called before grabbing but after initialization. Parameter is read-only.

- **Parameter type:** float
- **Unit** : Micrometers

#define PARAM_REG_PULSE_DIVIDER "reg_pulse_divider"

C# name: SensorParameter.PulseDivider

Skips camera trigger pulses. If value if N, then every Nth pulses are triggered. If value is equal to one, then all trigger pulses are served.

- **Parameter type:** int
- **Unit:** N

#define PARAM_REG_PULSE_FREQ "reg_pulse_freq"

C# name: SensorParameter.PulseFrequency

Set the frequency for image capture.

- **Parameter type:** int
- **Unit:** Hz
- **Value:** 0 = external pulsing, > 0 internal pulsing.

#define PARAM_REORDERING "reordering"

C# name: SensorParameter.Reordering

Enable or disable line reordering according to header.index attribute. 0 = disabled (default), 1 = enabled

- **Parameter type:** int

9.6.3.1 #define PARAM_REORDERING_DEVIATION "reordering_deviation"

C# name: SensorParameter.ReorderingDeviation

Defines reordering maximum index step deviation between subsequent lines. Exceeding triggers buffer clearing.

- **Parameter type:** int

9.6.3.2 #define PARAM_REORDERING_SPAN "reordering_span"

C# name: SensorParameter.ReorderingSpan

Defines reordering time span max time window in milliseconds.

- **Parameter type:** int

9.6.3.3

9.7 Parameters for Illumination

9.7.1 Macros

- #define PARAM_REG_PULSE_WIDTH_FLOAT "reg_pulse_width_float"
- #define PARAM_AGC_ENABLED "agc_enabled"
- #define PARAM_AGC_PULSE_WIDTH_LIMIT "agc_pulse_width_limit"
- #define PARAM_AGC_MIN_PULSE_WIDTH_LIMIT "agc_minimum_pulse_width_limit"
- #define PARAM_AGC_GAIN "agc_gain"
- #define PARAM_AGC_TARGET "agc_target"
- #define PARAM_PULSE_CURRENT_MIDDLE "pulse_current_middle"
- #define PARAM_PULSE_CURRENT_EDGES "pulse_current_edges"

9.7.2 Detailed Description

9.7.3 Macro Definition Documentation

9.7.3.1 #define PARAM_AGC_ENABLED "agc_enabled"

C# name: SensorParameter.AgcEnabled

Is Automatic Gain Control (AGC) enabled. If enabled, AGC will automatically tune the pulse width based on the signal from the surface(s).

- **Parameter type:** int
- **Range** : 0,1. Default value is 0 (disabled).

9.7.3.2 #define PARAM_AGC_GAIN "agc_gain"

C# name: SensorParameter.AgcGain

Gain for AGC P-controller.

- **Parameter type:** float
- **Range** : Default value is 1.2F.

9.7.3.3 #define PARAM_AGC_MIN_PULSE_WIDTH_LIMIT "agc_minimum_pulse_width_limit"

C# name: SensorParameter.AgcMinPulseWidthLimit

AGC pulse width minimum limit.

- **Parameter type:** int
- **Unit** : Microseconds
- **Range** : Default value is 1.

9.7.3.4 #define PARAM_AGC_PULSE_WIDTH_LIMIT "agc_pulse_width_limit"

C# name: SensorParameter.AgcWiLimit

AGC pulse width upper limit.

- **Parameter type:** int
- **Unit** : Microseconds
- **Range** : Default value is 2000.

9.7.3.5 #define PARAM_AGC_TARGET "agc_target"

C# name: SensorParameter.AgcTarget

AGC target intensity (8-bit grayscale).

- **Parameter type:** float
- **Range** : 0.0F-255.0F. Default value is 80.0f.

9.7.3.6 #define PARAM_PULSE_CURRENT_EDGES "pulse_current_edges"

C# name: SensorParameter.PulseCurrentEdges

Pulse current in the edges of the sensor.

- **Parameter type:** float
- **Range** : 0.1 - 1.0 ampere

9.7.3.7 #define PARAM_PULSE_CURRENT_MIDDLE "pulse_current_middle"

C# name: SensorParameter.PulseCurrentMiddle

Pulse current in the middle section of the sensor.

- **Parameter type:** float
- **Range** : 0.1 - 1.0 ampere

9.7.3.8 #define PARAM_REG_PULSE_WIDTH_FLOAT "reg_pulse_width_float"

C# name: SensorParameter.PulseWidthFloat

Illumination pulse width. Replaces PARAM_LED_DURATION.

- **Parameter type:** float
- **Unit:** Microseconds

9.7.3.9

9.8 Parameters for I/O

9.8.1 Macros

- #define PARAM_PEAK_COUNT_LIMIT "peak_count_limit"
- #define PARAM_OUTPUT_PIN_STATE "output_pin_state"
- #define PARAM_INPUT_PIN_STATE "input_pin_state"
- #define PARAM_Y_TOLERANCE_MIN "y_tolerance_min"
- #define PARAM_Y_TOLERANCE_MAX "y_tolerance_max"
- #define PARAM_TRIGGER_SOURCE "trigger_source"
- #define PARAM_TRIGGER_DISABLE_SOURCE "trigger_disable_source"
- #define PARAM_TRIGGER_ZERO_SOURCE "trigger_zero_source"
- #define PARAM_TRIGGER_MINUS_SOURCE "trigger_minus_source"
- #define PARAM_TRIGGER_DEBOUNCE "trigger_debounce"
- #define PARAM_SENSOR_BOARD_TEMPERATURE "sensor_board_temperature"
- #define PARAM_SENSOR_TEMPERATURE "sensor_temperature"
- #define PARAM_CHIP_TEMPERATURE "chip_temperature"
- #define PARAM_SENSOR_BOARD_TEMPERATURE_FLOAT "sensor_board_temperature_float"
- #define PARAM_ILLUMINATOR_TEMPERATURE "illuminator_temperature"
- #define PARAM_FRONT_PANEL_TEMPERATURE "front_panel_temperature"
- #define PARAM_SENSOR_GP_IO "sensor_gp_io"

9.8.2 Detailed Description

9.8.3 Macro Definition Documentation

9.8.3.1 #define PARAM_CHIP_TEMPERATURE "chip_temperature"

C# name: SensorParameter.ChipTemperature

Read temperature [Celsius degrees] on sensor chip. This is available only with LCI1220 and LCI1620 sensors.

- **Parameter type:** float

9.8.3.2 #define PARAM_FRONT_PANEL_TEMPERATURE "front_panel_temperature"

C# name: SensorParameter.FrontPanelTemperature

Read temperature [Celsius degrees] on sensor front panel. This is available only with LCI1220 and LCI1620 sensors.

- **Parameter type:** float

9.8.3.3 #define PARAM_ILLUMINATOR_TEMPERATURE "illuminator_temperature"

C# name: SensorParameter.IlluminatorTemperature

Read temperature [Celsius degrees] on illuminator. This is available only with LCI1220 and LCI1620 sensors.

- **Parameter type:** float

9.8.3.4 #define PARAM_INPUT_PIN_STATE "input_pin_state"

C# name: SensorParameter.InputPinState

Bitmask for reading inputs.

- **Parameter type:** int
- **Range** : 4 input channels (0x1 - 0xF). Default value is 0.

9.8.3.5 #define PARAM_OUTPUT_PIN_STATE "output_pin_state"

C# name: SensorParameter.OutputPinState

Bitmask for driving outputs.

- **Parameter type:** int
- **Range** : 4 valid outputs (0x1 - 0xF). Default value is 0.

9.8.3.6 #define PARAM_PEAK_COUNT_LIMIT "peak_count_limit"

C# name: SensorParameter.PeakCountLimit

If the peak core detects more peaks than this threshold, a warning LED is enabled.

- **Parameter type:** int
- **Range** : Default value is 10000.

9.8.3.7 #define PARAM_SENSOR_BOARD_TEMPERATURE "sensor_board_temperature"

C# name: SensorParameter.SensorBoardTemperature

Read temperature [Celsius degrees] on sensor board. This is available only with LCI401, LCI1200, LCI1201 and LCI1600 sensors.

- **Parameter type:** int

9.8.3.8 #define PARAM_SENSOR_BOARD_TEMPERATURE_FLOAT "sensor_board_temperature_float"

C# name: SensorParameter.SensorBoardTemperatureFloat

Read temperature [Celsius degrees] on sensor board. This is available only with LCI1220 and LCI1620 sensors.

- **Parameter type:** float

9.8.3.9 #define PARAM_SENSOR_GP_IO "sensor_gp_io"

C# name: SensorParameter.SensorGpIo

General Purpose I/O GPIO 3-bits for rev-6-ele, 2-bits for G3.

- **Parameter type:** int
- **Range** : . (0 = disabled, 1=output 1, 2=output 3, 3=output 3 and 7=all three)

9.8.3.10 #define PARAM_SENSOR_TEMPERATURE "sensor_temperature"

C# name: SensorParameter.SensorTemperature

Read temperature [Celsius degrees] on sensor chip. This is available only with LCI401, LCI1200, LCI1201 and LCI1600 sensors.

- **Parameter type:** int

9.8.3.11 #define PARAM_TRIGGER_DEBOUNCE "trigger_debounce"

C# name: SensorParameter.TriggerDebounce

Digital filter debounce time for inputs. Pulses below this threshold are ignored.

- **Parameter type:** float
- **Unit** : Microseconds
- **Range** : Default value is 1.0.

9.8.3.12 #define PARAM_TRIGGER_DISABLE_SOURCE "trigger_disable_source"

C# name: SensorParameter.TriggerDisableSource

Trigger disable input source pin number.

- **Parameter type:** int
- **Range** : 0-3. Default is 0, disabled.

9.8.3.13 #define PARAM_TRIGGER_MINUS_SOURCE "trigger_minus_source"

C# name: SensorParameter.TriggerMinusSource

Trigger minus input source pin number.

- **Parameter type:** int
- **Range** : 0-3. Default is input 2. Disabled is 0.

9.8.3.14 #define PARAM_TRIGGER_SOURCE "trigger_source"

C# name: SensorParameter.TriggerSource

Trigger input source pin number.

- **Parameter type:** int
- **Range** : 0-3. Default is 1. Disabled is 0

9.8.3.15 #define PARAM_TRIGGER_ZERO_SOURCE "trigger_zero_source"

C# name: SensorParameter.TriggerZeroSource

Trigger zero input source pin number.

- **Parameter type:** int
- **Range** : 0-3. Default is 3. Disabled is 0.

9.8.3.16 #define PARAM_Y_TOLERANCE_MAX "y_tolerance_max"

C# name: SensorParameter.YToleranceMax

Upper threshold for height indicator LED.

- **Parameter type:** int
- **Unit** : Pixels
- **Range** : Default value is 800.

9.8.3.17 #define PARAM_Y_TOLERANCE_MIN "y_tolerance_min"

C# name: SensorParameter.YToleranceMin

Lower threshold for height indicator LED.

- **Parameter type:** int
- **Unit** : Pixels
- **Range** : Default value is 200.

9.8.3.18

9.9 Parameters for Filtering

9.9.1 Macros

- #define PARAM_DETECT_MISSING_FIRST_LAYER "detect_missing_first_layer"
- #define PARAM_DETECT_MISSING_FIRST_LAYERX "detect_missing_first_layer_x"
- #define PARAM_DETECT_MISSING_FIRST_LAYER_MIN_LENGTH "detect_missing_first_layer_min_length"
- #define PARAM_RESAMPLE_LINE_X_RESOLUTION "resample_line_x_resolution"
- #define PARAM_FILL_GAP_X_MAX "fill_gap_x_max"
- #define PARAM_AVERAGE_Z_FILTER_SIZE "average_z_filter_size"
- #define PARAM_AVERAGE_INTENSITY_FILTER_SIZE "average_intensity_filter_size"
- #define PARAM_MEDIAN_Z_FILTER_SIZE "median_z_filter_size"
- #define PARAM_MEDIAN_INTENSITY_FILTER_SIZE "median_intensity_filter_size"
- #define PARAM_NOISE_REMOVAL "noise_removal"
- #define PARAM_PEAK_X_FILTER "peak_x_filter"
- #define PARAM_TRIM_EDGES "trim_edges"
- #define PARAM_CLUSTERING_DISTANCE_X "clustering_distance_x"
- #define PARAM_CLUSTERING_DISTANCE_Z "clustering_distance_z"
- #define PARAM_CLUSTER_MINIMUM_LENGTH "cluster_minimum_length"

9.9.2 Detailed Description

9.9.3 Macro Definition Documentation

#define PARAM_AVERAGE_INTENSITY_FILTER_SIZE "average_intensity_filter_size"

C# name: SensorParameter.AverageIntensityFilterSize

Size of the averaging filter for the line callback intensity values.

- **Parameter type:** float
- **Unit** : Micrometers

#define PARAM_AVERAGE_Z_FILTER_SIZE "average_z_filter_size"

C# name: SensorParameter.AverageZFilterSize

Size of the averaging filter for the line callback Z-values.

- **Parameter type:** float
- **Unit** : Micrometers

#define PARAM_CLUSTER_MINIMUM_LENGTH "cluster_minimum_length"

C# name: SensorParameter.ClusterMinimumLength

Clusters which are shorter than specified length are removed. 0 if feature is disabled.

- **Parameter type:** float
- **Unit** : Micrometers

#define PARAM_CLUSTERING_DISTANCE_X "clustering_distance_x"

C# name: SensorParameter.ClusteringDistanceX

Maximum X distance between two points in a cluster. 0 if feature is disabled.

- **Parameter type:** float
- **Unit** : Micrometers

#define PARAM_CLUSTERING_DISTANCE_Z "clustering_distance_z"

C# name: SensorParameter.ClusteringDistanceZ

Maximum Z distance between two points in a cluster. 0 if feature is disabled.

- **Parameter type:** float
- **Unit** : Micrometers

#define PARAM_DETECT_MISSING_FIRST_LAYER "detect_missing_first_layer"

C# name: SensorParameter.DetectMissingFirstLayer

Correct the first surface by detecting areas where the next layer is visible but the first one missing (i.e. holes in the first layer). Takes effect for line callback functions. 0, disabled (default) >1, enabled, minimum distance between two layers in micrometers (typically 30 um is used)

- **Parameter type:** float

#define PARAM_DETECT_MISSING_FIRST_LAYER_MIN_LENGTH "detect_missing_first_layer_min_length"

C# name: SensorParameter.DetectMissingFirstLayerLength

Defines minimum length of the first surface. Can be used to reduce wrong detection of noise points to first layer. 0, disabled (default) >1, enabled, minimum length in micrometers

- **Parameter type:** float

#define PARAM_DETECT_MISSING_FIRST_LAYERX "detect_missing_first_layer_x"

C# name: SensorParameter.DetectMissingFirstLayerX

Correct the first surface by detecting areas where the next layer is visible but the first one missing (i.e. holes in the first layer). Takes effect for line callback functions and only if PARAM_DETECT_MISSING_FIRST_LAYER is set. 0, disabled (default) >1, enabled, maximum x-coordinate distance between two points in micrometers

- **Parameter type:** float

#define PARAM_FILL_GAP_X_MAX "fill_gap_x_max"

C# name: SensorParameter.FillGapXmax

Defines maximum gap of missing measurement points filled by interpolation. PARAM_LAYER_MIN_THICKNESS parameter defines maximum Z difference for interpolated points. For opaque materials, a user can set parameter for the first layer.

- **Parameter type:** float
- **Unit** : Micrometers, 0 if feature is disabled.

#define PARAM_MEDIAN_INTENSITY_FILTER_SIZE "median_intensity_filter_size"

C# name: SensorParameter.MedianIntensityFilterSize

Size of the median filter for the line callback intensity values. If an even value is specified, the function adds 1 and uses the odd value of the filter mask for median filtering.

- **Parameter type:** int
- **Unit** : Pixels

#define PARAM_MEDIAN_Z_FILTER_SIZE "median_z_filter_size"

C# name: SensorParameter.MedianZFilterSize

Size of the median filter for the line callback intensity values. If an even value is specified, the function adds 1 and uses the odd value of the filter mask for median filtering.

- **Parameter type:** int
- **Unit** : Pixels

#define PARAM_NOISE_REMOVAL "noise_removal"

C# name: SensorParameter.NoiseRemoval

Noise removal algorithm for the line callback. Noise filter suppresses the noise and allows to use lower threshold which improves the data acquisition.

Noise filter removes single data points which do not have adjacent data points. Note! If noise removal for Y is enabled line callback has 1 profile delay

- **Parameter type:** int
- **Values** : 0 = noise removal is disabled, 1 = noise removal is enabled for X, 2 = noise removal is enabled for Y, 3 = noise removal is enabled for X and Y.

#define PARAM_PEAK_X_FILTER "peak_x_filter"

C# name: `SensorParameter.PeakXFilter`

Set X-filter length to smooth the image before the peak detection. X-filter suppresses noise effectively, which allows to use of lower threshold. X-filter functionality can be used to smooth image in X direction before the peak detection.Valid values are 0 (disabled), 3, 5 or 7 pixels. Filter function averages (moving average) the raw image signal in X direction using the given number of pixels.

- **Parameter type:** int
- **Values** : Default value is 0 = disabled. Eligible values are 3, 5 and 7.

#define PARAM_RESAMPLE_LINE_X_RESOLUTION "resample_line_x_resolution"

C# name: `SensorParameter.ResampleLineXResolution`

Change the resolution for the profile given in the line callback function. 0, disabled (default) >0, new resolution in micrometers. Must be bigger than the camera native resolution (i.e. for LCI1200 this must be bigger than 5.5 micrometers)

- **Parameter type:** float

#define PARAM_TRIM_EDGES "trim_edges"

C# name: `SensorParameter.TrimEdges`

Edge filter to remove artificial points detected at the end of the surfaces. The filter can be disabled/enabled at runtime in the line callback function and it takes effect on the next processed profiles.

- **Parameter type:** int
- **Values** : 0 = filter is disabled (default), 1 = filter is enabled.

9.10 Deprecated Parameters

9.10.1 Macros

- #define PARAM_PEAK_AVER_FIR_LENGTH "peak_aver_fir_length"
- #define PARAM_IMAGE_WIDTH "image_width"
- #define PARAM_IMAGE_OFFSETX "image_offsetx"
- #define PARAM_INTERLEAVE_FACTOR "interleavefactor"
- #define PARAM_INTERLEAVE_FILTER "interleavefilter"
- #define PARAM_INTERLEAVE_TARGET_INTENSITY "interleavetargetintensity"

- #define PARAM_INTERLEAVE_Y_THRESHOLD "interleaveythreshold"
- #define PARAM_LED_DURATION "led_duration"
- #define PARAM_FLIP_X_X0 "flip_x_x0"
- #define PARAM_FLIP_X_ENABLED "flip_x_enabled"
- #define PARAM_FLIP_Z_Z0 "flip_z_z0"
- #define PARAM_FLIP_Z_ENABLED "flip_z_enabled"

9.10.2 Detailed Description

9.10.3 Macro Definition Documentation

#define PARAM_FLIP_X_ENABLED "flip_x_enabled"

C# name: SensorParameter.FlipXEnabled

Determines if flip x is enabled or not. See PARAM_FLIP_X_X0 on how to define X0.

- **Parameter type:** int
- **Unit** : 1 enabled, 0 disabled.

#define PARAM_FLIP_X_X0 "flip_x_x0"

C# name: SensorParameter.FlipX0

Set the X0[um] of flip x. If PARAM_FLIP_X_ENABLED is enabled, the X[um] of the points (X[um], Z[um]) are flipped with respect to X0.

- **Parameter type:** float
- **Unit** : Micrometers

#define PARAM_FLIP_Z_ENABLED "flip_z_enabled"

C# name: SensorParameter.FlipZEnabled

Determines if flip z is enabled or not. See PARAM_FLIP_Z_Z0 on how to define Z0.

- **Parameter type:** int
- **Unit** : 1 enabled, 0 disabled.

#define PARAM_FLIP_Z_Z0 "flip_z_z0"

C# name: SensorParameter.FlipZ0

Set the Z0[um] of flip z. If PARAM_FLIP_Z_ENABLED is enabled, the Z[um] of the points (X[um], Z[um]) are flipped with respect to Z0.

- **Parameter type:** float
- **Unit** : Micrometers

#define PARAM_IMAGE_OFFSETX "image_offsetx"

C# name: SensorParameter.OffsetX

Set the X offset for the image capture.

- **Parameter type:** Int.

- **Unit** : Pixels
- **Values** : Typically 0. If value is non-zero, it should be noticed that the sum of offset X and image width shall not exceed the sensor width.

#define PARAM_IMAGE_WIDTH "image_width"

C# name: SensorParameter.Width

Width of the sensor image in pixel unit.

- **Parameter type:** Int.
- **Unit** : Pixels
- **Values** : For full camera sensor 2048.

#define PARAM_INTERLEAVE_FACTOR "interleavefactor"

C# name: SensorParameter.InterleaveFactor

Sets the value used to enable or disable interleaving. Interleaving applies different exposure for even and odd rows of the sensor. The exposure for odd rows is as-is, whereas even rows have short exposure, namely current_exposure / PARAM_INTERLEAVE_FACTOR.

- **Parameter type:** float
- **Values** : 1.0 disables interleaving. Any other value divides the exposure of even rows, but leaves the exposure of odd rows as-is.

#define PARAM_INTERLEAVE_FILTER "interleavefilter"

C# name: SensorParameter.InterleaveFilterMode

Sets the interleaving filter. Used only if PARAM_INTERLEAVE_FACTOR is other than 1.0.

- **Parameter type:** int
- **Values** : 0 = do not filter (the application should filter), 1 = filter using PARAM_INTERLEAVE_TARGET_INTENSITY and PARAM_INTERLEAVE_Y_THRESHOLD

#define PARAM_INTERLEAVE_TARGET_INTENSITY "interleavetargetintensity"

C# name: SensorParameter.InterleaveFilterTargetIntensity

Sets the target intensity for the interleaving. See PARAM_INTERLEAVE_FILTER for more information.

- **Parameter type:** float
- **Values** : 8-bit grayscale intensity (0.0 - 255.0)

#define PARAM_INTERLEAVE_Y_THRESHOLD "interleaveythreshold"

C# name: SensorParameter.InterleaveFilterYThreshold

Sets the Y threshold [px] for interleaving. See PARAM_INTERLEAVE_FILTER for more information.

- **Parameter type:** float
- **Values** : Y threshold [px] value.

#define PARAM_LED_DURATION "led_duration"

C# name: SensorParameter.LedDuration

The time which the LED is ON. LED cannot be ON all the time between two trigger pulses (PARAM_REG_PULSE_FREQ). To achieve this see chapter LED Pulse Duration.

- **Parameter type:** int
- **Unit:** Microseconds
- **Values** : Values should be greater than 1.

#define PARAM_PEAK_AVER_FIR_LENGTH "peak_aver_fir_length"

C# name: SensorParameter.FirAverLength

The length of averaging filter used for both detection and intensity calculation. Parameter is deprecated. Please use parameters PARAM_SIGNAL_DETECTION_FILTER_LENGTH and PARAM_PEAK_AVERAGE_INTENSITY_FILTER_LENGTH.

- **Parameter type:** int
- **Values** : Eligible values are 2, 4, 6, 8, 10, 12, 14 or 16.

10 Class Documentation

10.1 DynSensorControl Struct Reference

Stucture hold dynamic sensor control parameters.

10.1.1 Public Attributes

- uint32_t dyn_ctrl_location
  Encoder location position.

- uint32_t dyn_ctrl_height
  Sensor image height.

- uint32_t dyn_ctrl_offset
  Sensor image offset.

- float dyn_ctrl_pulse_width
  Illuminator pulse width in microseconds.

- uint32_t dyn_ctrl_threshold
  Sensor image 8-bit gray scale intensity value (0-255) threshold. Supported by LCI1220 and LCI1620 sensors.

10.1.2 Detailed Description

Stucture hold dynamic sensor control parameters.

10.1.3 Member Data Documentation

uint32_t DynSensorControl::dyn_ctrl_height

Sensor image height.

uint32_t DynSensorControl::dyn_ctrl_location

Encoder location position.

uint32_t DynSensorControl::dyn_ctrl_offset

Sensor image offset.

float DynSensorControl::dyn_ctrl_pulse_width

Illuminator pulse width in microseconds.

uint32_t DynSensorControl::dyn_ctrl_threshold

Sensor image 8-bit gray scale intensity value (0-255) threshold. Supported by LCI1220 and LCI1620 sensors.

10.2 FieldCalibrationResults Struct Reference

10.2.1 Public Attributes

- double averageStd
  Average standard deviation of sequential measurements. Indicates noise of the camera.

- double maximumStd
  Maximum standard deviation of sequential measurements. Indicates noise of the camera.

- double minimumStd
  Minimum standard deviation of sequential measurements. Indicates noise of the camera.

- double averageIntensity
  Average intensity of measured profiles.

- double usedPulseLength
  Used pulse length for measured profiles.

- uint64_t calibrationBlock
  Detected calibration block (0 = not detected, 1 = mirror, 2 = groove)

- double averageAbsoluteDeviation
  Average flatness correction calculated from the calibration block

- double maximumAbsoluteDeviation
  Maximum flatness correction calculated from the calibration block

- double grooveMeasurements[MAX_GROOVE_MEASUREMENTS]
  Measured groove heights in micrometers.

- uint64_t numOfGrooveMeasurements
  Number of measured groove heights.

- double calculatedGrooveCorrection
  Calculated z-correction.

- double profileZValues[2048]
  Measured profile. Array of z-values.

- double profileStdDevValues[2048]
  Stdev for each x-coordinate in the profile.

10.2.2 Member Data Documentation

double FieldCalibrationResults::averageAbsoluteDeviation

Average flatness correction calculated from the calibration block

double FieldCalibrationResults::averageIntensity

Average intensity of measured profiles.

double FieldCalibrationResults::averageStd

Average standard deviation of sequential measurements. Indicates noise of the camera.

double FieldCalibrationResults::calculatedGrooveCorrection

Calculated z-correction.

uint64_t FieldCalibrationResults::calibrationBlock

Detected calibration block (0 = not detected, 1 = mirror, 2 = groove)

double FieldCalibrationResults::grooveMeasurements[MAX_GROOVE_MEASUREMENTS]

Measured groove heights in micrometers.

double FieldCalibrationResults::maximumAbsoluteDeviation

Maximum flatness correction calculated from the calibration block

double FieldCalibrationResults::maximumStd

Maximum standard deviation of sequential measurements. Indicates noise of the camera.

double FieldCalibrationResults::minimumStd

Minimum standard deviation of sequential measurements. Indicates noise of the camera.

uint64_t FieldCalibrationResults::numOfGrooveMeasurements

Number of measured groove heights.

double FieldCalibrationResults::profileStdDevValues[2048]

Stdev for each x-coordinate in the profile.

double FieldCalibrationResults::profileZValues[2048]

Measured profile. Array of z-values.

double FieldCalibrationResults::usedPulseLength

Used pulse length for measured profiles.

10.3 HEADER Struct Reference

Header associated to each raw image or profile received from the camera.

10.3.1 Public Attributes

- uint64_t enabled_fields
  *Bitmask telling, which **optional** fields of this struct are set. To check if an optional field is set, you can use BIT_IS_SET and bit defines such as FS_HEADER_PULSE_WIDTH_BIT.*

- uint32_t reception_queue_size
  Number of point clouds in a queue inside API. If this value starts to accumulate, it is usually a sign of too slow application point cloud callback handler. Always set.

- union {
- double reception_frequency
  Frequency (in Hz) at which the API receives data. Set only if frequency calculation is enabled (default).

- unsigned long long time_stamp
  Time stamp (in 0.1 us) at which the API receives data. Set only if frequency calculation is disabled.

- };
- uint32_t index
  Zero-based point cloud reception index. Always set.

- uint8_t input_state
  *HW input pin state bitmask: |b0|b1|b2|b3|b4|b5|b6|b7|. **Optional.***

- float pulse_width
  us pulse width of the frame. ***Optional.***

- int32_t location
  Generic location. ***Optional.***

10.3.2 Detailed Description

Header associated to each raw image or profile received from the camera.

10.3.3 Member Data Documentation

union { ... }
uint64_t HEADER::enabled_fields

Bitmask telling, which **optional** fields of this struct are set. To check if an optional field is set, you can use BIT_IS_SET and bit defines such as FS_HEADER_PULSE_WIDTH_BIT.

uint32_t HEADER::index

Zero-based point cloud reception index. Always set.

uint8_t HEADER::input_state

HW input pin state bitmask: | b0 | b1 | b2 | b3 | b4 | b5 | b6 | b7 |. **Optional.**

int32_t HEADER::location

Generic location. **Optional.**

float HEADER::pulse_width

us pulse width of the frame. **Optional.**

double HEADER::reception_frequency

Frequency (in Hz) at which the API receives data. Set only if frequency calculation is enabled (default).

uint32_t HEADER::reception_queue_size

Number of point clouds in a queue inside API. If this value starts to accumulate, it is usually a sign of too slow application point cloud callback handler. Always set.

unsigned long long HEADER::time_stamp

Time stamp (in 0.1 us) at which the API receives data. Set only if frequency calculation is disabled.

10.4 Point2D Struct Reference

10.4.1 Public Attributes

- double x
- double y

10.4.2 Member Data Documentation

double Point2D::x
double Point2D::y

10.5 Point3D Struct Reference

10.5.1 Public Attributes

- double x
- double y
- double z

10.5.2 Member Data Documentation

10.5.2.1 double Point3D::x

10.5.2.2 double Point3D::y

10.5.2.3 double Point3D::z

10.5.2.4

10.6 STRUCT_POINT Struct Reference

Defines a point received by a callback handler for (ProfileCallback).

10.6.1 Public Attributes

- float X
  The x coordinate.

- float Y
  The y coordinate.

- float Intensity
  The intensity.

- union {
- float IntensitySecondary
  Intensity calculated with the detection filter. Set only if the feature is supported by the firmware.

- InterleavePhase interleavePhase
  Value indicates the exposure phase when using interleaving. If the interleaving is not used, point phase is set to long.

- };

10.6.2 Detailed Description

Defines a point received by a callback handler for (ProfileCallback).

10.6.3 Member Data Documentation

union { ... }
float STRUCT_POINT::Intensity

The intensity.

float STRUCT_POINT::IntensitySecondary

Intensity calculated with the detection filter. Set only if the feature is supported by the firmware.

InterleavePhase STRUCT_POINT::interleavePhase

Value indicates the exposure phase when using interleaving. If the interleaving is not used, point phase is set to long.

float STRUCT_POINT::X

The x coordinate.

float STRUCT_POINT::Y

The y coordinate.