

Xavia SDK c++ API

2.2.1

1 Xavia SDK for C++

Version: 2.0 Date: 24/12/2025

Contents:

- 1. Introduction
- 2. Basic example
- 3. Available data
- 4. Compatibility

1.1 1. Introduction

The Xavia SDK is a C++ library to interact with the Xavia sensor. The SDK provides a set of objects that allow developers to easily integrate these sensors into their applications.

With this SDK you will be able to:

- Start and stop the sensor from code.
- Receive point cloud data in your code.
- Store and read the point cloud data in the XenomatiX format.
- Receive errors and status changes from the sensor.
- Control multiple sensors from the same code.

Main features:

- Created with C++17 and modern C++ design principles.
- Fully object oriented.
- Easy to use. You can get started in 10 lines of code.
- Tested for performance.
- Fully unit tested.

1.2 2. Basic example

Since a picture is worth more than 1000 words, here is a basic point cloud recorder that shows how to use the Xavia SDK:

```
 {.numberLines}
#include "xavia/sensorFactory.h"
#include "xavia/iSensor.h"
#include "xavia/iPointCloud.h"

#include <thread>
#include <chrono>
#include <fstream>

namespace xsdk = xavia::sdk;

void main()
{
    xsdk::SensorFactory factory;

    // prepare the point cloud recording
    std::ofstream pointCloudFile;
    pointCloudFile.open("pcFile.xpc", std::ios_base::binary);
    auto PointCloudCallback = [&](const std::unique_ptr<xsdk::IPointCloud> pc)
      {
        std::vector<std::byte> pcData = pc->Serialize();
        pointCloudFile.write(pcData.data(), pcData.size());
      };

    // register your point cloud handling
    factory.SetPointcloudCallback(PointCloudCallback);

    // build sensor
    auto sensor = factory.Build();

    //run
    sensor->Start();
    std::this_thread::sleep_for(std::chrono::seconds(60));
    sensor->Stop();

    // close the output file
    pointCloudFile.close();
}
```

This example shows the usage of the `Sensor` object to operate the status of the sensor during its lifetime. In this example only the `Start` and `Stop` functions are used, but you can use it to query different properties of the sensor.
The `SensorFactory` object is there to do the configuration before connecting to the sensor. In this example we only attached a point cloud callback function to handle the incoming data (which in this case we wrote as a lambda function). This callback will take the point cloud object, ask it to `Serialize` itself, and then writes those bytes to a file. Later, we could use the `IPointCloud::Deserialize()` method to read the point cloud from the buffers in the file and process them.
In a real time application, you would instead keep using the `xavia::sdk::IPointCloud` object to extract e.g. the measured distance or the xyz coordinates of the point cloud.
The `SensorFactory::Build()` function is overloaded to take several arguments that allow you to indicate which sensor you want to connect to. In this way, it is possible to configure the factory once and then use it to build multiple sensor objects, each connected to a different lidar in your network. In line with modern c++, this build method returns objects of type `std::unique_ptr<xavia::sdk::isensor>`, meaning that you will take full control over the lifetime of the sensor object. Proper RAII implementations will ensure that the sensor object is properly destroyed when it goes out of scope.
This example is a minimal software to record data. In a production environment you should add more robustness by also handling errors (which are all passed as `std::exception` derivatives).

1.3 3. Available data

Through the SDK, the following point cloud information is available:

- **XYZ coordinates** of the points, represented in mm, in the coordinate system of the sensor.
- **Measured distance** in mm from the sensor origin.
- **Reflectivity measurement**. It represents the amount of reflected laser light.
- **Ambient measurement**. I represents the amount of light that is not due to the lasers.

Romeinse straat 6, 3001 Heverlee, België
Tel: +32 16 40 64 34
www.xenomatix.com

- **Confidence**. A confidence value indicating the quality of the measurement.

This information is provided in a vectorized format per topic, grouped per row. This information is transmitted row per row at the measurement frequency, which is around 1120Hz.
Besides the point cloud data itself, every line is accompanied by 'metadata'. This includes:

- Timestamp of the start of the frame.
- Timestamp of the start of this line. (there are 56 lines in a frame in the default configuration)
- The frame number. An monotonic incrementing number for each frame recorded by the sensor.
- The line number. An identifier for each line within a frame.
- The return index. In future versions the sensor will be able to support multiple returns.
- The trigger counter. A monotonic incrementing number for each trigger received by the sensor. Not every trigger will result in a frame if the triggering frequency is higher than the measurement frequency.
- Relative humidity measured inside the sensor housing
- Temperature measured inside the sensor housing

Furthermore, the sensor includes an onboard IMU. In future releases this information will also become available:

- Angular velocity of yaw, pitch and roll (in radian per second)
- Linear acceleration of x, y and z (in m/s^2)

1.4 4. Compatibility

The SDK is written in C++17 and compiled using MSVC143 and G++ 9.3.0. The target platforms are:

- x86_64 Windows
- x86_64 Linux
- arm64 Linux

The SDK is supplied in the form of a precompiled library (.dll and .lib, or .a and .so files), together with the necessary header and documentation files. It is compatible with cmake.

2 Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

exception

Exception 4

ICamera 5

IImage 8

ImageMetaData 10

ImuData 11

IPointCloud 11

ISensor 16

MetaData 19

OnboardProbes 20

SensorFactory 20

SerialNumber 24

3 Class Index

3.1 Class List

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

Exception
Exception base class for the Xavia SDK exceptions 4

ICamera
Camera object interface 5

IImage
The interface of the image object 8

ImageMetaData
Metadata struct containing general information for each camera image 10

ImuData
The IMU data registered when the point cloud was recorded 11

IPointCloud
The interface of the point cloud object 11

ISensor
Sensor object interface 16

MetaData
Metadata struct containing general information for each point cloud line 19

OnboardProbes
Struct containing information from sensors inside the sensors such as temperature and humidity 20

SensorFactory
Factory object, to configure and construct sensor objects 20

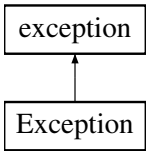
SerialNumber
Utility class that holds a serial number and allows interpretation 24

4 Class Documentation

4.1 Exception Class Reference

Exception base class for the Xavia SDK exceptions.
Inheritance diagram for Exception:

Romeinse straat 6, 3001 Heverlee, België
Tel: +32 16 40 64 34
www.xenomatix.com



Public Member Functions

- SensorError GetType () const

  Get the type of the exception.
- const char * **what** () const noexcept override

  return traditional, human readable version of the exception.

4.1.1 Detailed Description

Exception base class for the Xavia SDK exceptions.
Extends the standard exception with the 'GetType()' method. Use that method to understand which type of exception is thrown.

4.1.2 Member Function Documentation

GetType()

```
SensorError GetType () const
```

Get the type of the exception.

Returns

Enum indicating exception type.

See also

SensorError.

4.2 ICamera Class Reference

Camera object interface.

Public Member Functions

- virtual std::string GetCameraId () const =0

  Get the camera identifier.
- virtual std::wstring GetCameraIdW () const =0

  Get the camera identifier, wide format.
- virtual std::uint32_t GetExposureTimeUs () const =0

  Get the current camera exposure time.
- virtual bool HasWriteAccess ()=0

  Check if we have write access to the camera configuration.
- virtual bool IsStreaming () const =0

  Check if camera is currently streaming.
- virtual void ResetConfiguration ()=0

  Reset camera configuration to default (factory) values.
- virtual void SetExposureTimeUs (const std::uint32_t exposureTimeUs)=0

  Set the camera exposure time.
- virtual void Start (const std::size_t &timeoutMS=DEFAULT_CAMERA_TIMEOUT_MS)=0

  Starts the camera streaming.
- virtual void Stop (const std::size_t &timeoutMS=DEFAULT_CAMERA_TIMEOUT_MS)=0

  Stop the camera streaming.

4.2.1 Detailed Description

Camera object interface.
This class defines the interface for a camera. It provides methods to start, stop, and control the camera independently from the lidar sensor.
Access the camera via ISensor::GetCamera() which returns nullptr if no camera was configured during sensor build.

See also

ISensor::GetCamera()

SensorFactory for configuring camera during sensor creation

4.2.2 Member Function Documentation

GetCameraId()

```
virtual std::string GetCameraId () const  [pure virtual]
```
Get the camera identifier.

Returns

String identifying the camera (e.g., serial number, model)

Returns a string that uniquely identifies this camera, typically the serial number or model name from the camera hardware.

GetCameraIdW()

```
virtual std::wstring GetCameraIdW () const  [pure virtual]
```
Get the camera identifier, wide format.

Returns

String identifying the camera (e.g., serial number, model)

Returns a string that uniquely identifies this camera, typically the serial number or model name from the camera hardware.

GetExposureTimeUs()

```
virtual std::uint32_t GetExposureTimeUs () const  [pure virtual]
```
Get the current camera exposure time.

Returns

Current exposure time in microseconds

Returns the currently configured exposure time. This reflects the value set by SetExposureTimeUs or the default if never changed.

HasWriteAccess()

```
virtual bool HasWriteAccess ()  [pure virtual]
```
Check if we have write access to the camera configuration.

Returns

true if we have write access, false if not.

Without write access the SDK cannot change any camera settings or start and stop the stream. Write access is provided only to the first client that connects to the camera.

Romeinse straat 6, 3001 Heverlee, België
Tel: +32 16 40 64 34
www.xenomatix.com

IsStreaming()

```
virtual bool IsStreaming () const  [pure virtual]
```

Check if camera is currently streaming.

Returns

true if camera is streaming, false otherwise

ResetConfiguration()

```
virtual void ResetConfiguration ()  [pure virtual]
```

Reset camera configuration to default (factory) values.
Warning, this will overwrite any configuration settings for: Exposure, gain, triggering, framerate, PixelFormat

SetExposureTimeUs()

```
virtual void SetExposureTimeUs (
            const std::uint32_t exposureTimeUs)  [pure virtual]
```

Set the camera exposure time.

Parameters

exposureTimeUs	Exposure time in microseconds

Exceptions

sdk::exception	object if exposure time cannot be set or is out of valid range.

Sets the exposure time for the camera. The camera will use this exposure time for all subsequent frames. This can be called while the camera is streaming.
Valid range depends on camera hardware. Typical range: 20us - 1000000us (1 second).

Start()

```
virtual void Start (
            const std::size_t & timeoutMS = DEFAULT_CAMERA_TIMEOUT_MS)  [pure virtual]
```

Starts the camera streaming.

Parameters

timeoutMS	Timeout in milliseconds for camera to start (optional, default is 5 sec)

Exceptions

sdk::exception	object if camera fails to start within timeout.

Start means that the camera begins streaming images which will be delivered via the image callback registered on SensorFactory.
It is only valid to call this function if the camera is not already streaming. An exception is thrown if the camera fails to start.

Stop()

```
virtual void Stop (
            const std::size_t & timeoutMS = DEFAULT_CAMERA_TIMEOUT_MS)  [pure virtual]
```

Stop the camera streaming.

Parameters

timeoutMS	Timeout in milliseconds for camera to stop (optional, default is 5 sec)

Exceptions

sdk::exception	object if camera fails to stop within timeout.

Stop means that the camera stops streaming images and no further callbacks will occur.
It is only valid to call this function if the camera is currently streaming. An exception is thrown if the camera fails to stop.

4.3 IImage Class Reference

The interface of the image object.

Public Member Functions

- virtual void Deserialize (const std::uint8_t *pBuffer, const std::size_t size)=0

  Fill this image object from binary data.
- virtual void Deserialize (const std::vector< std::uint8_t > &data)=0

  Fill this image object from binary data.
- virtual std::uint32_t GetHeight () const =0

  Get the height of the image.
- virtual const ImageMetaData & GetMetaData () const =0

  Get the metadata of the current image.
- virtual const std::vector< std::uint8_t > & GetPixelData () const =0

  Get the raw pixel data.
- virtual xavia::sdk::PixelFormat GetPixelFormat () const =0

  Get the pixel format.
- virtual std::uint32_t GetWidth () const =0

  Get the width of the image.
- virtual void SaveAsJpeg (const std::wstring &filePath, const int quality=90, const bool chromaticSubsampling=false) const =0

  Save the image as an 8 bit jpeg file.
- virtual std::vector< std::uint8_t > Serialize ()=0

  Convert this image object into binary data.

4.3.1 Detailed Description

The interface of the image object.
Returned by the image callback, each image object represents a single frame from the camera.
The image object contains the pixel data, metadata such as timestamps and frame numbers, and dimensions.

4.3.2 Member Function Documentation

Deserialize() [1/2]

```
virtual void Deserialize (
            const std::uint8_t * pBuffer,
            const std::size_t size)  [pure virtual]
```

Fill this image object from binary data.

Parameters

in	*pBuffer*	pointer to the start of the binary data.
in	*size*	number of bytes to read from the buffer.

Use this to efficiently load the image from the XenomatiX proprietary format. The opposite of the Serialize() function.

Note

although it is acceptable to provide too many bytes, it is best that size matches exactly the required size. This function will keep an internal copy of the provided data.

Deserialize() [2/2]

```
virtual void Deserialize (
            const std::vector< std::uint8_t > & data)  [pure virtual]
```

Fill this image object from binary data.

Parameters

in	*data*	vector of bytes that hold the image information.

Use this to efficiently load the image from the XenomatiX proprietary format. The opposite of the Serialize() function.

Note

although it is acceptable to provide too many bytes, it is best that size matches exactly the required size. This function will keep an internal copy of the provided data.

GetHeight()

```
virtual std::uint32_t GetHeight () const  [pure virtual]
```

Get the height of the image.

Returns

Height in pixels.

GetMetaData()

```
virtual const ImageMetaData & GetMetaData () const  [pure virtual]
```

Get the metadata of the current image.

Returns

const reference to the metadata struct,

See also

ImageMetaData.

GetPixelData()

```
virtual const std::vector< std::uint8_t > & GetPixelData () const  [pure virtual]
```

Get the raw pixel data.

Returns

const reference to the vector of pixel data. The size and interpretation depend on width, height, and pixel format.

GetPixelFormat()

```
virtual xavia::sdk::PixelFormat GetPixelFormat () const  [pure virtual]
```
Get the pixel format.

Returns

Pixel format identifier (0=RGB8, 1=BGR8, 2=Mono8, etc.).

GetWidth()

```
virtual std::uint32_t GetWidth () const  [pure virtual]
```
Get the width of the image.

Returns

Width in pixels.

SaveAsJpeg()

```
virtual void SaveAsJpeg (
            const std::wstring & filePath,
            const int quality = 90,
            const bool chromaticSubsampling = false) const  [pure virtual]
```
Save the image as an 8 bit jpeg file.

Parameters

in	*filePath*	path to the output jpeg file.
in	*quality*	JPEG quality (0-100) (default 90).
in	*chromaticSubsampling*	Enable or disable chromatic subsampling (default false).

JPEG saving is only supported for RGB8 format. If the image is in Bayer format, it will be demosaiced before saving. If the image was converted to RGB12, it will be scaled down to 8 bits per channel.

Serialize()

```
virtual std::vector< std::uint8_t > Serialize ()  [pure virtual]
```
Convert this image object into binary data.

Returns

vector of bytes that hold the image information

Use this to efficiently store the image in the XenomatiX proprietary format. When dumping to disk, it is customary to use the .ximg extension.

See also

Deserialize() functions to read the information back.

4.4 ImageMetaData Struct Reference

Metadata struct containing general information for each camera image.

Public Attributes

- std::uint32_t **frameNumber** {0}

  Monotonically increasing counter. Unique for each frame acquired by the camera.
- std::uint64_t **frameTimeStampUs** {0}

Romeinse straat 6, 3001 Heverlee, België
Tel: +32 16 40 64 34
www.xenomatix.com

Posix timestamp in us of frame acquisition.

- std::uint32_t **height** {0}

  Image height in pixels.
- xavia::sdk::PixelFormat **pixelFormat** {xavia::sdk::PixelFormat::Unknown}

  Pixel format identifier.
- std::uint64_t **receivedTimeStampUs** {0}

  Posix timestamp in us when data is received by the SDK.
- std::uint32_t **width** {0}

  Image width in pixels.

4.4.1 Detailed Description

Metadata struct containing general information for each camera image.
This struct contains information such as frame numbers, timestamps, and image dimensions.

4.5 ImuData Struct Reference

The IMU data registered when the point cloud was recorded.

Public Attributes

- float **accXMS2**

  x acceleration expressed in meter per seconds squared.
- float **accYMS2**

  y acceleration expressed in meter per seconds squared.
- float **accZMS2**

  z acceleration expressed in meter per seconds squared.
- float **pitchDegS**

  pitch change indicated in degrees per second
- float **rollDegS**

  roll change indicated in degrees per second
- float **yawDegS**

  yaw change indicated in degrees per second

4.5.1 Detailed Description

The IMU data registered when the point cloud was recorded.

4.6 IPointCloud Class Reference

The interface of the point cloud object.

Public Member Functions

- virtual void Deserialize (const std::uint8_t *pBuffer, const std::size_t size)=0

  Fill this point cloud object from binary data.
- virtual void Deserialize (const std::vector< std::uint8_t > &data)=0

  Fill this point cloud object from binary data.
- virtual const std::vector< std::uint16_t > & GetAmbient () const =0

  Get the background light of points of the current line.
- virtual const std::vector< rgb > & GetColors () const =0

  Get the rgb color values for each point.
- virtual const std::vector< float > & GetConfidence () const =0

  Get the confidence of points of the current line.

- virtual const std::vector< float > & GetDistanceMm () const =0

  Get the measured distance of the points of the current line.
- virtual std::uint64_t GetFrameTimestampUs () const =0

  Get the frame timestamp associated with this dataset.
- virtual const ImuData & GetImuData () const =0

  Get the IMU data of the current line.
- virtual std::uint64_t GetLineTimestampUs () const =0

  Get the line timestamp associated with this dataset.
- virtual const MetaData & GetMetaData () const =0

  Get the metadata of the current line.
- virtual std::size_t GetNrPoints () const =0

  Get the number of points in the current line.
- virtual const OnboardProbes & GetOnboardProbes () const =0

  Get the temperature and humidity of the current line.
- virtual const std::vector< float > & GetReflectivity () const =0

  Get the reflectivity of the points of the current line.
- virtual const std::vector< float > & GetXMm () const =0

  Get the x reconstruction of points of the current line.
- virtual const std::vector< float > & GetYMm () const =0

  Get the y reconstruction of points of the current line.
- virtual const std::vector< float > & GetZMm () const =0

  Get the z reconstruction of points of the current line.
- virtual const std::vector< std::uint8_t > & Serialize ()=0

  Convert this point cloud object into binary data.
- virtual void SetColors (const std::vector< rgb > &colors)=0

  Set the rgb color values for each point.

4.6.1 Detailed Description

The interface of the point cloud object.
Returned by the point cloud callback, each point cloud object represents a single line in the FOV of the sensor.
The point cloud object contains: the (x,y,z) reconstruction of each point, the measured distance from the sensor, the reflectivity and ambient light values, a confidence level for each point, and the metadata.

4.6.2 Member Function Documentation

Deserialize() [1/2]

```
virtual void Deserialize (
            const std::uint8_t * pBuffer,
            const std::size_t size)  [pure virtual]
```

Fill this point cloud object from binary data.

Parameters

in	*pBuffer*	pointer to the start of the binary data.
in	*size*	number of bytes to read from the buffer.

Use this to efficiently load the point cloud from the XenomatiX proprietary format. The opposite of the Serialize() function.

Note

although it is acceptable to provide too many bytes, it is best that size matches exactly the required size. This function will keep an internal copy of the provided data.

Deserialize() [2/2]

```
virtual void Deserialize (
            const std::vector< std::uint8_t > & data)  [pure virtual]
```

Fill this point cloud object from binary data.

Parameters

in	*data*	vector of bytes that hold the point cloud information.

Use this to efficiently load the point cloud from the XenomatiX proprietary format. The opposite of the Serialize() function.

Note

although it is acceptable to provide too many bytes, it is best that size matches exactly the required size. This function will keep an internal copy of the provided data.

GetAmbient()

```
virtual const std::vector< std::uint16_t > & GetAmbient () const  [pure virtual]
```

Get the background light of points of the current line.

Returns

const reference to the vector with background light measurements. This vector holds GetNrPoints() elements.

GetColors()

```
virtual const std::vector< rgb > & GetColors () const  [pure virtual]
```

Get the rgb color values for each point.

Returns

Reference to the vector containing rgb values .

If color information is present, the returned vector contains GetNrPoints() elements, each representing the color of a point.
If the point cloud does not contain color information, an empty vector is returned.

GetConfidence()

```
virtual const std::vector< float > & GetConfidence () const  [pure virtual]
```

Get the confidence of points of the current line.

Returns

const reference to the vector confidences (between 0 and 1, where 1 is 100% confidence). This vector holds GetNrPoints() elements.

GetDistanceMm()

```
virtual const std::vector< float > & GetDistanceMm () const  [pure virtual]
```

Get the measured distance of the points of the current line.

Returns

const reference to the vector of radial distances from the sensor in mm. This vector holds GetNrPoints() elements.

GetFrameTimestampUs()

```
virtual std::uint64_t GetFrameTimestampUs () const  [pure virtual]
```
Get the frame timestamp associated with this dataset.

Returns

Frame timestamp expressed in microseconds.

GetImuData()

```
virtual const ImuData & GetImuData () const  [pure virtual]
```
Get the IMU data of the current line.

Returns

const reference to the imuData struct,

See also

ImuData.

Note

ImuData is currently not yet released. Values will always be 0.

GetLineTimestampUs()

```
virtual std::uint64_t GetLineTimestampUs () const  [pure virtual]
```
Get the line timestamp associated with this dataset.

Returns

Line timestamp expressed in microseconds.

GetMetaData()

```
virtual const MetaData & GetMetaData () const  [pure virtual]
```
Get the metadata of the current line.

Returns

const reference to the metadata struct,

See also

MetaData.

GetNrPoints()

```
virtual std::size_t GetNrPoints () const  [pure virtual]
```
Get the number of points in the current line.

Returns

number of points as unsigned integer. e.g. 192. This is the number of elements inside the other vector returns of the pointcloud.

GetOnboardProbes()

```
virtual const OnboardProbes & GetOnboardProbes () const  [pure virtual]
```
Get the temperature and humidity of the current line.

Returns

const reference to the probes struct,

See also

OnboardProbes.

GetReflectivity()

```
virtual const std::vector< float > & GetReflectivity () const  [pure virtual]
```
Get the reflectivity of the points of the current line.

Returns

const reference to the vector reflectivity values. Expressed in % (can exceed 100%). This vector holds GetNrPoints() elements.

GetXMm()

```
virtual const std::vector< float > & GetXMm () const  [pure virtual]
```
Get the x reconstruction of points of the current line.

Returns

const reference to the vector of x coordinates in mm, with the sensor as origin. This vector holds GetNrPoints() elements.

GetYMm()

```
virtual const std::vector< float > & GetYMm () const  [pure virtual]
```
Get the y reconstruction of points of the current line.

Returns

const reference to the vector of y coordinates in mm, with the sensor as origin. This vector holds GetNrPoints() elements.

GetZMm()

```
virtual const std::vector< float > & GetZMm () const  [pure virtual]
```
Get the z reconstruction of points of the current line.

Returns

const reference to the vector of z coordinates in mm, with the sensor as origin. This vector holds GetNrPoints() elements.

Serialize()

```
virtual const std::vector< std::uint8_t > & Serialize ()  [pure virtual]
```
Convert this point cloud object into binary data.

Returns

vector of bytes that hold the point cloud information

Use this to efficiently store the point cloud in the XenomatiX proprietary format. When dumping to disk, it accustomary to use the .xpc extension.

See also

deserialize() functions to read the information back.

SetColors()

```
virtual void SetColors (
            const std::vector< rgb > & colors)  [pure virtual]
```

Set the rgb color values for each point.

Parameters

in	*colors*	rgb values, one per point.

The provided vector should contain GetNrPoints() elements.

4.7 ISensor Class Reference

Sensor object interface.

Public Member Functions

- virtual std::shared_ptr< ICamera > GetCamera () const =0

  Get the camera associated with this sensor.
- virtual SensorState GetCurrentState () const =0

  Provides the current state of the sensor (running, idle, error).
- virtual std::vector< SensorError > GetErrors (const std::size_t &timeoutMS=DEFAULT_STATE_CHANGE↩ _TIMEOUT_MS)=0

  Retrieves the state from the sensor and reports a list of errors.
- virtual SerialNumber GetSerialNumber () const =0

  Provides the serial number of the sensor from local memory.
- virtual void Poweroff ()=0

  Sends a 'poweroff' signal to the sensor.
- virtual void Reboot ()=0

  Sends a 'reboot' signal to the sensor.
- virtual void Start (const std::size_t &timeoutMS=DEFAULT_STATE_CHANGE_TIMEOUT_MS)=0

  Starts the sensor.
- virtual void Stop (const std::size_t &timeoutMS=DEFAULT_STATE_CHANGE_TIMEOUT_MS)=0

  Stop the sensor.

4.7.1 Detailed Description

Sensor object interface.
This class defines the interface for a sensor. It provides methods to start, stop, and get the current state and other information of the sensor.

See also

SensorFactory for creating instances of this class

4.7.2 Member Function Documentation

GetCamera()

```
virtual std::shared_ptr< ICamera > GetCamera () const  [pure virtual]
```

Get the camera associated with this sensor.

Returns

Shared pointer to ICamera interface, or nullptr if no camera was configured

Returns a pointer to the camera object if a camera was configured during sensor creation via SensorFactory. If no camera was configured, returns nullptr.
The camera can be controlled independently from the sensor using its own Start(), Stop(), and exposure control methods.

See also

ICamera for camera control methods

SensorFactory::SetCameraIPAddress() for configuring camera

GetCurrentState()

```
virtual SensorState GetCurrentState () const  [pure virtual]
```

Provides the current state of the sensor (running, idle, error).

Returns

sensor state object (

See also

sensorState.h)

This function does not interact with the hardware but serves previously received information. Use GetErrors() to trigger an update if you suspect desynchronisation.

See also

sensorState, GetErrors()

GetErrors()

```
virtual std::vector< SensorError > GetErrors (
            const std::size_t & timeoutMS = DEFAULT_STATE_CHANGE_TIMEOUT_MS)  [pure virtual]
```

Retrieves the state from the sensor and reports a list of errors.

Parameters

timeoutMS	The maximum amount of time to wait the response (optional, default 5 sec)

Exceptions

sdk::exception	if no valid response is received in the timeout

Returns

A list of sensorError objects that indicate any problems or an empty vector if there are none.

This function send a TCP message to the sensor to request the sensor state. The return will contain information like the current state (idle, running, error), and more information on the errors if any.
A side effect of this function is that the state of the sensor object in the SDK is updated to match the state of the hardware.

Warning

This function throws an exception if it receives a sensor status response message that comes from a different sensor. This can happen if you have multiple sensors in the network, or if you are trying to query multiple sensors at once.

See also

SensorError

GetSerialNumber()

```
virtual SerialNumber GetSerialNumber () const  [pure virtual]
```

Provides the serial number of the sensor from local memory.

Returns

serialNumber object

This function does not interact with the hardware but serves previously received information. Use GetErrors() to trigger an update if you suspect desynchronisation.

See also

SerialNumber, GetErrors()

Poweroff()

```
virtual void Poweroff ()  [pure virtual]
```

Sends a 'poweroff' signal to the sensor.
Poweroff means that all software will stop, the components will be powered down and the sensor will remain powered off. In order to restart the sensor either toggle the ignition line or unplug and re-plug the power cord.
When the sensor powers off, the current sensor object is invalidated. Destroy all references to it and build a new sensor object if you need it.

Note

this function does not check if the action is executed, it simply posts the request.

Reboot()

```
virtual void Reboot ()  [pure virtual]
```

Sends a 'reboot' signal to the sensor.
Reboot means that all the components will be powered down and restarted as if from a clean boot.
When the sensor reboots, the current sensor object is invalidated. Destroy all references to it and build a new sensor object once the system has time to restart.

Note

this function does not check if the action is executed, it simply posts the request.

Romeinse straat 6, 3001 Heverlee, België
Tel: +32 16 40 64 34
www.xenomatix.com

Start()

```
virtual void Start (
            const std::size_t & timeoutMS = DEFAULT_STATE_CHANGE_TIMEOUT_MS)  [pure virtual]
```

Starts the sensor.

Parameters

timeoutMS	Timeout in milliseconds in which success response is expected (optional, default is 5 sec)

Exceptions

sdk::exception	object if no response is received within timeoutMS, or if the new state is not running.

Start means that the sensor is moved from idle to running state. It is only valid to call this function if the current state is idle. An exception is thrown otherwise. An exception is also thrown if the sensor did not respond timely or if the response is unexpected.

See also

GetCurrentState()

Stop()

```
virtual void Stop (
            const std::size_t & timeoutMS = DEFAULT_STATE_CHANGE_TIMEOUT_MS)  [pure virtual]
```

Stop the sensor.

Parameters

timeoutMS	Timeout in milliseconds in which success response is expected (optional, default is 5 sec)

Exceptions

sdk::exception	object if no response is received within timeoutMS, or if the new state is not idle.

Stop means that the sensor is moved from running to idle state. It is only valid to call this function if the current state is running. An exception is thrown otherwise. An exception is also thrown if the sensor did not respond timely or if the response is unexpected.

See also

GetCurrentState()

4.8 MetaData Struct Reference

Metadata struct containing general information for each point cloud line.

Public Attributes

- std::uint32_t **frameNumber** {0}

  Monotonically increasing counter. Unique for each frame acquired by the sensor.
- std::uint64_t **frameTimeStampUs** {0}

  Posix timestamp in us of start of frame acquisition.
- std::uint8_t **internalFrameNumber** {0}

  Internal, continuous incrementing frame counter. Overflows at 255.

- std::uint8_t **lineNumber** {0}

  Line number/counter. Indicates the vertical position within a frame of a dataset.
- std::uint64_t **lineTimeStampUs** {0}

  Posix timestamp in us of start of line acquisition.
- std::uint8_t **returnIndex** {0}

  Indicator for multiple returns mode (currently not supported, will always be 0).
- std::uint64_t **transmitTimeStampUs** {0}

  Posix timestamp in us when data leaves sensor.
- std::uint32_t **triggerCounter** {0}

  Monotonically increasing counter indicating a trigger connected to this frame.

4.8.1 Detailed Description

Metadata struct containing general information for each point cloud line.
This struct contains information such as the frame and line numbers, the timestamps and the return index.

4.9 OnboardProbes Struct Reference

struct containing information from sensors inside the sensors such as temperature and humidity.

Public Attributes

- float **humidityRelative**

  Relative humidity measurement from inside the sensor.
- float **temperatureDegC**

  Temperature measurement from inside the sensor in degrees Celsius.

4.9.1 Detailed Description

struct containing information from sensors inside the sensors such as temperature and humidity.

4.10 SensorFactory Class Reference

Factory object, to configure and construct sensor objects.

Public Member Functions

- std::shared_ptr< ISensor > Build (const std::string_view &ip4String=DEFAULT_XMANAGER_IP_↩ ADDRESS, const int port=DEFAULT_XMANAGER_PORT)

  Build the sensor object.
- void SetAlertCallback (alertCallback callback)

  Register a function where the SDK will report alerts.
- void SetCameraIPAddress (const std::string_view &ip4String)

  Configure the camera IP address for this sensor.
- void SetErrorCallback (errCallback callback)

  Register a function where the SDK will report errors.
- void SetImageBufferSize (std::size_t bufferSize)

  Allows control over the amount of incoming image messages that are buffered by the SDK.
- void SetImageCallback (imgCallback callback)

  Register a function where the SDK will report incoming camera image data.
- void SetImageOutputFormat (const PixelFormat format)

  Configure the pixel format of output images.
- void SetPointCloudBufferSize (std::size_t bufferSize)

  Allows control over the amount of incoming point cloud messages that are buffered by the SDK.
- void SetPointCloudCallback (pcCallback callback)

  Register a function where the SDK will report incoming point cloud data.

4.10.1 Detailed Description

Factory object, to configure and construct sensor objects.
Use the sensor factory to create objects of type ISensor. Use the available methods of the factory to configure the sensor object before creating it.
The factory can be reused to create multiple sensors with the same (or similar) configuration, or it can be used to recreate the same sensor object after the previous one was destroyed.
See 'iSensor.h' for more information on the sensor object itself.

4.10.2 Member Function Documentation

Build()

```
std::shared_ptr< ISensor > Build (
            const std::string_view & ip4String = DEFAULT_XMANAGER_IP_ADDRESS,
            const int port = DEFAULT_XMANAGER_PORT)
```

Build the sensor object.

Parameters

in	*ip4String*	address of the sensor (optional, defaults to 10.10.100.11)
in	*port*	of the sensor (optional, defaults to 4000)

Use this function to connect to the sensor and build the sensor object. This function will either return a valid sensor object to continue operation or will throw an exception.
If a camera IP address was configured via SetCameraIPAddress(), the camera will also be initialized and accessible via ISensor::GetCamera().
In case of errors use the logging to determine the underlying cause.

SetAlertCallback()

```
void SetAlertCallback (
            alertCallback callback)
```

Register a function where the SDK will report alerts.

Parameters

in	*callback*	function or lambda of the signature void(const Alert, const std::optional<bool>)

Alerts are events from the sensor indicating that something occurred to which the sensor is responding. Examples are overheating, opening of the sensor, ... Most Alerts have a boolean flag to indicate if the alert is raised or released. For example: when the sensor overheats an alert of type temperature_critical will be issued with the flag true. When the sensor cools again, a second alert of type temperature_critical will be issued with flag false.
Not all alerts have flags. E.g. the shutdown alert (indicating the sensor is powering off) does not have a boolean flag.
For most alerts, the sensor will stop functioning normally when the alert is raised.

SetCameraIPAddress()

```
void SetCameraIPAddress (
            const std::string_view & ip4String)
```

Configure the camera IP address for this sensor.

Parameters

in	*ip4String*	IPv4 address of the camera (e.g., "10.10.100.12")

Use this to provide a non default IP address for the camera. If you don't use this function, the default IP address

10.10.100.12 will be used.
The camera will be accessible via ISensor::GetCamera() after Build() completes.
Note: the SDK will only attempt to initialize the camera if an image callback is registered via SetImageCallback(). If no image callback is registered, the camera IP address will be ignored.

SetErrorCallback()

```
void SetErrorCallback (
            errCallback callback)
```

Register a function where the SDK will report errors.

Parameters

in	*callback*	function or lambda of the signature void(const Exception&)

Errors in this context are exceptions streamed by the sensor itself. Usually they are not the result of an action in the sdk.
See `error.h` header for the definition of Exception

SetImageBufferSize()

```
void SetImageBufferSize (
            std::size_t bufferSize)
```

Allows control over the amount of incoming image messages that are buffered by the SDK.

Parameters

in	*bufferSize*	The preallocated size of the buffer.

The default preallocated buffer is 5 frames. Adjust based on available memory and processing capabilities.
A large buffer will result in stale data if processing cannot keep up.

SetImageCallback()

```
void SetImageCallback (
            imgCallback callback)
```

Register a function where the SDK will report incoming camera image data.

Parameters

in	*callback*	function or lambda of the signature void(std::unique_ptr<IImage>)

This function will be called by the SDK for every camera frame received. The callback execution should be fast to avoid blocking image acquisition.
The data is provided as a unique_ptr, which means that ownership of the image object is transferred from the SDK to the callback function. It will not be used again in the SDK, do with it as you please.
The SDK will only attempt to initialize the camera if an image callback is registered.
See the `iImage.h` header for the definition of IImage

SetImageOutputFormat()

```
void SetImageOutputFormat (
            const PixelFormat format)
```

Configure the pixel format of output images.

Parameters

in	*format*	The desired pixel format for the output images.

The default format is RGB12.
There are three options:

- BayerRG12: raw Bayer data from the camera, 12 bits per pixel (packed).
- RGB8: 3 bytes per pixel (RGBRGBRGB).
- RGB12: 12-bit RGB data, unpacked, 2 bytes per color channel (RGBRGBRGB).

Consequences:

- BayerRG12: most compact format and no compression calculations required, but requires demosaicing to get RGB values. Does not support to save as jpg.
- RGB8: straightforward to use, but larger in size compared to BayerRG12. Good for visualization.
- RGB12: highest quality, but requires more memory and processing power. Keeps the full 12-bit color information from the camera, which can be beneficial for post-processing and analysis.

This choice is important. The SDK will not be able to change the format after the image has been processed initially. Once it is BayerRG12, this SDK cannot convert it to RGB. Similarly, once it is RGB8, the image cannot be converted to RGB12.

SetPointCloudBufferSize()

```
void SetPointCloudBufferSize (
            std::size_t bufferSize)
```

Allows control over the amount of incoming point cloud messages that are buffered by the SDK.

Parameters

in	*bufferSize*	The preallocated size of the buffer.

The default preallocated buffer is 2 full frames. Lower the buffer size if ram memory is limited. Less than 20 lines will likely result in occasional misses, depending on processor speed, processor load and network occupancy. Lower than 3 lines will give continuous misses due to bunching of data on the ethernet line. Do not use this to compensate for processing not keeping up to recording. If processing cannot keep up, the buffer will fill up anyway and the size will never be enough. A large buffer will result in stale data.

SetPointCloudCallback()

```
void SetPointCloudCallback (
            pcCallback callback)
```

Register a function where the SDK will report incoming point cloud data.

Parameters

in	*callback*	function or lambda of the signature void(std::unique_ptr<IPointCloud>)

This function will be called by the sdk for every line. Due to network variations this will not be a steady rhythm, but groups of 2 or 3 lines at once. The sdk has an internal buffer and will provide the next line either when the previous callback returns or when new data arrives from the sensor.
In order to keep up with the sensor, it is advised that callbacks return within 850 microseconds or less.
The data is provided as a unique_ptr, which means that ownership of the point cloud object is transferred from the SDK to the callback function. It will not be used again in the sdk, do with it as you please.
See the `iPointCloud.h` header for the definition of IPointCloud

4.11 SerialNumber Class Reference

utility class that holds a serial number and allows interpretation.

4.11.1 Detailed Description

utility class that holds a serial number and allows interpretation.

Romeinse straat 6, 3001 Heverlee, België
Tel: +32 16 40 64 34
www.xenomatix.com