Pico Zense TOF RGBD Camera SDK Developer Guide

Windows/Linux

2019.06

Pico Technology Co., Ltd.

Table of Contents

1 Overview 1

2 SDK Structure 2

2.1 SDK Structure on Windows 2

2.2 SDK Structure on Linux 2

3 Requirements 3

4 Setting up the Development Environment 4

4.1 Hardware Installation 4

4.1.1 Setting up the Hardware Driver in Windows 4

4.1.2 Setting up the Hardware Driver in Linux 4

4.2 Setting up the Windows Development Environment 5

4.2.1 Creating a new Application 5

4.2.2 (Optional) Downloading and Installing OpenCV for Windows 7

4.3 Setting up the Linux Development Environment 8

4.3.1 Setting up a PC with an Nvidia Graphics Card 8

4.3.2 Setting up a PC with an Intel Graphics Card 9

4.3.3 Verifying the Installation of VDPAU 9

4.3.4 (Optional) Downloading and Installing OpenCV for Linux 10

5 API Usage 11

6 Quickstart 13

6.1 Windows Quickstart 13

6.2 Linux Quickstart 13

7 API Reference 14

7.1 Enumerations 14

7.1.1 PsDataMode 14

7.1.2 PsDepthRange 14

7.1.3 PsFilterType 15

7.1.4 PsFrameType 16

7.1.5 PsPixelFormat 16

7.1.6 PsPropertyType 17

7.1.7 PsResolution 17

7.1.8 PsReturnStatus 18

7.1.9 PsSensorType 18

7.1.10 PsStreamType 18

7.1.11 PsWDRTotalRange 19

7.1.12 PsWDRStyle 19

7.2 Structures 20

7.2.1 PsAudioFrame 20

7.2.2 PsCameraExtrinsicParameters 20

7.2.3 PsCameraParameters 20

7.2.4 PsDepthVector3 21

7.2.5 PsFrame 21

7.2.6 PsFrameMode 22
7.2.7 PsGMMGain 22
7.2.8 PsImu 22
7.2.9 PsImuWithParams 22
7.2.10 PsRGB888Pixel 23
7.2.11 PsBGR888Pixel 23
7.2.12 PsVector3f 23
7.2.13 PsWDROutputMode 23

7.3 Functions 25

7.3.1 PsInitialize 25
7.3.2 PsShutdown 25
7.3.3 PsOpenDevice 25
7.3.4 PsCloseDevice 25
7.3.5 PsStartFrame 26
7.3.6 PsStopFrame 26
7.3.7 PsReadNextFrame 27
7.3.8 PsGetFrame 27
7.3.9 PsGetIMU 28
7.3.10 PsGetImuWithParams 28
7.3.11 PsGetAudio 29
7.3.12 PsConvertDepthFrameToWorldVector 29
7.3.13 PsConvertDepthToWorld 30
7.3.14 PsConvertWorldToDepth 30
7.3.15 PsGetCameraParameters 31
7.3.16 PsGetCameraExtrinsicParameters 32
7.3.17 PsGetComputeRealDepthCorrectionEnabled 32
7.3.18 PsGetDepthDistortionCorrectionEnabled 33
7.3.19 PsGetDepthRange 33
7.3.20 PsGetDeviceCount 34
7.3.21 PsGetFilter 34
7.3.22 PsGetFrameMode 35
7.3.23 PsGetGMMGain 35
7.3.24 PsGetIrDistortionCorrectionEnabled 35
7.3.25 PsGetMapperEnabledDepthToRGB 36
7.3.26 PsGetMapperEnabledRGBToDepth 36
7.3.27 PsGetMapperEnabledRGBToIR 37
7.3.28 PsGetProperty 37
7.3.29 PsGetPulseCount 38
7.3.30 PsGetResolution 38
7.3.31 PsGetRGBDistortionCorrectionEnabled 39
7.3.32 PsGetSmoothingFilterEnabled 39
7.3.33 PsGetSpatialFilterEnabled 40
7.3.34 PsGetSynchronizeEnabled 40
7.3.35 PsGetThreshold 40
7.3.36 PsGetTimeFilterEnabled 41
7.3.37 PsGetWDROutputMode 41
7.3.38 PsSetColorPixelFormat 42
7.3.39 PsSetComputeRealDepthCorrectionEnabled 42
7.3.40 PsSetDataMode 43
7.3.41 PsSetDepthDistortionCorrectionEnabled 43
7.3.42 PsSetDepthRange 44
7.3.43 PsSetFilter 44

7.3.44 PsSetFrameMode 45
7.3.45 PsSetGMMGain..45
7.3.46 PsSetIrDistortionCorrectionEnabled...46
7.3.47 PsSetMapperEnabledDepthToRGB...46
7.3.48 PsSetMapperEnabledRGBToDepth...47
7.3.49 PsSetMapperEnabledRGBToIR 47
7.3.50 PsSetProperty 48
7.3.51 PsSetPulseCount...48
7.3.52 PsSetResolution 49
7.3.53 PsSetRGBDistortionCorrectionEnabled 49
7.3.54 PsSetSmoothingFilterEnabled 49
7.3.55 PsSetSpatialFilterEnabled 50
7.3.56 PsSetSynchronizeEnable...50
7.3.57 PsSetThreshold 51
7.3.58 PsSetTimeFilterEnabled..51
7.3.59 PsSetWDROutputMode..52
7.3.60 PsSetWDRStyle 52
7.3.61 PsSetWDRFusionThreshold 53

1 Overview

Welcome to the SDK Developer Guide for the Pico Zense TOF RGBD Camera (DCAM710). The DCAM710 is a standalone 3D camera hardware module, developed by Pico, that uses time-of-flight (TOF) technology to capture depth information while recording imagery. The DCAM710 is packaged in a small form factor and can adapt to a wide range of environments while providing high-precision depth information.

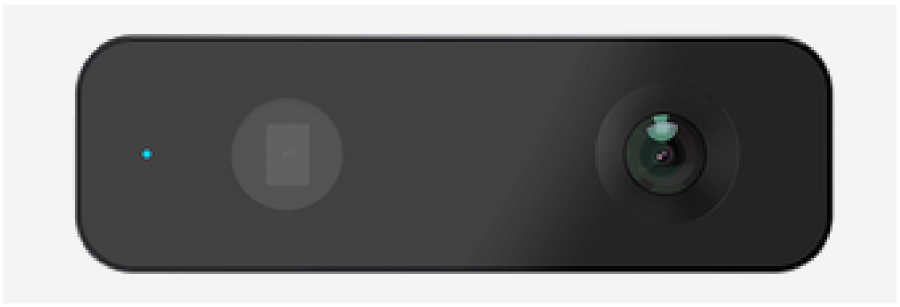


Figure 1 – Pico Zense TOF RGBD Camera ：DCAM710

Some of the use cases for acquiring depth information include:

- Gesture, facial, and fatigue recognition
- TV and game motion-sensitivity interaction
- Navigation and obstacle avoidance
- Advanced automotive vision systems
- 3D modeling
- Industrial control

The Pico Zense SDK enables developers to programmatically interact with the DCAM710 on Windows and Linux platforms.

This document provides developers with the following information:

- SDK Structure
- How to set up a development environment
- An overview of the API usage and workflow
- A quickstart tutorial
- An API reference guide

2 SDK Structure

2.1 SDK Structure on Windows

The Pico Zense SDK package for Windows contains the following directories and files:

- **Bin**:
    - **picozense_api.dll** (available in x64 and x86 versions): The API runtime DLL. This DLL, along with the dependencies in the directory, must be copied to the same directory as your target executable application.
- **DCAM710_Env.bat**: Batch file to set up environment variables used when building the SDK.
- **Include**: The API's header files:
    - **PicoZense_api.h**: API functions.
    - **PicoZense_define.h**: Preprocessor defines for library imports and exports.
    - **PicoZense_enums.h**: API enumerations.
    - **PicoZense_types.h**: API data structures.
- **Lib**:
    - **picozense_api.lib**: The API binary to link with at build time.
- **VERSION**: Text file specifying the SDK's version number.

2.2 SDK Structure on Linux

The Pico Zense SDK package for Linux contains the following directories and files:

- **Include**: The API's header files:
    - **PicoZense_api.h**: The API header file.
    - **PicoZense_define.h**: Preprocessor defines for library imports and exports.
    - **PicoZense_enums.h**: API enumerations.
    - **PicoZense_types.h**: API data structures.
- **Lib**:
    - **libpicozense_api.so**: The runtime API shared object.

Note: Examples for using the SDK can be found on the ZenseSamples GitHub repo.

3 Requirements

The Pico Zense TOF RGBD Camera SDK has the following requirements:

Supported Operating Systems:

- **Windows**: Windows 7 64-bit, Windows 10 64-bit
- **Linux**: Ubuntu 14.04 64-bit, Ubuntu 16.04 64-bit, Ubuntu 18.04[1]

RAM: A minimum of 4GB

[1] OpenCV must be installed from an older version of Ubuntu. See Section 4.3.4 for more information.

4 Setting up the Development Environment

4.1 Hardware Installation

Connect the DCAM710 to a PC using a USB cable as shown in Figure 2:

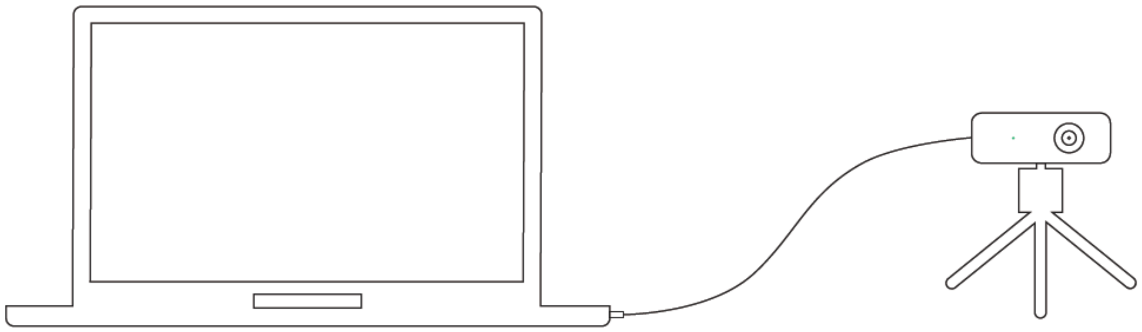
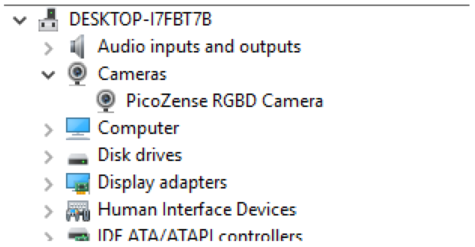


Figure 2 - Hardware installation

4.1.1 Setting up the Hardware Driver in Windows

When the DCAM710 is successfully connected, Windows will display the device driver installation screen. The driver will auto-install and then display `Pico Zense RGBD Camera` in Windows Device Manger:



```
DESKTOP-I7FBT7B
  > Audio inputs and outputs
  v Cameras
      PicoZense RGBD Camera
  > Computer
  > Disk drives
  > Display adapters
  > Human Interface Devices
  > IDE ATA/ATAPI controllers
```

Figure 3 – Pico Zense RGBD Camera Driver in Windows Device Manager

4.1.2 Setting up the Hardware Driver in Linux

Run the `install.sh` script provided with the SDK to copy the USB device rule to `/etc/udev/rules.d/`. The OpenCV installation will also be copied to a common directory.

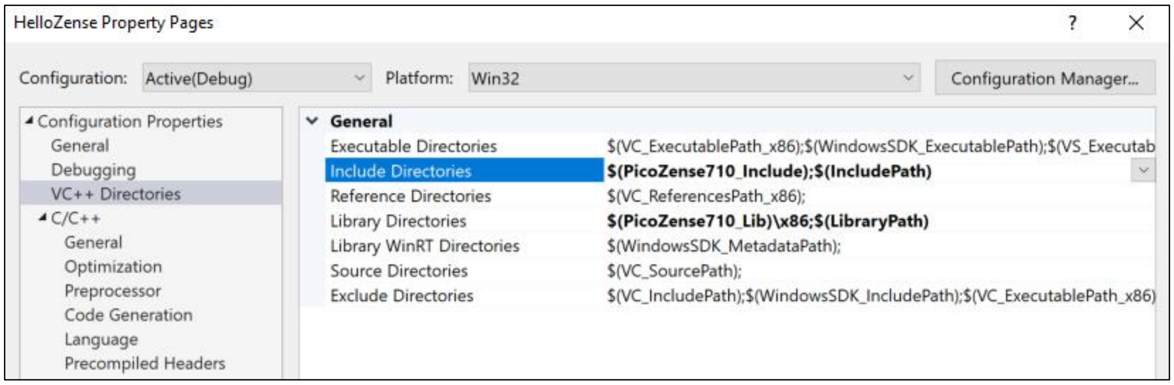
Note: Developers should review the script prior to running it, to ensure that the directories specified match the layout of the development system.

4.2 Setting up the Windows Development Environment

4.2.1 Creating a new Application

Follow the steps below to create a new application:

1. Create a new application project in Visual Studio.
2. Open the project's properties.
3. Select **VC++ Directories**.
4. Locate the **Include Directories** row, click on the down arrow, and select **<Edit…>**:

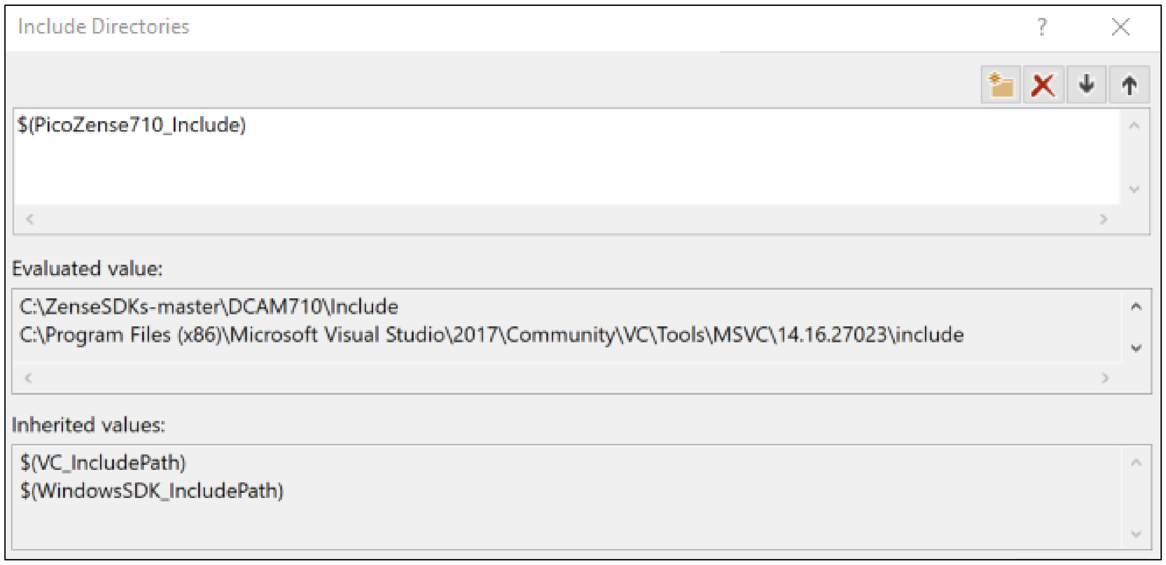


Configuration: Active(Debug) Platform: Win32

General

Executable Directories	$(VC_ExecutablePath_x86);$(WindowsSDK_ExecutablePath);$(VS_Executab
Include Directories	**$(PicoZense710_Include);$(IncludePath)**
Reference Directories	$(VC_ReferencesPath_x86);
Library Directories	**$(PicoZense710_Lib)\x86;$(LibraryPath)**
Library WinRT Directories	$(WindowsSDK_MetadataPath);
Source Directories	$(VC_SourcePath);
Exclude Directories	$(VC_IncludePath);$(WindowsSDK_IncludePath);$(VC_ExecutablePath_x86)

5. Add `$(PicoZense710_Include)` to the include directories for the Pico Zense SDK and close the dialog:



$(PicoZense710_Include)

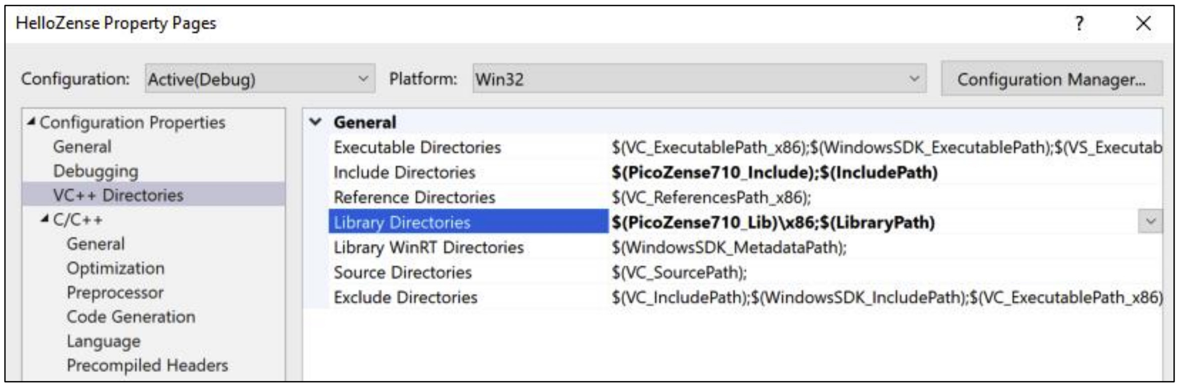
Evaluated value:

C:\ZenseSDKs-master\DCAM710\Include
C:\Program Files (x86)\Microsoft Visual Studio\2017\Community\VC\Tools\MSVC\14.16.27023\include

Inherited values:

$(VC_IncludePath)
$(WindowsSDK_IncludePath)

6. Locate the **Library Directories** row, click on the down arrow, and select **<Edit…>**:

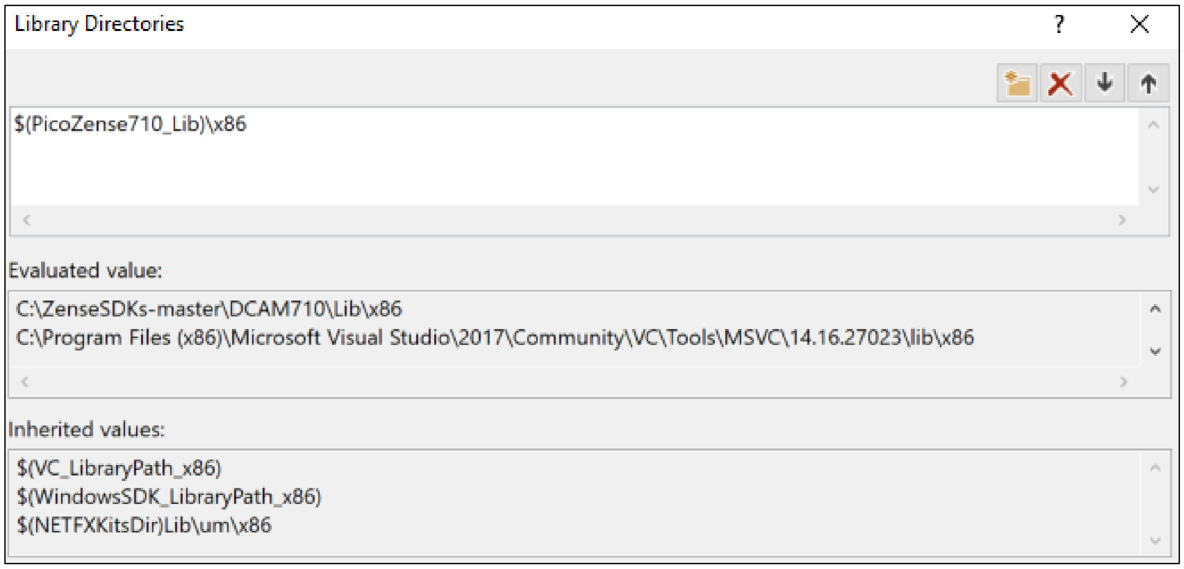


Configuration: Active(Debug) Platform: Win32

General

Executable Directories	$(VC_ExecutablePath_x86);$(WindowsSDK_ExecutablePath);$(VS_Executab
Include Directories	**$(PicoZense710_Include);$(IncludePath)**
Reference Directories	$(VC_ReferencesPath_x86);
Library Directories	**$(PicoZense710_Lib)\x86;$(LibraryPath)**
Library WinRT Directories	$(WindowsSDK_MetadataPath);
Source Directories	$(VC_SourcePath);
Exclude Directories	$(VC_IncludePath);$(WindowsSDK_IncludePath);$(VC_ExecutablePath_x86)

7. Add `$(PicoZense710_Lib)\x86` to the Pico Zense SDK's library directories and close the dialog:



$(PicoZense710_Lib)\x86

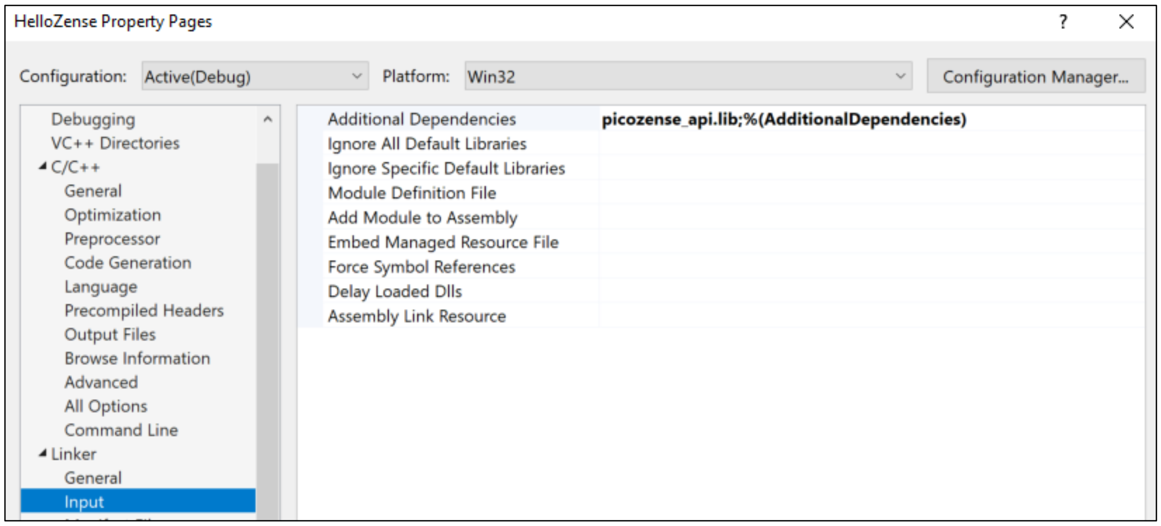
Evaluated value:

C:\ZenseSDKs-master\DCAM710\Lib\x86
C:\Program Files (x86)\Microsoft Visual Studio\2017\Community\VC\Tools\MSVC\14.16.27023\lib\x86

Inherited values:

$(VC_LibraryPath_x86)
$(WindowsSDK_LibraryPath_x86)
$(NETFXKitsDir)Lib\um\x86

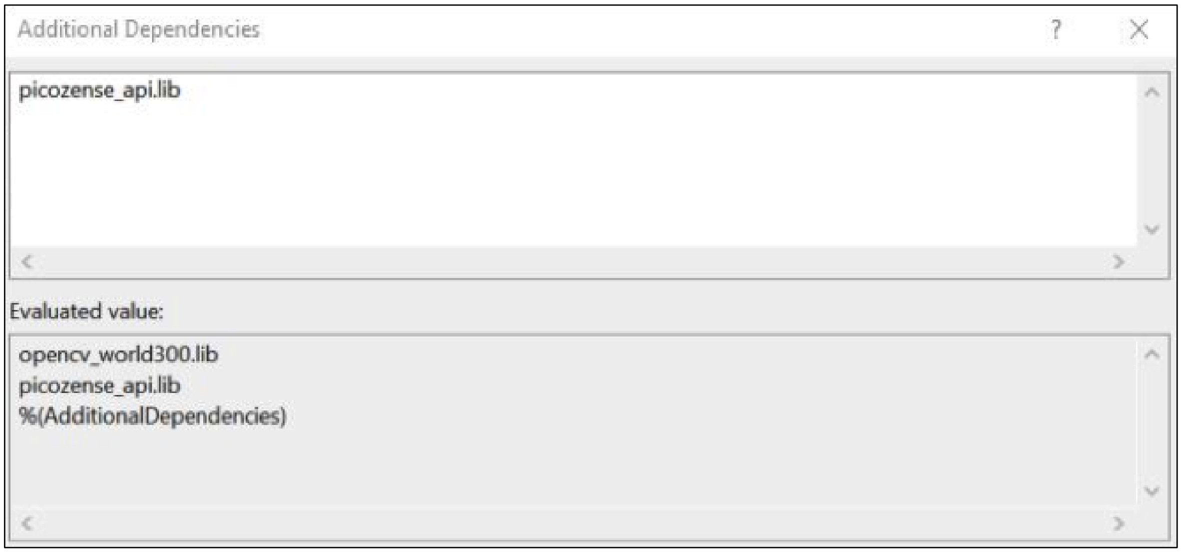
8. Select **Linker > Input**.
9. Locate the **Additional Dependencies** row, click on the down arrow, and select **<Edit...>**.



Configuration: Active(Debug) Platform: Win32

Additional Dependencies	**picozense_api.lib;%(AdditionalDependencies)**
Ignore All Default Libraries	
Ignore Specific Default Libraries	
Module Definition File	
Add Module to Assembly	
Embed Managed Resource File	
Force Symbol References	
Delay Loaded Dlls	
Assembly Link Resource	

10. Add `picozense_api.lib` and close the dialog:



```
picozense_api.lib
```

Evaluated value:

```
opencv_world300.lib
picozense_api.lib
%(AdditionalDependencies)
```

11. Click **OK** to apply the changes to the project.

4.2.2 (Optional) Downloading and Installing OpenCV for Windows

4.2.2.1 Downloading OpenCV

OpenCV is a computer vision library that can be used in conjunction with the Pico Zense API, and is used in some of the Pico Zense SDK's samples. Follow the steps below to obtain OpenCV for Windows:

1. Navigate to OpenCV in a web browser and locate Version 3.x.x.
2. Click on **Win pack** to download *OpenCV*:

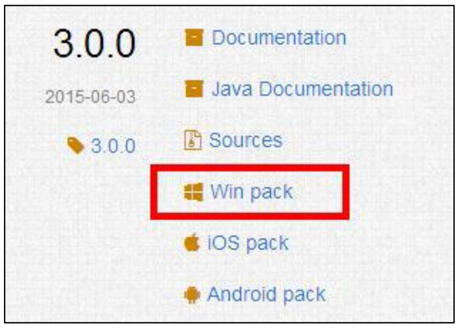


Figure 4 - Downloading and installing OpenCV

3. Install the *OpenCV* package.
4. Set the environment variable `OPENCV_DIR` to the absolute path of OpenCV's `build` directory (e.g. `c:\opencv-3.0.0\build`).

4.2.2.2 Adding OpenCV to a Project

Follow the steps below to add OpenCV to a Visual Studio Project:

1. Open the project in Visual Studio.
2. Open the project's properties.

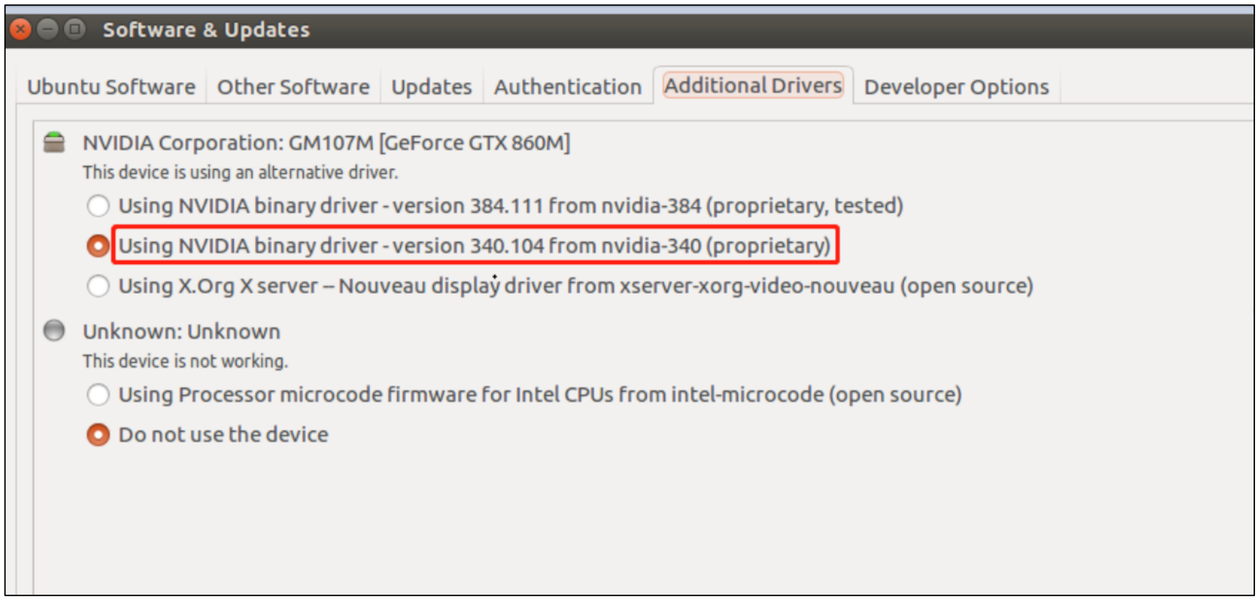
3. Select **VC++ Directories**.
4. Locate the **Include Directories** row, click on the down arrow, and select **<Edit…>**.
5. Add `$(OPENCV_DIR)\include` to the include directories for the OpenCV SDK and close the dialog.
6. Locate the **Library Directories** row, click on the down arrow, and select **<Edit…>**.
7. Add `$(OPENCV_DIR)\x86\vc12\lib` to the paths and close the dialog.
8. Select **Linker > Input**.
9. Locate the **Additional Dependencies** row, click on the down arrow, and select **<Edit…>**.
10. Add `opencv_world300.lib` and close the dialog.
11. Click **OK** to apply the changes to the project.

4.3 Setting up the Linux Development Environment

4.3.1 Setting up a PC with an Nvidia Graphics Card

If your PC is equipped with an Nvidia video card, follow the steps below to prepare your Linux development environment:

1. Update the graphic driver to an Nvidia driver, as shown in Figure 5:


Software & Updates

Ubuntu Software | Other Software | Updates | Authentication | Additional Drivers | Developer Options

NVIDIA Corporation: GM107M [GeForce GTX 860M]
This device is using an alternative driver.
- ○ Using NVIDIA binary driver - version 384.111 from nvidia-384 (proprietary, tested)
- ● Using NVIDIA binary driver - version 340.104 from nvidia-340 (proprietary)
- ○ Using X.Org X server – Nouveau display driver from xserver-xorg-video-nouveau (open source)

Unknown: Unknown
This device is not working.
- ○ Using Processor microcode firmware for Intel CPUs from intel-microcode (open source)
- ● Do not use the device

Figure 5 - Graphics driver settings

2. Install *Video Decode and Presentation API* for Unix (*VDPAU*):

```
sudo apt-get update
sudo apt-get install libvdpau-dev
sudo apt-get install vdpauinfo
```

4.3.2 Setting up a PC with an Intel Graphics Card

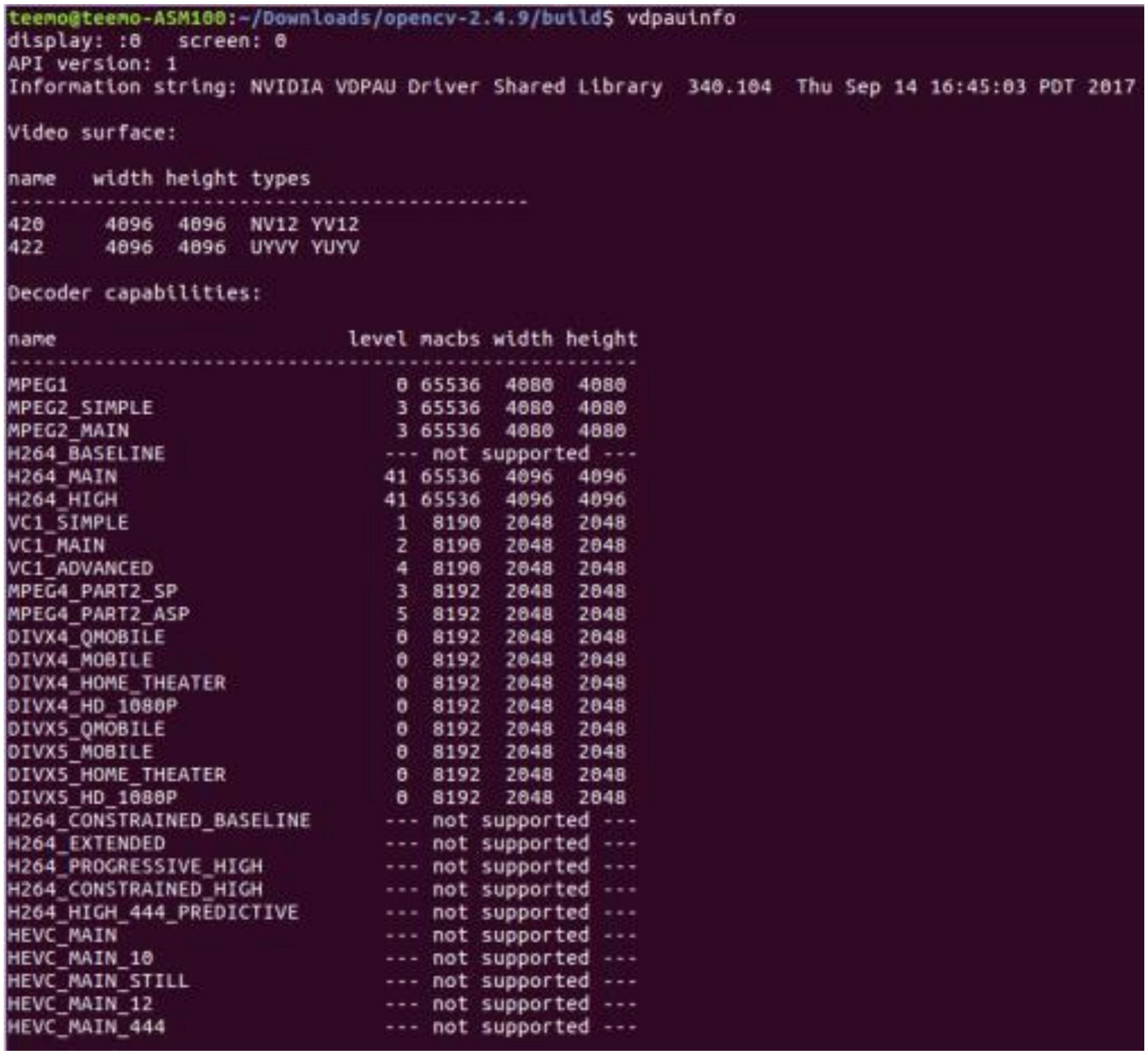
If your PC is equipped with an Intel video card, follow the steps below to prepare your Linux development environment:

1. Install the binaries for the graphics driver and *Video Decode and Presentation API for Unix* (*VDPAU*):

```
sudo add-apt-repository ppa:nilarimogard/webupd8
sudo apt-get update
sudo apt-get install libvdpau-va-gl1
sudo apt-get install i965-va-driver
sudo apt-get install vdpauinfo
```

4.3.3 Verifying the Installation of VDPAU

Run the `vdpauinfo` command. If *VDPAU* has been installed correctly, `vdpauinfo` will display information about the decoder's capabilities, similar to that shown in Figure 6:



```
teemo@teemo-ASM100:~/Downloads/opencv-2.4.9/build$ vdpauinfo
display: :0   screen: 0
API version: 1
Information string: NVIDIA VDPAU Driver Shared Library  340.104  Thu Sep 14 16:45:03 PDT 2017

Video surface:

name   width height types
-------------------------------------------
420     4096  4096  NV12 YV12
422     4096  4096  UYVY YUYV

Decoder capabilities:

name                        level macbs width height
----------------------------------------------------
MPEG1                           0 65536  4080  4080
MPEG2_SIMPLE                    3 65536  4080  4080
MPEG2_MAIN                      3 65536  4080  4080
H264_BASELINE                  --- not supported ---
H264_MAIN                      41 65536  4096  4096
H264_HIGH                      41 65536  4096  4096
VC1_SIMPLE                      1  8190  2048  2048
VC1_MAIN                        2  8190  2048  2048
VC1_ADVANCED                    4  8190  2048  2048
MPEG4_PART2_SP                  3  8192  2048  2048
MPEG4_PART2_ASP                 5  8192  2048  2048
DIVX4_QMOBILE                   0  8192  2048  2048
DIVX4_MOBILE                    0  8192  2048  2048
DIVX4_HOME_THEATER              0  8192  2048  2048
DIVX4_HD_1080P                  0  8192  2048  2048
DIVX5_QMOBILE                   0  8192  2048  2048
DIVX5_MOBILE                    0  8192  2048  2048
DIVX5_HOME_THEATER              0  8192  2048  2048
DIVX5_HD_1080P                  0  8192  2048  2048
H264_CONSTRAINED_BASELINE      --- not supported ---
H264_EXTENDED                  --- not supported ---
H264_PROGRESSIVE_HIGH          --- not supported ---
H264_CONSTRAINED_HIGH          --- not supported ---
H264_HIGH_444_PREDICTIVE       --- not supported ---
HEVC_MAIN                      --- not supported ---
HEVC_MAIN_10                   --- not supported ---
HEVC_MAIN_STILL                --- not supported ---
HEVC_MAIN_12                   --- not supported ---
HEVC_MAIN_444                  --- not supported ---
```

Figure 6 - Output from vdpauinfo

If the following errors are displayed:

```
display: :0 screen: 0 Failed to open VDPAU backend libvdpau_i965.so:
cannot open shared object file: No such file or directory
Error creating VDPAU device: 1
```

then invoke the commands below to correct the problem:

```
cd /usr/lib/x86_64-linux-gnu/vdpau/
sudo ln -s libvdpau_va_gl.so libvdpau_i965.so
sudo ln -s libvdpau_va_gl.so.1 libvdpau_i965.so.
```

4.3.4 (Optional) Downloading and Installing OpenCV for Linux

OpenCV is a computer vision library that can be used in conjunction with the Pico Zense API, and is used in some of the Pico Zense SDK's samples.

The Ubuntu Linux SDK includes OpenCV 3.4, which can be used on newer versions of Ubuntu (16.x and higher), with the addition of older supporting library versions:

- `libjasper1 (libjasper.so.1)`
- `libpng12-0 (libpng12.so.0)`
- `libwebp5 (libwebp.so.5)`

There are two methods for adding the older libraries:

1. Add older, Xenial Xerus repositories to your local Ubuntu installation by adding the following to `/etc/apt/sources.list`:

   ```
   deb http://security.ubuntu.com/ubuntu xenial-security main
   deb http://us.archive.ubuntu.com/ubuntu xenial main
   ```

   Note: An appropriate, local repository should be substituted.

   Once added, the older libraries can be installed by `apt`.
2. Manually download them from a `.deb` package repository and install them with `dpkg` tools.

Note: Similar approaches can be used for RPM-based distributions.

5 API Usage

Using the Pico Zense API to programmatically interact with the DCAM710 involves the following operations:

- Include `PicoZense_api.h`.
- Initialize the SDK with `PsInitialize`.
- Get the number of devices connected using `PsGetDeviceCount`.
- Open a device with `PsOpenDevice`.
- Set the device's properties such as the depth range.
- Begin the frame capture processes for depth, IR, and/or RGB frames process by invoking `PsStartFrame` for each frame type to be captured.
- While in a loop:
  - Start image capture on the next frame using `PsReadNextFrame`.
  - Capture depth, IR, and/or RGB frame images using `PsGetFrame`.
- Stop the frame capture processes for depth, IR, and/or RGB frames using `PsStopFrame`.
- Close the device using `PsCloseDevice`.
- Shutdown the SDK with `PsShutdown`.

This workflow is illustrated in Figure 7:

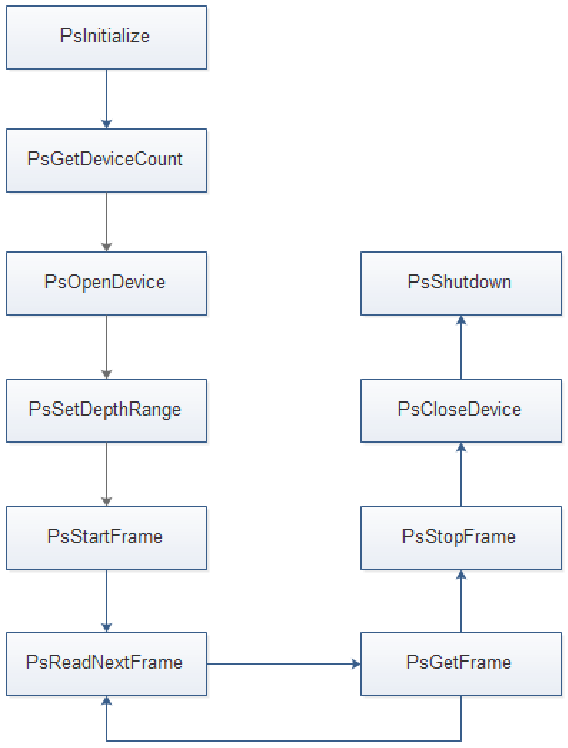


Figure 7 - API Workflow

The following sample shows the code for this workflow, and the code comments refer to the workflow described above:

```
//include the Pico Zense API’s main header file
#include "PicoZense_api.h"

int main(int argc, char* argv[])
{
        PsReturnStatus status;

        // Initialize the SDK
        status = PsInitialize();

        // Get the number of devices currently connected
        int32_t deviceCount = 0;
        status = PsGetDeviceCount(&deviceCount);

        int32_t deviceIndex = 0;

        // Open the specified depth camera device
        status = PsOpenDevice(deviceIndex);

        // Set properties such as the depth range
        status = PsSetDepthRange(deviceIndex, PsNearRange);

        // Start collecting image data streams for depth, IR, and RGB
        status = PsStartFrame(deviceIndex, PsDepthFrame);
        status = PsStartFrame(deviceIndex, PsIRFrame);
        status = PsStartFrame(deviceIndex, PsRGBFrame);

        // Image processing main loop
        for ( ;; )
        {
                // Capture a frame image
                PsReadNextFrame(deviceIndex);

                PsFrame depthFrame = {0};
                PsFrame irFrame = { 0 };
                PsFrame rgbFrame = { 0 };

                // Obtain a frame of image data for the specified image types
                status = PsGetFrame(deviceIndex, PsDepthFrame, &depthFrame);
                status = PsGetFrame(deviceIndex, PsIRFrame, &irFrame);
                status = PsGetFrame(deviceIndex, PsRGBFrame, &rgbFrame);

                // Process the frame data here...
        }

        // Close the depth, IR, and RGB streams
        status = PsStopFrame(deviceIndex, PsDepthFrame);
        status = PsStopFrame(deviceIndex, PsIRFrame);
        status = PsStopFrame(deviceIndex, PsRGBFrame);

        // Close the specified depth camera device
        status = PsCloseDevice(deviceIndex);

        // Shut down the SDK and release all of the resources created by the SDK
        status = PsShutdown();
        return 0;
}
```

6 Quickstart

The following subsections describe how to build the *FrameViewer* example.

6.1 Windows Quickstart

Follow the steps below to build *FrameViewer* in Visual Studio.

Note: The Visual Studio project files use relative directories, so the file structure of the SDK must be kept intact. This is also the case for the sample `makefile`.

1. Open `\PicoZenseSDK_Windows_<version>_DCAM710\Samples\FrameViewer\FrameViewer.sln` with Visual Studio and build the solution.

   Note: The project will automatically copy `opencv_world300.dll`, `PicoZense_api.dll` and all config files to the same directory as the target executable when building the solution.

2. Run and debug *FrameViewer*.

Note: The SDK also includes a pre-built copy of `FrameViewer.exe` and its dependencies in the SDK's `Tools` directory.

6.2 Linux Quickstart

A makefile for the Linux version of *FrameViewer* is provided in the SDK package.

To build the FrameViewer app:

1. Open a terminal window.
2. Navigate to the `FrameViewer` directory:

```
cd PicoZenseSDK_Ubuntu16.04_<version>_DCAM710/Samples/FrameViewer
```

3. Invoke `make` to start the build.

7 API Reference

7.1 Enumerations

7.1.1 PsDataMode

The data modes that determines the frame output from the device and the frame rate (fps).

Name	Description
`PsDepthAndRGB_30`	Output both depth and RGB frames at 30 fps. The resolution of a depth frame is 640*480. The resolution of an RGB frame can be set using `PsSetFrameMode`, which supports 1920*1080/1280*720/640*480/640*360.
`PsIRAndRGB_30`	Outputs both IR and RGB frames at 30 fps. The resolution of an IR frame is 640*480. The resolution of an RGB frame can be set using `PsSetFrameMode`, which supports 1920*1080/1280*720/640*480/640*360.
`PsDepthAndIR_30`	Outputs both depth and IR frames at 30 fps. The resolution for both depth and IR frames is 640*480.
`PsDepthAndIRAndRGB_30`	Outputs depth/IR/RGB frames at 30 fps. The resolution of both depth and IR frames is 640*360. The resolution of an RGB frame can be set using `PsSetFrameMode`, which supports 1920*1080/1280*720/640*480/640*360.
`PsDepthAndIR_15_RGB_30`	Outputs depth and IR frames at 15 fps, alternating between the two. The resolution of both depth and IR frames is 640*480. The resolution of an RGB frame can be set using `PsSetFrameMode`, which supports 1920*1080/1280*720/640*480/640*360.
`PsWDR_Depth`	WDR (Wide Dynamic Range) depth mode. Supports alternating multi-range depth frame output (e.g. Near/Far/Near/Far/Near).
`PsWDR_IR`	WDR (Wide Dynamic Range) IR mode. Not currently implemented.
`PsWDR_DepthAndIR`	WDR (Wide Dynamic Range) Depth and IR mode. Not currently implemented.

7.1.2 PsDepthRange

Defines the depth range modes that can be set for the DCAM710.

Note: Currently only the `PsNearRange`, `PsMidRange`, and `PsFarRange` modes are supported.

Name	Description

`PsNearRange`	`Near` range mode – 1 meter, determined by specific calibration.
`PsMidRange`	`Middle` range mode – 2 meters, determined by specific calibration.
`PsFarRange`	`Far` range mode – 4 meters, determined by specific calibration.
`PsXNearRange`	`XNear` range mode – 5 meters, determined by specific calibration.
`PsXMidRange`	`XMid` range mode – 7 meters , determined by specific calibration
`PsXFarRange`	`XFar` range mode – 7 meters , determined by specific calibration
`PsXXNearRange`	`XXNear` range mode – 9 meters , determined by specific calibration
`PsXXMidRange`	`XXMiddle` range mode – 11 meters , determined by specific calibration
`PsXXFarRange`	`XXFar` range mode – 15 meters , determined by specific calibration

7.1.3 PsFilterType

Specifies the filter type.

Name	**Description**
`PsComputeRealDepthFilter`	Compute the real depth, in the depth camera coordinate system. Enabled by default.
`PsSmoothingFilter`	Smoothing filter. Enabled by default.

7.1.4 PsFrameType

Specifies the type of image frame.

Name	Description
`PsDepthFrame`	Depth frame with 16 bits per pixel, in millimeters.
`PsIRFrame`	IR frame with 16 bits per pixel.
`PsGrayFrame`	Mono gray frame with 8 bits per pixel. Not current available on the DCAM710.
`PsRGBFrame`	RGB frame with 24 bits per pixel in RGB/BGR format.
`PsMappedRGBFrame`	RGB frame with 24 bits per pixel in RGB/BGR format, that is mapped to depth camera space where the resolution is the same as the depth frame's resolution. This frame type can be enabled using `PsSetMapperEnabledDepthToRGB`.
`PsMappedDepthFrame`	Depth frame with 16 bits per pixel, in millimeters, that is mapped to RGB camera space where the resolution is same as the RGB frame's resolution. This frame type can be enabled using `PsSetMapperEnabledRGBToDepth`.
`PsMappedIRFrame`	IR frame with 16 bits per pixel that is mapped to RGB camera space where the resolution is the same as the RGB frame's resolution. This frame type can be enabled using `PsSetMapperEnabledRGBToIR`.
`PsConfidenceFrame`	Confidence frame with 16 bits per pixel, not implemented currently.
`PsWDRDepthFrame`	WDR depth frame with 16 bits per pixel in millimeters. This only takes effect when the data mode set to `PsWDR_Depth`. For more information see `PsDataMode`.

7.1.5 PsPixelFormat

Specifies the pixel format.

Name	Description
`PsPixelFormatDepthMM16`	Depth image pixel format, 16 bits per pixel in mm.
`PsPixelFormatGray16`	IR image pixel format, 16 bits per pixel.
`PsPixelFormatGray8`	Gray image pixel format, 8 bits per pixel.
`PsPixelFormatRGB888`	Color image pixel format, 24 bits per pixel RGB format.

PsPixelFormatBGR888	Color image pixel format, 24 bits per pixel BGR format.

7.1.6 PsPropertyType

Specifies the property to get or set on a device.

Name	Description
PsPropertySN_Str	Device serial number (e.g.PD7110CGC9270020W). The maximum length is 64 bytes.
PsPropertyFWVer_Str	Device firmware version number (e.g. DCAM710_c086_pc_sv0.01_R6_20180917_b35). The maximum length is 64 bytes.
PsPropertyHWVer_Str	Device hardware version number (e.g. R6). The maximum length is 64 bytes.
PsPropertyDataMode_UInt8	Sets the data mode when invoking PsSetDataMode. See PsDataMode for more information. The data mode: • **PsDepthAndRGB_30**: Output both depth and RGB frames at 30fps. The resolution of a depth frame is 640*480. The resolution of an RGB frame can be set using PsSetFrameMode, which supports resolutions of 1920*1080, 1280*720, 640*480, and 640*360. • **PsIRAndRGB_30**: Output both IR and RGB frames at 30fps. The resolution of an IR frame is 640*480. The resolution of an RGB frame can be set using PsSetFrameMode, which supports resolutions of 1920*1080, 1280*720, 640*480, and 640*360. • **PsDepthAndIR_30**: Output both depth and IR frames at 30fps. The resolution of both frame types is 640*480. • **PsDepthAndIRAndRGB_30**: Output depth, IR, and RGB frames at 30fps. The resolution of the depth and IR frames is 640*360. The resolution of the RGB frames can be set using PsSetFrameMode, which supports resolutions of 1920*1080, 1280*720, 640*480, and 640*360.

7.1.7 PsResolution

Specifies the resolution of RGB frames.

Name	Description
PsRGB_Resolution_1920_1080	The resolution of RGB is 1920*1080.
PsRGB_Resolution_1280_720	The resolution of RGB is 1280*720.

`PsRGB_Resolution_640_480`	The resolution of RGB is 640*480.
`PsRGB_Resolution_640_360`	The resolution of RGB is 640*360.

7.1.8 PsReturnStatus

Specifies the result of an API call.

Name	Description
`PsRetOK`	The function completed successfully.
`PsRetNoDeviceConnected`	There is no depth camera connected or the camera has not been connected correctly. Check the hardware connection or try unplugging and re-plugging the USB cable.
`PsRetInvalidDeviceIndex`	The input device index is invalid.
`PsRetDevicePointerIsNull`	The device structure pointer is null.
`PsRetInvalidFrameType`	The input frame type is invalid.
`PsRetFramePointerIsNull`	The output frame is empty.
`PsRetNoPropertyValueGet`	Cannot get the value for the specified property.
`PsRetNoPropertyValueSet`	Cannot set the value for the specified property.
`PsRetPropertyPointerIsNul`	The input property value buffer pointer is null.
`PsRetPropertySizeNotEnough`	The input property value buffer size is too small to store the specified property value.
`PsRetInvalidDepthRange`	The input depth range mode is invalid.
`PsRetInputPointerIsNull`	An input pointer parameter is null.
`PsRetReadNextFrameError`	An error occurred when capturing the next image frame.
`PsRetCameraNotOpened`	The camera has not been opened.
`PsRetInvalidCameraType`	The specified type of camera is invalid.
`PsRetInvalidParams`	One or more of the parameter values provided are invalid.
`PsRetOthers`	An unknown error occurred.

7.1.9 PsSensorType

Specifies the type of camera sensor.

Name	Description
`PsDepthSensor`	Depth camera.
`PsRgbSensor`	Color (RGB) camera.

7.1.10 PsStreamType

Specifies the type of stream.

Name	Description
`PsStreamDepth`	Depth stream.
`PsStreamIR`	IR stream.
`PsStreamRGB`	RGB stream.
`PsStreamAudio`	Audio stream.
`PsStreamIMU`	IMU data stream.

7.1.11 PsWDRTotalRange

Specifies the number of depth ranges defined for WDR.

Name	Description
`PsWDRTotalRange_Two`	Two depth ranges.
`PsWDRTotalRange_Three`	Three depth ranges.

7.1.12 PsWDRStyle

Specifies the WDR style used for `PsSetWDRStyle`. This determines if the WDR image output is a fusion from multiple ranges (e.g. Near/Far fusion) or an alternative output (e.g. Near/Far/Near/Far ...).

Name	Description
`PsWDR_FUSION`	WDR image output is fused from multiple ranges.
`PsWDR_ALTERNATION`	WDR image output alternates between depths (e.g. Near/Far/Near/Far ...).

7.2 Structures

7.2.1 PsAudioFrame

Contains a frame of audio data.

Name	**Type**	**Description**
`audioFormat`	8-bit unsigned integer	Specifies the audio format. Set to `0` for PCM.
`numChannels`	8-bit unsigned integer	Specifies the number of channels. Set to `1` for mono or `2` for stereo.
`bitsPerSample`	8-bit unsigned integer	Specifies the bits per sample (e.g. 16 bits).
`sampleRate`	32-bit unsigned integer	Specifies the sample rate at which the audio frame was captured (e.g. 16 kHz).
`pData`	8-bit unsigned integer pointer	Pointer to a buffer containing the audio data for the frame.
`dataLen`	32-bit unsigned integer	The length, in bytes, of the data in `pData`.

7.2.2 PsCameraExtrinsicParameters

Specifies the camera's location and orientation extrinsic parameters.

Name	**Type**	**Description**
`rotation`	Array of `double`	Orientation stored as an array of 9 `double` representing a 3x3 rotation matrix.
`translation`	Array of `double`	Location stored as an array of 3 `double` representing a 3-D translation vector.

7.2.3 PsCameraParameters

Camera intrinsic parameters and distortion coefficients.

Name	**Type**	**Description**
`fx`	double	Focal length x, in pixels.
`fy`	double	Focal length y, in pixels.
`cx`	double	Principal point x, in pixels.
`cy`	double	Principal point y, in pixels.

`k1`	double	First-order radial distortion coefficient.
`k2`	double	Second-order radial distortion coefficient.
`p1`	double	Tangential distortion coefficient.
`p2`	double	Tangential distortion coefficient.
`k3`	double	Third-order radial distortion coefficient.
`k4`	double	Fourth-order radial distortion coefficient.
`k5`	double	Fifth-order radial distortion coefficient.
`k6`	double	Sixth-order radial distortion coefficient.

7.2.4 PsDepthVector3

Contains depth information for a given pixel.

Name	**Type**	**Description**
`depthX`	32-bit integer	The x coordinate of the pixel.
`depthY`	32-bit integer	The y coordinate of the pixel.
`depthZ`	`PsDepthPixel` (16-bit unsigned integer)	The depth of the pixel, in millimeters.

7.2.5 PsFrame

Specifies the type of frame and contains the data stored in the frame.

Name	**Type**	**Description**
`frameIndex`	32-bit unsigned integer	The index of the frame.
`frameType`	8-bit unsigned integer	The type of frame. See `PsFrameType` for more information.
`pixelFormat`	8-bit unsigned integer	The pixel format used by a frame. See `PsPixelFormat` for more information.
`imuFrameNo`	8-bit unsigned integer	Used to synchronize with IMU, in the range of 0 to 255.
`pFrameData`	8-bit unsigned integer pointer	A buffer containing the frame's image data.

`dataLen`	32-bit unsigned integer	The length of `pFrame`, in bytes.
`exposureTime`	`float`	The exposure time, in milliseconds.
`depthRange`	`PsDepthRange` pointer	The depth range mode of the frame. Used only for depth frames.
`width`	16-bit unsigned integer	The width of the frame, in pixels.
`height`	16-bit unsigned integer	The height of the frame, in pixels.

7.2.6 PsFrameMode

Specifies the properties of an image frame.

Name	Type	Description
`pixelFormat`	`PsPixelFormat`	The pixel format used by a frame.
`resolutionWidth`	32-bit integer	The width of the image, in pixels.
`resolutionHeight`	32-bit integer	The height of the image, in pixels.
`Fps`	32-bit integer	The image stream frame rate.

7.2.7 PsGMMGain

Specifies the properties of an image frame.

Name	Type	Description
`gain`	`uint16_t`	The GMM gain value of the device.
`Option`	uint8_t	The option type of setting the GMM gain effective time.

7.2.8 PsImu

Specifies the IMU data.

Name	Type	Description
`Acc`	`PsVector3f`	The accelerometer sensor measurement (m/s^2).
`Gyro`	`PsVector3f`	The gyroscope sensor measurement (rad/s).
`frameNo`	8-bit unsigned integer	The frame number.

7.2.9 PsImuWithParams

Specifies the IMU with extra parameters.

Name	Type	Description
`Acc`	`PsVector3f`	The accelerometer sensor measurement (m/s^2).
`Gyro`	`PsVector3f`	The gyroscope sensor measurement (rad/s).
`temp`	`float`	The temperature, in degrees Celsius.
`frameNo`	8-bit unsigned integer	The frame number in the range of 0 to 255.

7.2.10 PsRGB888Pixel

Color image pixel type in 24-bit RGB format.

Name	Type	Description
r	8-bit unsigned integer	Red
g	8-bit unsigned integer	Green
b	8-bit unsigned integer	Blue

7.2.11 PsBGR888Pixel

Color image pixel type in 24-bit BGR format.

Name	Type	Description
B	8-bit unsigned integer	Blue
G	8-bit unsigned integer	Green
r	8-bit unsigned integer	Red

7.2.12 PsVector3f

Stores the x, y, and z components of a 3D vector.

Name	Type	Description
`x`	`float`	The x component of the vector.
`y`	`float`	The y component of the vector.
`z`	`float`	The z component of the vector.

7.2.13 PsWDROutputMode

WDR (Wide Dynamic Range) output mode settings (e.g. Near/Far range fusion).

Name	**Type**	**Description**
totalRange	PsWDRTotalRange	The number of ranges supported. Currently only two or three ranges are supported (e.g. Near/Far or Near/Mid/Far).
range1	PsDepthRange	The type of the first depth range.
range1Count	8-bit unsigned integer	The count of successive range1 frames.
range2	PsDepthRange	The type of the second depth range.
range2Count	8-bit unsigned integer	The count of successive range2 frames.
range3	PsDepthRange	Third range. This range only takes effect when totalRange is set to 3.
range3Count	8-bit unsigned integer	The count of successive range3 frames. This only takes effect when totalRange is set to 3.

7.3 Functions

7.3.1 PsInitialize

`PsReturnStatus PsInitialize()`

Initializes the API. This function must be invoked before any other Pico Zense APIs.

Parameters

None

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.2 PsShutdown

`PsReturnStatus PsShutdown()`

Shuts down the API and clears all resources allocated by the API. After invoking this function, no other Pico Zense APIs can be invoked.

Parameters

None

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.3 PsOpenDevice

`PsReturnStatus PsOpenDevice(int32_t deviceIndex)`

Opens the device specified by `deviceIndex`. The device must be subsequently closed using `PsCloseDevice`.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device to open. Device indices range from 0 to device count – 1.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.4 PsCloseDevice

`PsReturnStatus PsCloseDevice(int32_t deviceIndex)`

Closes the device specified by `deviceIndex` that was opened using `PsOpenDevice`.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device to close. Device indices range from 0 to device count – 1.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.5 PsStartFrame

`PsReturnStatus PsStartFrame(int32_t deviceIndex, PsFrameType frameType)`

Starts capturing the image stream indicated by `frameType` on the device specified by `deviceIndex`. Invoke `PsStopFrame` to stop capturing the image stream.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device on which to start capturing the image stream. Device indices range from 0 to device count – 1.
`frameType`	`PsFrameType`	The type of image stream to start capturing.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.6 PsStopFrame

`PsReturnStatus PsStopFrame(int32_t deviceIndex, PsFrameType frameType)`

Stops capturing the image stream on the device specified by `deviceIndex`, that was started using `PsStartFrame`.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device on which to stop capturing the image

		stream. Device indices range from 0 to device count – 1.
`frameType`	`PsFrameType`	The type of image stream to stop capturing.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.7 PsReadNextFrame

```
PsReturnStatus PsReadNextFrame(int32_t deviceIndex)
```

Captures the next image frame from the device specified by `deviceIndex`. This API must be invoked before capturing frame data using `PsGetFrame`.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device on which to read the next frame. Device indices range from 0 to device count – 1.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.8 PsGetFrame

```
PsReturnStatus PsGetFrame(int32_t deviceIndex, PsFrameType frameType,
PsFrame* pPsFrame)
```

Returns the image data for the current frame from the device specified by `deviceIndex`. Before invoking this API, invoke `PsReadNextFrame` to capture one image frame from the device.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device to capture an image frame from. Device indices range from 0 to device count – 1.
`frameType`	`PsFrameType`	The image frame type.

`ppFrame`	`PsFrame pointer`	Pointer to a `PsFrame` variable in which to store the returned image data.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.9 PsGetIMU

`PsReturnStatus PsGetImu(int32_t deviceIndex, PsImu& imu)`

Returns IMU data from the device specified by `deviceIndex` (the update rate is 1000hz).

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device from which to get the IMU data. Device indices range from 0 to device count – 1.
`imu`	`PsImu` reference	Reference to a `PsImu` variable in which to store the IMU data.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.10 PsGetImuWithParams

`PsReturnStatus PsGetImuWithParams(int32_t deviceIndex, PsImuWithParams& imu)`

Returns IMU with parameters data from the device specified by `deviceIndex` (the update rate is 1000hz).

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device from which to get the IMU data. Device indices range from 0 to device count – 1.
`pImuDataWithParams`	`PsImuWithParams` reference	Reference to a `PsImuWithParams` variable in which to store the IMU data.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.11 PsGetAudio

```
PsReturnStatus PsGetAudio(int32_t deviceIndex, PsAudioFrame* audio)
```

Returns audio data from the device specified by `deviceIndex`.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device from which to get audio. Device indices range from 0 to device count – 1.
`audio`	`PsAudioFrame` pointer	Pointer to a `PsAudioFrame` variable in which to store the audio data.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.12 PsConvertDepthFrameToWorldVector

```
PsReturnStatus PsConvertDepthFrameToWorldVector(int32_t deviceIndex, const
PsFrame& depthFrame, PsVector3f* pWorldVector)
```

Converts the input Depth frame from depth coordinate space to world coordinate space on the device specified by <code>deviceIndex</code>.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device from which to get audio. Device indices range from 0 to device count – 1.
`depthFrame`	`PsFrame`	Pointer to a `PsFrame` variable in which stored the returned image data.
`pWorldVector`	`PsVector3f` pointer	Pointer to a buffer in which to output the converted x, y, and z values of the world coordinates, measured in millimeters.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.13 PsConvertDepthToWorld

```
PsReturnStatus PsConvertDepthToWorld (int32_t deviceIndex, PsDepthVector3*
pDepthVector, PsVector3f* pWorldVector, int32_t pointCount)
```

Converts the input points from depth coordinate space to world coordinate space on the device specified by `deviceIndex`.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device on which to perform the operation. Device indices range from 0 to device count – 1.
`pDepthVector`	`PsDepthVector3` pointer	Pointer to a buffer containing the x, y, and z values of the depth coordinates to be converted. x and y are measured in pixels, where 0, 0 is located at the top left corner of the image. z is measured in millimeters, based on the `PsPixelFormat` depth frame.
`pWorldVector`	`PsVector3f` pointer	Pointer to a buffer in which to output the converted x, y, and z values of the world coordinates, measured in millimeters.
`pointCount`	32-bit integer	The number of points to convert.

Returns
`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.14 PsConvertWorldToDepth

```
PsReturnStatus PsConvertWorldToDepth(int32_t deviceIndex, PsVector3f*
pWorldVector, PsDepthVector3* pDepthVector, int32_t pointCount)
```

Converts the input points from world coordinate space to depth coordinate space on the device specified by `deviceIndex`.

Parameters

Name	Type	Description

`deviceIndex`	32-bit integer	The index of the device on which to perform the operation. Device indices range from 0 to device count – 1.
`pWorldVector`	`PsVector3f` pointer	Pointer to a buffer containing the x, y, and z values of the input world coordinates to be converted, measured in millimeters.
`pDepthVector`	`PsDepthVector3` pointer	Pointer to a buffer in which to output the converted x, y, and z values of the depth coordinates. x and y are measured in pixels, where 0, 0 is located at the top left corner of the image. z is measured in millimeters, based on the `PsPixelFormat` depth frame.
`pointCount`	32-bit integer	The number of coordinates to convert.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.15 PsGetCameraParameters

```
PsReturnStatus PsGetCameraParameters(int32_t deviceIndex, PsSensorType
sensorType, PsCameraParameters* pCameraParameters)
```

Returns the internal intrinsic and distortion coefficient parameters of the device specified by `deviceIndex`.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device from which to get the internal parameters. Device indices range from 0 to device count – 1.
`sensorType`	`PsSensorType`	The type of sensor from which to get parameter information. Set to `PsDepthSensor` to specify the depth sensor, or `PsRgbSensor` to specify the RGB sensor.

`pCameraParameters`	`PsCameraParameters`	Pointer to a `PsCameraParameters` variable in which to store the parameter values.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.16 PsGetCameraExtrinsicParameters

```
PsGetCameraExtrinsicParameters(int32_t deviceIndex,
PsCameraExtrinsicParameters* pCameraExtrinsicParameters)
```

Returns the camera rotation and translation coefficient parameters for the device specified by `deviceIndex`.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device from which to get the extrinsic parameters. Device indices range from 0 to device count – 1.
`pCameraExtrinsicParameters`	`PsCameraExtrinsicParameters` pointer	Pointer to a `PsGetCameraExtrinsicParameters` variable in which to store the parameters.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.17 PsGetComputeRealDepthCorrectionEnabled

```
PsReturnStatus PsGetComputeRealDepthCorrectionEnabled(int32_t deviceIndex,
bool *bEnabled)
```

Returns the boolean value of whether the ComputeRealDepth feature is enabled or disabled.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device from which to get the extrinsic parameters. Device indices range from 0 to device count – 1.

`bEnabled`	`bool`	Pointer to a variable in which to store the returned Boolean value.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.18 PsGetDepthDistortionCorrectionEnabled

`PsReturnStatus PsGetDepthDistortionCorrectionEnabled(int32_t deviceIndex, bool *bEnabled)`

Returns the boolean value of whether the depth distortion correction feature is enabled or disabled.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device from which to get the depth range. Device indices range from 0 to device count – 1.
`bEnabled`	`bool`	Pointer to a variable in which to store the returned Boolean value.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.19 PsGetDepthRange

`PsReturnStatus PsGetDepthRange(int32_t deviceIndex, PsDepthRange* pDepthRange)`

Returns the depth range mode for the device specified by `deviceIndex`.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device from which to get the depth range. Device indices range from 0 to device count – 1.
`pDepthRange`	`PsDepthRange` pointer	A pointer to a `PSDepthRange` variable in which to store the returned depth range mode.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.20 PsGetDeviceCount

```
PsReturnStatus PsGetDeviceCount(int32_t* pDeviceCount)
```

Returns the number of camera devices currently connected.

Parameters

Name	Type	Description
`pDeviceCount`	32-bit integer pointer	Pointer to a 32-bit integer variable in which to return the device count.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.21 PsGetFilter

```
PsReturnStatus PsGetFilter(int32_t deviceIndex, PsFilterType filterType,
bool *bEnabled)
```

Returns the boolean value of whether the filter feature specified by <code>filterType</code> is enabled or disabled.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device to capture an image frame from. Device indices range from 0 to device count – 1.
`filterType`	`PsFrameType`	Specifies the type of filter.
`bEnabled`	`bool`	Pointer to a variable in which to store the returned Boolean value.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.22 PsGetFrameMode

```
PsReturnStatus PsGetFrameMode(int32_t deviceIndex, PsFrameType frameType,
PsFrameMode* pFrameMode)
```

Returns the image frame mode corresponding to the frame type specified by `frameType`.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device from which to get the frame mode. Device indices range from 0 to device count – 1.
`frameType`	`PsFrameType`	The type of image frame for which to get the frame mode.
`pFrameMode`	`PsFrameMode` pointer	Pointer to a `PsFrameMode` variable in which to store the image frame mode.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.23 PsGetGMMGain

```
PsReturnStatus PsGetGMMGain(int32_t deviceIndex, uint16_t* gmmgain)
```

Returns the device's GMM gain on the device specified by `deviceIndex`.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device from which to get the GMM gain. Device indices range from 0 to device count – 1.
`gmmgain`	16-bit unsigned integer pointer	Pointer to a variable in which to store the returned GMM gain.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.24 PsGetIrDistortionCorrectionEnabled

`PsReturnStatus PsGetIrDistortionCorrectionEnabled(int32_t deviceIndex, bool *bEnabled)`

Returns the boolean value of whether the Ir distortion correction feature is enabled or disabled.

Parameters

Name	**Type**	**Description**
`deviceIndex`	32-bit integer	The index of the device from which to get the IMU data. Device indices range from 0 to device count – 1.
`bEnabled`	`bool`	Pointer to a variable in which to store the returned Boolean value.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.25 PsGetMapperEnabledDepthToRGB

`PsReturnStatus PsGetMapperEnabledDepthToRGB(int32_t deviceIndex, bool *bEnabled)`

Returns the boolean value of whether the mapping of the depth image to RGB space feature is enabled or disabled.

Parameters

Name	**Type**	**Description**
`deviceIndex`	32-bit integer	The index of the device from which to get the property value. Device indices range from 0 to device count – 1.
`bEnabled`	bool	Pointer to a variable in which to store the returned Boolean value.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.26 PsGetMapperEnabledRGBToDepth

`PsReturnStatus PsGetMapperEnabledRGBToDepth(int32_t deviceIndex, bool *bEnabled)`

Returns the boolean value of whether the mapping of the RGBToDepth feature is enabled or disabled.

Parameters

Name	**Type**	**Description**

`deviceIndex`	32-bit integer	The index of the device from which to get the property value. Device indices range from 0 to device count – 1.
`bEnabled`	bool	Pointer to a variable in which to store the returned Boolean value.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.27 PsGetMapperEnabledRGBToIR

```
PsReturnStatus PsGetMapperEnabledRGBToIR(int32_t deviceIndex, bool
*bEnabled)
```

Returns the Boolean value of whether the mapping of the RGBToIR feature is enabled or disabled.

Parameters

Name	**Type**	**Description**
`deviceIndex`	32-bit integer	The index of the device from which to get the property value. Device indices range from 0 to device count – 1.
`bEnabled`	bool	Pointer to a variable in which to store the returned Boolean value.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.28 PsGetProperty

```
PsReturnStatus PsGetProperty(int32_t deviceIndex, PsPropertyType
propertyType, void* pData, int32_t* pDataSize)
```

Returns a specific property value for the device specified by `deviceIndex`.

Parameters

Name	**Type**	**Description**
`deviceIndex`	32-bit integer	The index of the device from which to get the property value. Device indices range from 0 to device count – 1.
`propertyType`	32-bit integer	The type of property to get from the device. See

		`PsPropertyType` for more information.
`pData`	Void pointer	Pointer to a buffer to store the returned property value.
`pDataSize`	32-bit integer	The size, in bytes, of the property value returned in `pData`.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.29 PsGetPulseCount

```
PsReturnStatus PsGetPulseCount(int32_t deviceIndex, uint16_t* pulseCount)
```

Returns the pulse count for the device specified by `deviceIndex`.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device on which to get the pulse count. Device indices range from 0 to device count – 1.
`pPulseCount`	16-bit unsigned integer pointer	Pointer to a 16-bit unsigned integer variable in which to store the pulse count value.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.30 PsGetResolution

```
PsReturnStatus PsGetResolution(int32_t deviceIndex, uint16_t* resolution)
```

Returns the the RGB frame Resolution.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device on which to get the pulse count. Device indices range from 0 to device count – 1.

`resolution`	`PsResolution`	Pointer to a variable in which to store the returned resolution.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.31 PsGetRGBDistortionCorrectionEnabled

```
PsReturnStatus PsGetRGBDistortionCorrectionEnabled(int32_t deviceIndex,
bool *bEnabled)
```

Returns the Boolean value of whether the RGB distortion correction feature is enabled or disabled.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device from which to get the threshold. Device indices range from 0 to device count – 1.
`bEnabled`	b**ool**	Pointer to a variable in which to store the returned Boolean value.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.32 PsGetSmoothingFilterEnabled

```
PsReturnStatus PsGetSmoothingFilterEnabled(int32_t deviceIndex, bool
*bEnabled)
```

Returns the Boolean value of whether the Smoothing Filter feature is enabled or disabled.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device from which to get the threshold. Device indices range from 0 to device count – 1.
`bEnabled`	b**ool**	Pointer to a variable in which to store the returned Boolean value.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.33 PsGetSpatialFilterEnabled

```
PsReturnStatus PsGetSpatialFilterEnabled(int32_t deviceIndex, bool
*bEnabled)
```

Returns the Boolean value of whether the Spatial Filter feature is enabled or disabled.

Parameters

Name	**Type**	**Description**
`deviceIndex`	32-bit integer	The index of the device from which to get the threshold. Device indices range from 0 to device count – 1.
`bEnabled`	b**ool**	Pointer to a variable in which to store the returned Boolean value.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.34 PsGetSynchronizeEnabled

```
PsReturnStatus PsGetSynchronizeEnabled(int32_t deviceIndex, bool
*bEnabled)
```

Returns the Boolean value of whether the RGB and Depth synchronize feature is enabled or disabled.

Parameters

Name	**Type**	**Description**
`deviceIndex`	32-bit integer	The index of the device from which to get the threshold. Device indices range from 0 to device count – 1.
`bEnabled`	b**ool**	Pointer to a variable in which to store the returned Boolean value.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.35 PsGetThreshold

```
PsReturnStatus PsGetThreshold(int32_t deviceIndex, uint16_t* pThreshold)
```

Returns the threshold value for the background filter on the device specified by `deviceIndex`. The value represents the cut-off point for distant data that the filter should ignore. For example, if 20.0 is specified, data with 20% or less confidence will be dropped.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device from which to get the threshold. Device indices range from 0 to device count – 1.
`pThreshold`	16-bit unsigned pointer	Pointer to a 16-bit unsigned integer variable in which to return the threshold value.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.36 PsGetTimeFilterEnabled

```
PsReturnStatus PsGetTimeFilterEnabled(int32_t deviceIndex, bool *bEnabled)
```

Returns the Boolean value of whether the Time Filter feature is enabled or disabled.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device from which to get the threshold. Device indices range from 0 to device count – 1.
`bEnabled`	b**ool**	Pointer to a variable in which to store the returned Boolean value.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.37 PsGetWDROutputMode

```
PsReturnStatus PsGetWDROutputMode(int32_t deviceIndex, PsWDROutputMode*
pWDRMode)
```

Returns the WDR mode of the device specified by `deviceIndex`.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device from which to get the WDR mode.

		Device indices range from 0 to device count – 1.
`pWDRMode`	`PsWDROutputMode`	Pointer to a `PsWDROutputMode` variable in which to store the current WDR output mode.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.38 PsSetColorPixelFormat

```
PsReturnStatus PsSetColorPixelFormat(int32_t deviceIndex, const
PsPixelFormat pixelFormat)
```

Sets the color image pixel format on the device specified by `deviceIndex`. Currently only RGB and BGR formats are supported.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device to set the pixel format. Device indices range from 0 to device count – 1.
`pixelFormat`	`PsPixelFormat`	The color pixel format to use. Pass in one of the values defined by `PsPixelFormat`. Currently only `PsPixelFormatRGB888` and `PsPixelFormatBGR888` are supported.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.39 PsSetComputeRealDepthCorrectionEnabled

```
PsReturnStatus PsSetComputeRealDepthCorrectionEnabled(int32_t deviceIndex,
bool bEnabled)
```

Enables or disables the ComputeRealDepth feature.

Parameters

Name	Type	Description

deviceIndex	32-bit integer	The index of the device to set the pixel format. Device indices range from 0 to device count – 1.
bEnabled	bool	Set to <code>true</code> to enable the feature or <code>false</code> to disable the feature.

Returns

PsRetOK if the function succeeded, or one of the error values defined by PsReturnStatus.

7.3.40 PsSetDataMode

```
PsReturnStatus PsSetDataMode (int32_t deviceIndex, PsDataMode dataMode)
```

Sets the output data mode on the device specified by deviceIndex.

Parameters

Name	Type	Description
deviceIndex	32-bit integer	The index of the device for which to set the data mode. Device indices range from 0 to device count – 1.
dataMode	PsDataMode	The output data mode.

Returns

PsRetOK if the function succeeded, or one of the error values defined by PsReturnStatus.

7.3.41 PsSetDepthDistortionCorrectionEnabled

```
PsReturnStatus PsSetDepthDistortionCorrectionEnabled (int32_t deviceIndex,
bool bEnabled)
```

Enables or disables the depth distortion correction feature on the device specified by deviceIndex.

Parameters

Name	Type	Description
deviceIndex	32-bit integer	The index of the device on which to enable or disable the depth distortion correction feature. Device indices range from 0 to device count – 1.

`bEnabled`	`bool`	Set to `true` to enable the feature or `false` to disable the feature.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.42 PsSetDepthRange

`PsReturnStatus PsSetDepthRange(int32_t deviceIndex, PsDepthRange depthRange)`

Sets the depth range mode for the device specified by `deviceIndex`.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device on which to set the depth range. Device indices range from 0 to device count – 1.
`depthRange`	`PsDepthRange`	Specifies the depth range mode.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.43 PsSetFilter

```
PsReturnStatus PsSetFilter(int32_t deviceIndex, int32_t filterType, bool
bEnabled)
```

Enables or disables a filter on the device specified by `deviceIndex`.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device on which to enable or disable the filter. Device indices range from 0 to device count – 1.
`filterType`	32-bit integer	Specifies the type of filter to enable or disable.
`bEnabled`	`bool`	Set to `true` to enable the feature or `false` to disable the feature.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.44 PsSetFrameMode

```
PsReturnStatus PsSetFrameMode(int32_t deviceIndex, PsFrameType frameType,
PsFrameMode* pFrameMode)
```

Sets the image frame mode on the device specified by `deviceIndex`.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device for which to set the frame mode. Device indices range from 0 to device count – 1.
`frameType`	`PsFrameType`	The type of image frame for which to set the frame mode.
`pFrameMode`	`PsFrameMode` pointer	Pointer to a `PsFrameMode` variable specifying the image frame mode.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.45 PsSetGMMGain

```
PsReturnStatus PsSetGMMGain (int32_t deviceIndex, uint16_t gmmgain)
```

Sets the GMM gain on the device specified by `deviceIndex`.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device on which to set the GMM gain. Device indices range from 0 to device count – 1.
`gmmgain`	16-bit unsigned integer	The GMM gain value to set.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.46 PsSetIrDistortionCorrectionEnabled

```
PsReturnStatus PsSetIrDistortionCorrectionEnabled (int32_t deviceIndex,
bool bEnabled)
```

Enables or disables the IR distortion correction feature on the device specified by `deviceIndex`.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device on which to enable or disable the IR distortion correction feature. Device indices range from 0 to device count – 1.
`bEnabled`	`bool`	Set to `true` to enable the feature or `false` to disable the feature.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.47 PsSetMapperEnabledDepthToRGB

```
PsReturnStatus PsSetMapperEnabledDepthToRGB(int32_t deviceIndex, bool
bEnabled)
```

Enables or disables mapping of the RGB image in Depth space on the device specified by `deviceIndex`. When enabled, `PsGetFrame` can be invoked passing `PsMappedRGBFrame` as the frame type, to get the RGB frame that is mapped to Depth space. The resolution of the mapped RGB frame is the same as that of the Depth image.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device on which to enable or disable mapping. Device indices range from 0 to device count – 1.
`bEnabled`	`bool`	Set to `true` to enable the feature or `false` to disable the feature.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined `PsReturnStatus`.

7.3.48 PsSetMapperEnabledRGBToDepth

```
PsReturnStatus PsSetMapperEnabledRGBToDepth(int32_t deviceIndex, bool
bEnabled)
```

Enables or disables mapping of the Depth image in RGB space on the device specified by `deviceIndex`. When enabled, `PsGetFrame` can be invoked passing `PsMappedDepthFrame` as the frame type, to get the Depth frame that is mapped to RGB space. The resolution of the mapped Depth frame is the same as that of the RGB image.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device on which to enable or disable mapping. Device indices range from 0 to device count – 1.
`bEnabled`	`bool`	Set to `true` to enable the feature or `false` to disable the feature.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.49 PsSetMapperEnabledRGBToIR

```
PsReturnStatus PsSetMapperEnabledRGBToIR(int32_t deviceIndex, bool
bEnabled)
```

Enables or disables mapping of the IR image in RGB space on the device specified by `deviceIndex`. When enabled, `PsGetFrame` can be invoked passing `PsMappedIRFrame` as the frame type, to get the IR frame that is mapped to RGB space. The resolution of the mapped IR frame is the same as that of the RGB image. Note: this can only take effect when there also have Depth frames.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device on which to enable or disable mapping. Device indices range from 0 to device count – 1.
`bEnabled`	`bool`	Set to `true` to enable the feature or `false` to disable the feature.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.50 PsSetProperty

```
PsReturnStatus PsSetProperty(int32_t deviceIndex, PsPropertyType
propertyType, void* pData, int32_t dataSize)
```

Sets a property value for the device specified by `deviceIndex`.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device on which to set the property. Device indices range from 0 to device count – 1.
`propertyType`	`PsPropertyType`	The type of property to set on the device.
`pData`	Void pointer	Pointer to a buffer containing the property value.
`dataSize`	32-bit integer	The size, in bytes, of the property value contained in `pData`.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.51 PsSetPulseCount

```
PsReturnStatus PsSetPulseCount(int32_t deviceIndex, uint16_t pulseCount)
```

Sets the pulse count for the device specified by `deviceIndex`.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device on which to set the pulse count. Device indices range from 0 to device count – 1.
`pPulseCount`	16-bit unsigned integer	The pulse count value to set.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.52 PsSetResolution

```
PsReturnStatus PsSetResolution(int32_t deviceIndex, PsResolution
resolution)
```

Sets the RGB frame resolution on a device specified by <code>deviceIndex</code>.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device on which to set the pulse count. Device indices range from 0 to device count – 1.
`resolution`	PsResolution	Pointer to a variable in which to store the returned resolution.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.53 PsSetRGBDistortionCorrectionEnabled

```
PsReturnStatus PsSetRGBDistortionCorrectionEnabled (int32_t deviceIndex,
bool bEnabled)
```

Enables or disables the RGB distortion correction feature on the device specified by `deviceIndex`.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device on which to enable or disable the RGB distortion correction feature. Device indices range from 0 to device count – 1.
`bEnabled`	`bool`	Set to `true` to enable the feature or `false` to disable the feature.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.54 PsSetSmoothingFilterEnabled

```
PsReturnStatus PsSetSmoothingFilterEnabled(int32_t deviceIndex, bool
bEnabled)
```

Enables or disables the Smoothing Filter feature.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device on which to enable or disable the RGB distortion correction feature. Device indices range from 0 to device count – 1.
`bEnabled`	`bool`	Set to `true` to enable the feature or `false` to disable the feature.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.55 PsSetSpatialFilterEnabled

```
PsReturnStatus PsSetSpatialFilterEnabled(int32_t deviceIndex, bool
bEnabled)
```

Enables or disables the Spatial Filter feature.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device on which to enable or disable the RGB distortion correction feature. Device indices range from 0 to device count – 1.
`bEnabled`	`bool`	Set to `true` to enable the feature or `false` to disable the feature.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.56 PsSetSynchronizeEnable

```
PsSetSynchronizeEnabled(int32_t deviceIndex, bool bEnabled)
```

Enables or disables the RGB and Depth frame synchronize feature.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device on which to enable or disable the feature. Device indices range from 0 to device count – 1.
`bEnabled`	`bool`	Set to `true` to enable the feature or `false` to disable the feature.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.57 PsSetThreshold

`PsSetThreshold(int32_t deviceIndex, uint16_t threshold)`

Sets the threshold value for the background filter on the device specified by `deviceIndex`. The value represents the cut-off point for distant data that the filter should ignore. For example, if `20.0` is specified, data with 20% or less confidence will be dropped.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device on which to set the threshold. Device indices range from 0 to device count – 1.
`threshold`	16-bit unsigned integer	The threshold value to set.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.58 PsSetTimeFilterEnabled

`PsReturnStatus PsSetTimeFilterEnabled(int32_t deviceIndex, bool bEnabled)`

Enables or disables the Time Filter feature.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device on which to enable or disable the feature.

		Device indices range from 0 to device count – 1.
`bEnabled`	`bool`	Set to `true` to enable the feature or `false` to disable the feature.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.59 PsSetWDROutputMode

```
PsReturnStatus PsSetWDROutputMode(int32_t deviceIndex, PsWDROutputMode*
pWDRMode)
```

Sets the WDR output mode on the device specified by `deviceIndex`.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device on which to set WDR output mode. Device indices range from 0 to device count – 1.
`pWDROutputMode`	`PsWDROutputMode`	Pointer to a `PsWDROutputMode` variable specifying the mode to set.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.60 PsSetWDRStyle

```
PsReturnStatus PsSetWDRStyle(int32_t deviceIndex, PsWDRStyle wdrStyle)
```

Sets the WDR style on the device specified by `deviceIndex`.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device on which to set the WDR style. Device indices range from 0 to device count – 1.
`wdrStyle`	`PsWDRStyle`	The style to set.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.

7.3.61 PsSetWDRFusionThreshold

```
PsReturnStatus PsSetWDRFusionThreshold(int32_t deviceIndex, uint16_t
threshold1, uint16_t threshold2 = 0)
```

Sets the fusion threshold for WDR mode on the device specified by `deviceIndex`.

Parameters

Name	Type	Description
`deviceIndex`	32-bit integer	The index of the device on which to set the fusion threshold. Device indices range from 0 to device count – 1.
`threshold1`	16-bit unsigned integer	The first threshold value.
`threshold2`	16-bit unsigned integer	The second threshold value.

Returns

`PsRetOK` if the function succeeded, or one of the error values defined by `PsReturnStatus`.