

DepthVista

# DepthVista SDK API Manual



Version 1.11

e-con Systems

3/16/2024

### **Disclaimer**

e-con Systems reserves the right to edit/modify this document without any prior intimation of whatsoever.

# Contents

<b>INTRODUCTION TO DEPTHVISTA</b>	<b>5</b>
<b>DESCRIPTION</b>	<b>5</b>
<b>PREREQUISITE</b>	<b>5</b>
<b>INSTALLING DEPTHVISTASDK IN LINUX</b>	<b>6</b>
<b>SUPPORTED APIS</b>	<b>7</b>
<b>RESULT INITIALIZE()</b>	<b>7</b>
<b>RESULT DEINITIALIZE()</b>	<b>7</b>
<b>RESULT GETDEVICECOUNT(UINT32_T* DEVICECOUNT)</b>	<b>8</b>
<b>RESULT GETDEVICELISTINFO(UINT32_T DEVICECOUNT, DEVICEINFO* DEVICESLIST)</b>	<b>9</b>
<b>RESULT OPENDevice(DEVICEINFO DEVINFO, DEVICEHANDLE* DEVHANDLE)</b>	<b>10</b>
<b>RESULT CLOSEDevice(DEVICEHANDLE DEVHANDLE)</b>	<b>11</b>
<b>RESULT ISOPENED(DEVICEHANDLE DEVHANDLE)</b>	<b>11</b>
<b>RESULT GETNEXTFRAME(DEVICEHANDLE DEVHANDLE)</b>	<b>12</b>
<b>RESULT GETTOFFRAME(DEVICEHANDLE DEVHANDLE, FRAMETYPE FRAMETYPE, TOFFRAME* FRAME)</b>	<b>13</b>
<b>RESULT GETFRAMES(DEVICEHANDLE DEVHANDLE, FRAMES* FRAMES)</b>	<b>14</b>
<b>INT GETFRAMESPERSECOND(DEVICEHANDLE DEVHANDLE)</b>	<b>16</b>
<b>RESULT SETDATA MODE(DEVICEHANDLE DEVHANDLE, DATA MODE DATA MODE)</b>	<b>17</b>
<b>RESULT GETDATA MODE(DEVICEHANDLE DEVHANDLE, DATA MODE* DATA MODE)</b>	<b>18</b>
<b>RESULT SETDEPTH RANGE(DEVICEHANDLE DEVHANDLE, UINT16_T DEPTH RANGE)</b>	<b>19</b>
<b>RESULT GETDEPTH RANGE(DEVICEHANDLE DEVHANDLE, UINT16_T* DEPTH RANGE)</b>	<b>20</b>
<b>RESULT SETTOFCORING(DEVICEHANDLE DEVHANDLE, UINT16_T TOFCORE)</b>	<b>21</b>
<b>RESULT GETTOFCORING(DEVICEHANDLE DEVHANDLE, UINT16_T* TOFCORE)</b>	<b>22</b>
<b>RESULT SETTOFIRGAIN(DEVICEHANDLE DEVHANDLE, UINT16_T TOFIRGAIN)</b>	<b>23</b>
<b>RESULT GETTOFIRGAIN(DEVICEHANDLE DEVHANDLE, UINT16_T* TOFIRGAIN)</b>	<b>24</b>
<b>RESULT GETUVCCONTROL(DEVICEHANDLE DEVHANDLE, INT32_T CONTROLID, UVCPROP* CONTROLVALUE)</b>	<b>24</b>
<b>RESULT SETUVCCONTROL(DEVICEHANDLE DEVHANDLE, INT32_T CONTROLID, INT32_T CONTROLVALUE)</b>	<b>26</b>
<b>RESULT SETIMU CONFIG(DEVICEHANDLE DEVHANDLE, IMU CONFIG_TYPEDEF IMU CONFIG)</b>	<b>27</b>
<b>RESULT GETIMU CONFIG(DEVICEHANDLE DEVHANDLE, IMU CONFIG_TYPEDEF* IMU CONFIG)</b>	<b>28</b>
<b>RESULT CONTROLIMU CAPTURE(DEVICEHANDLE DEVHANDLE, IMU DATA INPUT_TYPEDEF* IMU INPUT)</b>	<b>29</b>
<b>RESULT GETIMU VALUE(DEVICEHANDLE DEVHANDLE, IMU DATA INPUT_TYPEDEF* IMU AXES)</b>	<b>30</b>
<b>RESULT GETANTI FLICKER DETECTION(DEVICEHANDLE DEVHANDLE, UINT8_T* AFDetect)</b>	<b>30</b>
<b>RESULT SETANTI FLICKER DETECTION(DEVICEHANDLE DEVHANDLE, UINT8_T AFDetect)</b>	<b>32</b>
<b>RESULT GETSCENE MODE(DEVICEHANDLE DEVHANDLE, UINT8_T* SCENE MODE)</b>	<b>33</b>
<b>RESULT SETSCENE MODE(DEVICEHANDLE DEVHANDLE, UINT8_T SCENE MODE)</b>	<b>34</b>
<b>RESULT GETSPECIAL EFFECT(DEVICEHANDLE DEVHANDLE, UINT8_T* SPLEFFECT)</b>	<b>35</b>
<b>RESULT SETSPECIAL EFFECT(DEVICEHANDLE DEVHANDLE, UINT8_T SPLEFFECT)</b>	<b>36</b>
<b>RESULT GETDENOISE(DEVICEHANDLE DEVHANDLE, UINT8_T* DENOISE)</b>	<b>37</b>

RESULT SETDENOISE(DEVICEHANDLE DEVHANDLE, UINT8_T DENOISE)	38
RESULT GETAUTOEXPOSUREROI(DEVICEHANDLE DEVHANDLE, UINT8_T* ROIMODE, UINT8_T* GWINSIZE)	39
RESULT SETAUTOEXPOSUREROI(DEVICEHANDLE DEVHANDLE, UINT8_T ROIMODE, UINT32_T WIDTH, UINT32_T HEIGHT, UINT32_T XCOR, UINT32_T YCOR, UINT8_T WINSIZE)	41
RESULT GETORIENTATION(DEVICEHANDLE DEVHANDLE, UINT8_T* ORIENTATION)	42
RESULT SETORIENTATION(DEVICEHANDLE DEVHANDLE, UINT8_T ORIENTATION)	44
RESULT GETFACEDETECTION(DEVICEHANDLE DEVHANDLE, UINT8_T* FACEDET, UINT8_T* STATUSSTRUCT, UINT8_T* OVERLAYRECT)	45
RESULT SETFACEDETECTION(DEVICEHANDLE DEVHANDLE, UINT8_T FACEDET, UINT8_T STATUSSTRUCT, UINT8_T OVERLAYRECT)	46
RESULT GETSMILEDETECTION(DEVICEHANDLE DEVHANDLE, UINT8_T* SMILEDET, UINT8_T* STATUSSTRUCT)	47
RESULT SETSMILEDETECTION(DEVICEHANDLE DEVHANDLE, UINT8_T SMILEDET, UINT8_T STATUSSTRUCT)	49
RESULT GETIMUEMBEDDEDATA(DEVICEHANDLE DEVHANDLE, UINT8_T* IMUDATA)	50
RESULT SETIMUEMBEDDEDATA(DEVICEHANDLE DEVHANDLE, UINT8_T IMUDATA)	51
RESULT GETEXPOSURECOMPENSATION(DEVICEHANDLE DEVHANDLE, UINT32_T* EXPOCOMP)	52
RESULT SETEXPOSURECOMPENSATION(DEVICEHANDLE DEVHANDLE, UINT32_T EXPOCOMP)	53
RESULT GETFRAMERATECTRL(DEVICEHANDLE DEVHANDLE, UINT8_T* FRAMERATECTRL)	53
RESULT SETFRAMERATECTRL(DEVICEHANDLE DEVHANDLE, UINT8_T FRAMERATECTRL)	54
RESULT SETDEFAULT(DEVICEHANDLE DEVHANDLE)	55
RESULT GETUNIQUEID(DEVICEHANDLE DEVHANDLE, UINT64_T* UNIQUEID)	56
RESULT SETAVGREGION(DEVICEHANDLE DEVHANDLE, AVGREGION REGION)	56
RESULT SETFILTERTYPE(DEVICEHANDLE DEVHANDLE, INT CTRLID, BOOL SELECTED)	57
RESULT SETDEPTHPOS(DEVICEHANDLE DEVHANDLE, DEPTHPTR POS)	58
RESULT SETCURSORCOLOR(DEVICEHANDLE DEVHANDLE, INT COLOR)	59
RESULT SETUNDISTORTION(DEVICEHANDLE DEVHANDLE, INT UNDISTORT)	59
RESULT SETRGBDMAPPING (DEVICEHANDLE DEVHANDLE, INT RGBDMAPPING)	60
RESULT SETPLANARIZATION(DEVICEHANDLE DEVHANDLE, INT PLANARIZE)	61
RESULT RESULT SETFLYINGPIXELFILTER(DEVICEHANDLE DEVHANDLE, INT STATE)	62
RESULT GETDEPTHIRVALUES(DEVICEHANDLE DEVHANDLE, INT* AVGDEPTH, INT* STDDEPTH, INT* AVGIR, INT* STDIR)	63
VOID REGISTERFRAMECALLBACK(DEVICEHANDLE DEVHANDLE, FUNCTION<VOID(INT)> CB)	64
VOID REGISTERNOTIFICATIONCALLBACK(DEVICEHANDLE DEVHANDLE, FUNCTION<VOID(INT)> CB)	64
RESULT UPDATEAVGXANDY(DEVICEHANDLE DEVHANDLE, INT AVG_X, INT AVG_Y)	65
RESULT UPDATECOLORMAP(DEVICEHANDLE DEVHANDLE, INT MIN, INT MAX, INT COLORMAP)	66
RESULT SETAVGIRDISPLAY(DEVICEHANDLE DEVHANDLE, UINT8_T AVGIRDISPLAY)	67
VOID PERROR(STD::STRING MESSAGE = "")	67
RESULT READFIRMWAREVERSION(DEVICEHANDLE DEVHANDLE, UINT8_T* MAJORVERSION, UINT8_T* MINORVERSION1, UINT16_T* MINORVERSION2, UINT16_T* MINORVERSION3)	68
RESULT CALIBREADREQDEPTHINTRINSIC(DEVICEHANDLE DEVHANDLE, INT* DEPTHINTFILELENGTH)	69
RESULT CALIBREADDEPTHINTRINSIC(DEVICEHANDLE DEVHANDLE, INT* DEPTHINTFILELENGTH, UNSIGNED CHAR* DEPTHINTRINSICBUFFER)	70
RESULT CALIBREADREQRGBINTRINSIC (DEVICEHANDLE DEVHANDLE, INT* RGBINTFILELENGTH)	71
RESULT CALIBREADRGBINTRINSIC(DEVICEHANDLE DEVHANDLE, INT* RGBINTFILELENGTH, UNSIGNED CHAR* RGBINTRINSICBUFFER)	72
RESULT CALIBREADREQEXTRINSIC (DEVICEHANDLE DEVHANDLE, INT* EXTFILELENGTH)	73

<b>RESULT CALIBREADINTRINSIC(DEVICEHANDLE DEVHANDLE, INT* EXTFILELENGTH, UNSIGNED CHAR* INTRINSICBUFFER)</b>	<b>74</b>
<b>RESULT GETSDKVERSION (UINT8_T* MAJORVERSION, UINT8_T* MINORVERSION1, UINT16_T* MINORVERSION2)</b>	<b>75</b>
<b>RESULT SETLOGLEVEL (LOGLEVEL LOGSTATE)</b>	<b>75</b>
<b>RESULT SETLOGGING(LOGGING LOG)</b>	<b>76</b>
<b>RESULT GETDEVICECALIBRATIONPARAMS(DEVICEHANDLE DEVHANDLE, CALIBRATIONPARAMS* CALIBPARAMS)</b>	<b>77</b>
<b>ENUM</b>	<b>78</b>
<b>DATA MODE</b>	<b>78</b>
<b>LOG LEVEL</b>	<b>79</b>
<b>LOGGING</b>	<b>79</b>
<b>CAMERA MODE</b>	<b>79</b>
<b>DEPTH RANGE</b>	<b>80</b>
<b>FRAME TYPE</b>	<b>80</b>
<b>RESULT</b>	<b>81</b>
<b>AVG REGION</b>	<b>82</b>
<b>UVC PROP ID</b>	<b>83</b>
<b>CALIB VALID FLAG</b>	<b>84</b>
<b>STRUCTURE</b>	<b>85</b>
<b>DEVICE INFO</b>	<b>85</b>
<b>INTRINSIC CALIB PARAMS</b>	<b>85</b>
<b>EXTRINSIC CALIB PARAMS</b>	<b>86</b>
<b>CALIBRATION PARAMS</b>	<b>86</b>
<b>DEVICE HANDLE</b>	<b>87</b>
<b>UVC PROP</b>	<b>87</b>
<b>DEPTH PTR</b>	<b>88</b>
<b>TOFF FRAME</b>	<b>88</b>
<b>FRAMES</b>	<b>89</b>
<b>IMU CONFIG_TYPEDEF</b>	<b>89</b>
<b>IMU DATA INPUT_TYPEDEF</b>	<b>91</b>
<b>IMU DATA OUTPUT_TYPEDEF</b>	<b>92</b>
<b>FAQ</b>	<b>93</b>
<b>SUPPORT</b>	<b>94</b>

# Introduction to DepthVista

---

DepthVista is a 3D camera based on Time of Flight (TOF) technology, USB Video Class (UVC) compliant, USB 3.2 Gen 1 SuperSpeed USB camera from e-con Systems, which has over two decades of experience in designing, developing, and manufacturing OEM cameras.

DepthVista is an RGB-D camera contains both RGB and TOF depth cameras. RGB camera has 1/2.6" AR0234CS CMOS digital image sensor with global shutter from onsemi™. It has dedicated high performance color Image signal processor. TOF depth camera has 1/4" CCD sensor and dedicated depth processor. DepthVista is a two-board solution containing camera board with the USB 3.2 Gen 1 interface and Laser board along with enclosure.

## Description

DepthVista has USB interface controller with USB Type-C connector to interface with the host PC. It is a ready-to-manufacture camera board with all the necessary firmware built-in and is compatible with the UVC version 1.0 standard. You can integrate this camera into the products, and this helps to cut short the time-to-market.

DepthVista is UVC compatible and will work with the standard drivers available with Windows and Linux OS. There is no need for any additional driver installation. So, video streaming through UVC is possible without any special drivers on OSes that have built-in support for UVC standards.

This document highlights the usage of APIs, Enums and Structures supported in DepthVista SDK.

## Prerequisite

Any version of OpenCV above 4.5.4 can be used along with DepthVistaSDK if required.

# Installing DepthVistaSDK in Linux

---

This section describes the installation of DepthVistaSDK in linux.

The steps to install the DepthVistaSDK are as follows:

1. Run the following command to extract the **package** file.

```
unzip -X <packageName.zip>
```

<Extracted Directory>\Linux\Bin\Ubuntu20.04\x64\SDK will have an install.sh file.

**Note:** For Ubuntu 22.04 the install.sh file will be present in <Extracted Directory>\Linux\Bin\Ubuntu22.04\x64\SDK.

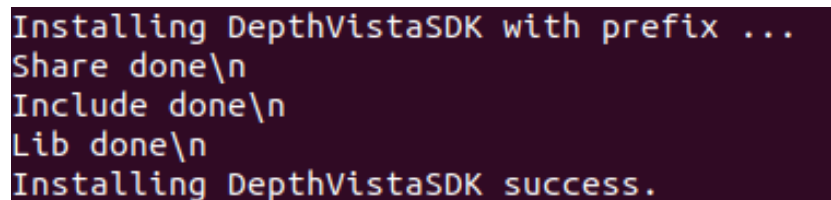
2. Open the folder containing install.sh in terminal.
3. Run the following command to give executable permission for install.sh file.

```
chmod +x install.sh
```

4. Run the following command to install the **DepthVistaSDK**.

```
sudo ./install.sh
```

Once installation is success, **Installation DepthVistaSDK success** message appears.



```
Installing DepthVistaSDK with prefix ...  
Share done\  
Include done\  
Lib done\  
Installing DepthVistaSDK success.
```

Figure 1: Installation success screenshot.

# Supported APIs

The details and usage of the supported APIs are explained below.

## Result Initialize()

This function initializes all the needed resources, and it must be called before calling any other APIs in DepthVista SDK.

Parameters	Description
Nil	N/A

### Return:

- Returns **Ok** when the device is initialized successfully.
- Returns **NotInitialized** when the device is not initialized successfully.

### Sample Code:

```
void InitializeCam()  
{  
    if (Initialize())>0  
    {  
        printf("Initialize success");  
    }  
    else  
    {  
        printf("Initialize Failed");  
    }  
}
```

## Result Deinitialize()

This function de-initialize all the API in DepthVista SDK and clear all resources allocated. After invoking this API, no other APIs can be invoked.

Parameters	Description
Nil	N/A

### Return:

- Returns **Ok** when the device is de-initialized successfully.
- Returns **NotDeInitialized** when the device is not de-initialized.



### Sample Code:

```
void DeinitializeCam()
{
    if(Deinitialize()>0)
    {
        printf("Deinitialize Success\r\n");
    }
    else
    {
        printf("Deinitialize failed\r\n");
    }
}
```

## Result GetDeviceCount(uint32\_t\* deviceCount)

This function is used to get the number of DepthVista devices connected to the host PC. The device count will be stored in the given 32-bit integer pointer.

Parameters	Description
uint32_t* deviceCount	[OUT] Pointer to store device count.

### Return:

- Returns **Ok** when the number of devices is obtained successfully.
- Returns **NotInitialized** when the device is not initialized.
- Returns **NoDeviceConnected** when there are no devices connected.

### Sample Code:

```
void getDeviceCount()
{
    uint32_t deviceCount;
    if(GetDeviceCount(&deviceCount)>0)
    {
        printf("Number of devices connected is %d\r\n\r\n", deviceCount );
    }
    else
    {
        printf("GetDeviceCount Failed\r\n\r\n");
    }
}
```

## Result GetDeviceListInfo(uint32\_t deviceCount, DeviceInfo\* devicesList)

This function returns the list of information of all DepthVista devices connected to the host PC. The information is contained in a pointer pointing to the array of *DeviceInfo* structure. DeviceInfo structure holds the device name, VID, PID, device path and the serial number of the camera.

Parameters	Description
uint32_t deviceCount	[IN] Number of devices connected.
DeviceInfo* devicesList	[OUT] Array of DeviceInfo whose size is deviceCount.

### Return:

- Returns **Ok** when the information of all the devices connected was obtained.
- Returns **NotInitialized** when the device is not initialized.
- Returns **NoDeviceConnected** when there are no devices connected.
- Returns **InvalidDeviceIndex** when the device index is invalid.

### Sample Code:

```
void GetDeviceListInfo_()
{
    uint32_t deviceCount; // get device count from
    GetDeviceCount

    DeviceInfo* devicesList = new
    DeviceInfo[deviceCount];

    if(GetDeviceListInfo(deviceCount, devicesList )>0)
    {
        uint16_t index;
        for(index = 0; index < deviceCount; index++)
        {
            printf("Device name is %s",
            devicesList[index].deviceName);

            printf("Device PID is %s",
            devicesList[index].pid);

            printf("Device VID is %s",
            devicesList[index].vid);

            printf("Device devicePath is %s",
            deviceList[index].devicePath);

            printf("Device serialNo is %s",
            deviceList[index].serialNo);

        }
    }
    else
```

```

    {
        printf("GetDeviceListInfo failed\r\n");
    }
}

```

## Result **OpenDevice(DeviceInfo devInfo, DeviceHandle\* devHandle)**

This function opens the device specified by devInfo and return a DeviceHandle in the devHandle parameter. This handle grants exclusive access to the device and will be used as parameter in other API calls. The device must be subsequently closed using CloseDevice().

Parameters	Description
DeviceInfo devInfo	[IN] Device information of the device which is to be opened.
DeviceHandle* devHandle	[OUT] Device Handle of the device which has been opened

### Return:

- Returns **Ok** when the device corresponding to the devInfo has been opened successfully.
- Returns **InvalidNode** if device has been disconnected but tried to open.
- Returns **CameraAlreadyOpen** if the device is already opened.
- Returns **NotInitialized** when the device is not initialized.
- Returns **NoDeviceConnected** when there are no devices connected.

### Sample Code:

```

void OpenDevice_()
{
    DeviceInfo info; //DeviceInfo that we got from
    GetDeviceListInfo()
    DeviceHandle handle;
    if(OpenDevice(info, &handle) >0) //The handle is
    used as input parameter for other APIs
    {
        printf("OpenDevice Success\r\n");
    }
    else
    {
        printf("OpenDevice failed\r\n");
    }
}

```

## Result CloseDevice(DeviceHandle devHandle)

This function closes the device that was opened using OpenDevice().

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device which has been closed

### Return:

- Returns **Ok** when the device is closed successfully.
- Returns **CameraNotOpened** when the device is not opened.

### Sample Code:

```
void CloseDevice_()
{
    DeviceHandle handle; //Handle we got from
    OpenDevice()
    if(CloseDevice(handle)>0 )
    {
        printf("CloseDevice Success\r\n");
    }
    else
    {
        printf("CloseDevice failed\r\n");
    }
}
```

## Result IsOpened(DeviceHandle devHandle)

This function return whether the device corresponding to the given devHandle is opened or not.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device which must checked for open.

### Return:

- Returns **Ok** when the device is opened.
- Returns **CameraNotOpened** when the camera is not opened.

### Sample Code:

```
void IsOpened_()
{
```

```

    DeviceHandle handle; //Handle we got from
OpenDevice()
    if(IsOpened(handle)>0 )
    {
        printf("Device is opened\r\n");
    }
    else
    {
        printf("Device is closed \r\n");
    }
}

```

## Result GetNextFrame(DeviceHandle devHandle)

This function retrieves frame data from the device that is corresponding to the given devHandle. This API must be invoked before retrieving frame data using GetToFFrame().

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.

### Return:

- Returns **Ok** when the image frame is retrieved successfully.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **TimeoutError** when the wait event timed out.
- Returns **FrameSeparationFailed** when frame separation fails.

### Sample Code:

```

void GetNextFrame_()
{
    DeviceHandle handle; //Handle we got from
OpenDevice()
    if(GetNextFrame(handle)>0)
    {
        printf("GetNextFrame success\r\n");
        // call to GetToFFrame();
    }
    else
    {
        printf("GetNextFrame failed\r\n");
    }
}

```

```
}
```

## Result GetToFFrame(DeviceHandle devHandle, FrameType frameType, ToFFrame\* frame)

This function retrieves the current frame with the data specified by *FrameType*, from the device that is corresponding to devHandle. Before invoking this API, invoke GetNextFrame() to receive frame from the device.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
FrameType frameType	[IN]Type of frame for which the image data is needed. The frame types supported are as follows: <ul style="list-style-type: none"> <li>• <b>IRPreviewFrame</b>: Separate IR frame.</li> <li>• <b>DepthColorMap</b>: Depth data that is applied with Color map for preview purpose.</li> <li>• <b>RGBFrame</b>: Separate RGB frame.</li> <li>• <b>DepthRawFrame</b>: Raw depth frame without applying color map. Hence this frame cannot be used for preview.</li> </ul>
ToFFrame* Frame	[OUT]Address of the ToFFrame structure in which the image data is filled. The structure member are as follows: <ul style="list-style-type: none"> <li>• <b>unsigned char* frame_data</b>: This unsigned char pointer holds the address of the memory that holds the frame data.</li> <li>• <b>uint16_t width</b>: This holds the width of the frame.</li> <li>• <b>uint16_t height</b>: This holds the height of the frame.</li> <li>• <b>uint8_t pixel_format</b>: 0 for UYVY, 1 for Y16 and 2 for RGB pixel format.</li> <li>• <b>uint32_t size</b>: This holds the size of the image buffer in bytes.</li> </ul>

### Return:

- Returns **Ok** when the TOFFrame structure is retrieved successfully.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **TimeoutError** when the wait event timed out.
- Returns **InvalidFrameType** when the mentioned frame type is invalid.

### Sample Code:

```
void GetToFFrame_()
{
```

```

ToFFrame frame;
uint8_t frameType;
DeviceHandle handle; //Handle we got from
OpenDevice()
if (GetToFFrame(handle, frameType, &frame)>0)
{
    // Usage of ToFFrame struct
    uint16_t* image_buf =
(uint16_t*)malloc(frame.size);
    memcpy(image_buf, frame.frame_data,
frame.size);
    printf("frame width is %d\r\n", frame.width);
    printf("frame height is %d\r\n",
frame.height);
    switch(frame.pixel_format)
    {
        Case 0:
            printf("Pixel format is UYVY");
        Case 1:
            printf("Pixel format is Y16");
        Case 2:
            printf("Pixel format is RGB");
    }
}
else
{
    printf("GetTofFrame failed\r\n");
}
}

```

## Result GetFrames(DeviceHandle devHandle, Frames\* frames)

This function can retrieve the frame data of all *FrameTypes* - RGB, IR, Raw Depth and Depth Color Map from the device that is corresponding to devHandle. Before invoking this API, invoke GetNextFrame() to receive frame from the device.

**Note:** Data in Frames structure will be filled based on the current DataMode. For example, if current DataMode is Depth\_Mode, only raw\_depth frame and depth\_colormap will contain data and rest of the frame structure will be a NULL.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
Frames* frames	[OUT] Address of the Frames structure. The structure member are as follows: <ul style="list-style-type: none"> <li>• <b>ToFFrame rgb</b> : ToFFrame structure with RGB frame data.</li> <li>• <b>ToFFrame ir</b> : ToFFrame structure with IR frame data.</li> <li>• <b>ToFFrame raw_depth</b> : ToFFrame structure with DepthRawFrame frame data.</li> <li>• <b>ToFFrame depth_colormap</b>: ToFFrame structure with DepthColorMap frame data.</li> </ul>

#### Return:

- Returns **Ok** when the Frames structure is retrieved successfully.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **TimeoutError** when the wait event timed out.

#### Sample Code:

```
void GetFrames_()
{
    Frames frames;
    DeviceHandle handle; //Handle we got from
    OpenDevice()
    if(GetFrames(handle, &frames)>0)
    {
        // Usage of Frames struct
        // RGB frame data
        uint16_t* rgb_image_buf =
        (uint16_t*)malloc(frames.rgb.size);
        memcpy(rgb_image_buf, frames.rgb.frame_data,
        frames.rgb.size);
        printf("frame width is %d\r\n",
        frames.rgb.width);
        printf("frame height is %d\r\n",
        frames.rgb.height);

        // IR frame data
```



```

        uint16_t* ir_image_buf =
        (uint16_t*)malloc(frames.ir.size);

        memcpy(ir_image_buf, frames.ir.frame_data,
frames.ir.size);

        printf("frame width is %d\r\n",
frames.ir.width);

        printf("frame height is %d\r\n",
frames.ir.height);


        //DepthColorMap frame data
        uint16_t* depth_image_buf =
        (uint16_t*)malloc(frames.depth_colormap.size);

        memcpy(depth_image_buf,
frames.depth_colormap.frame_data,
frames.depth_colormap.size);

        printf("frame width is %d\r\n",
frames.depth_colormap.width);

        printf("frame height is %d\r\n",
frames.depth_colormap.height);
    }
    else
    {

        printf("GetTofFrame failed\r\n");

    }
}

```

## Int GetFramesPerSecond(DeviceHandle devHandle)

This function returns the number of frames obtained per second in the device that is corresponding to the given devHandle.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.

### Return:

- Returns FPS of the device corresponding to devHandle.
- Returns **CameraNotOpened** when the device is not opened

### Sample Code:

```

void GetFramesPerSecond_()
{

```

```
DeviceHandle handle; //Handle we got from
OpenDevice()

printf("FPS : %d\r\n", getFramesPerSecond(handle));
}
```

## Result SetDataMode(DeviceHandle devHandle, DataMode dataMode)

This function sets the stream mode specified by dataMode in the device that is corresponding to the given devHandle. All the supported data modes are listed in the enum *DataMode*.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
DataMode dataMode	<p>[IN]Stream Mode that is to be set. Can we any value from the enum <i>DataMode</i>. The datamodes supported are as follows:</p> <ul style="list-style-type: none"> <li>• <b>Depth_IR_Mode:</b> Output depth frame (640 x 480) and IR frame (640 x 480) at 30 FPS.</li> <li>• <b>Depth_Mode:</b> Output depth frame(640x480) at 30 FPS.</li> <li>• <b>IR_Mode:</b> Output IR frame (640 x 480) at 30 FPS.</li> <li>• <b>Depth_IR_RGB_VGA_Mode:</b> Output depth frame (640 x 480), IR frame (640 x 480) and RGB frame (640 x 480) at 30 FPS.</li> <li>• <b>Depth_IR_RGB_HD_Mode:</b> Output depth frame (640 x 480), IR frame (640 x 480) and RGB frame (1280 x 720) at 30 FPS.</li> <li>• <b>RGB_VGA_Mode:</b> Output RGB frame (640 x 480) at 60 FPS.</li> <li>• <b>RGB_HD_Mode:</b> Output RGB frame (1280 x 720) at 60 FPS.</li> <li>• <b>RGB_Full_HD_Mode:</b> Output RGB frame (1920 x 1080) at 30 FPS.</li> <li>• <b>RGB_1200p_Mode:</b> Output RGB frame (1920 x 1200) at 30 FPS.</li> </ul>

### Return:

- Returns **Ok** when the Data mode is set successfully.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **SystemCallFail** when the system call API failed.
- Returns **TimeoutError** when the wait event timed out.

### Sample Code:

```
void SetDataMode_()
{
```

```

    DeviceHandle handle; //Handle we got from
    OpenDevice()

    DataMode dataMode;
    if(SetDataMode(handle, dataMode)>0)
    {
        printf("SetDataMode success\r\n");
    }
    else
    {
        printf("SetDataMode failed\r\n");
    }
}

```

## Result GetDataMode(DeviceHandle devHandle, DataMode\* dataMode)

This function gets the stream mode in which the device is streaming currently from the device that is corresponding to the given devHandle. All the supported data modes are listed in the enum *DataMode*.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
DataMode* dataMode	[OUT]dataMode is filled with the stream mode in which the device is streaming currently.

### Return:

- Returns **Ok** when the Data mode is obtained successfully.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **SystemCallFail** when the system call API failed.

### Sample Code:

```

void GetDataMode_()
{
    DataMode* dataMode;
    DeviceHandle handle; //Handle we got from
    OpenDevice()
    if(GetDataMode(handle, dataMode)>0)
    {
        switch(dataMode)
        {
            case Depth_IR_Mode:

```

```

        printf("Current data modes is
Depth_IR_Mode\r\n");
        case Depth_Mode:
            printf("Current data modes is
Depth_Mode\r\n");
        case IR_Mode:
            printf("Current data modes is
IR_Mode\r\n");
        case Depth_IR_RGB_VGA_Mode:
            printf("Current data modes is
Depth_IR_RGB_VGA_Mode\r\n");
        case Depth_IR_RGB_HD_Mode:
            printf("Current data modes is
Depth_IR_RGB_HD_Mode\r\n");
        case RGB_VGA_Mode:
            printf("Current data modes is
RGB_VGA_Mode\r\n");
        case Depth_Mode:
            printf("Current data modes is
RGB_HD_Mode\r\n");
        case Depth_Mode:
            printf("Current data modes is
RGB_Full_HD_Mode\r\n");
        case Depth_Mode:
            printf("Current data modes is
RGB_1200p_Mode\r\n");
    }
}
else
{
    printf("GetDataMode failed\r\n");
}
}

```

## Result SetDepthRange(DeviceHandle devHandle, uint16\_t depthRange)

This function sets the depth range specified by depthRange in the device that is corresponding to the given devHandle. The device supports two depth range as follows:

- **Far Mode:** Effective depth range in this mode is between 1000 mm to 6000 mm.

- **Near Mode:** Effective depth range in this mode is between 200 mm to 1200 mm.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
uint16_t depthRange	[IN]Depth Range that is to be set. 0 for Near mode. 1 for Far mode.

#### Return:

- Returns **Ok** when the depth range is set successfully.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **SystemCallFail** when the system call API failed.

#### Sample Code:

```
void SetDepthMode_()
{
    uint16_t depthRange;
    DeviceHandle handle; //Handle we got from
    OpenDevice()
    if (SetDepthRange(handle, depthRange)>0)
    {
        printf("SetDepthRange success\r\n");
    }
    else
    {
        printf("SetDepthRange failed\r\n");
    }
}
```

### Result GetDepthRange(DeviceHandle devHandle, uint16\_t\* depthRange)

This function gets the depth range in which the device is streaming currently from the device that is corresponding to the given devHandle. The device supports two depth range as follows:

- **Far Mode:** Effective depth range in this mode is between 1000 mm to 6000 mm.
- **Near Mode:** Effective depth range in this mode is between 200 mm to 1200 mm.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.

```
uint16_t*
depthRange
```

[OUT]depthRange is filled with the depth range in which the device is streaming currently.  
 0 for Near mode.  
 1 for Far mode.

#### Return:

- Returns **Ok** when the depth range is obtained successfully.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **SystemCallFail** when the system call API failed.

#### Sample Code:

```
void GetDepthMode_()
{
    uint16_t* depthRange;
    DeviceHandle handle; //Handle we got from
    OpenDevice()
    if(GetDepthRange(handle, depthRange)>0)
    {
        if(depthRange == 0)
            printf("The depth range is near mode\r\n");
        else
            printf("The depth range is fat mode\r\n");
    }
    else
    {
        printf("GetDepthRange failed\r\n");
    }
}
```

### Result SetTOFCoring(DeviceHandle devHandle, uint16\_t TOFCore)

This function sets the TOF coring value specified by TOFCore to the device that is corresponding to the given devHandle.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
uint16_t TOFCore	[IN]Coring value that is to be set. The range is from 0 to 16383.

#### Return:

- Returns **Ok** when the TOF Coring value is set successfully.

- Returns **CameraNotOpened** when the device is not opened.
- Returns **SystemCallFail** when the system call API failed.

#### Sample Code:

```
void SetTOFCoring_()
{
    uint16_t TOFCore;
    DeviceHandle handle; //Handle we got from
    OpenDevice()
    if(SetTOFCoring(handle, TOFCore)>0)
    {
        printf("SetTOFCoring success\r\n");
    }
    else
    {
        printf("SetTOFCoring failed\r\n");
    }
}
```

### Result GetTOFCoring(DeviceHandle devHandle, uint16\_t\* TOFCore)

This function gets the TOF coring value from the device that is corresponding to the given devHandle.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
uint16_t* TOFCore	[OUT]TOFCore is filled with the TOF coring value. It ranges from 0 to 16383.

#### Return Type:

- Returns **Ok** when the TOF Coring value is obtained successfully.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **SystemCallFail** when the system call API failed.

#### Sample API:

```
void GetTOFCoring_()
{
    uint16_t TOFCore;
    DeviceHandle handle; //Handle we got from
    OpenDevice()
    if(GetTOFCoring(handle, &TOFCore)>0)
```

```

{
    printf("The Coring value is %d\r\n",TOFCore);
}
else
{
    printf("GetTOFCoring failed\r\n");
}
}

```

## Result SetTOFIRGain(DeviceHandle devHandle, uint16\_t TOFIRGain)

This function sets the TOF IR gain value specified by TOFIRGain to the device that is corresponding to the given devHandle. The default value is 16.

TOF gain is a software gain added to the IR.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
uint16_t TOFIRGain	[IN]IR Gain value that is to be set. It ranges from 1 to 100.

### Return:

- Returns **Ok** when the TOF IR Gain value is set successfully.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **InvalidValue** when the given value is an invalid one.

### Sample Code:

```

void SetTOFIRGain_()
{
    uint16_t TOFIRGain;
    DeviceHandle handle; //Handle we got from
OpenDevice()
    if(SetTOFIRGain(handle, TOFIRGain)>0)
    {
        printf("SetTOFIRGain success\r\n");
    }
    else
    {
        printf("SetTOFIRGain failed\r\n");
    }
}

```



## Result GetTOFIRGain(DeviceHandle devHandle, uint16\_t\* TOFIRGain)

This function gets the TOF IR gain value from the device that is corresponding to the given devHandle. The default value is 16.

TOF gain is a software gain added to the IR.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
uint16_t* TOFIRGain	[OUT] TOFIRGain is filled with the TOF IR Gain value. It ranges from 1 to 100.

### Return:

- Returns **Ok** when the TOF IR Gain value is obtained successfully.
- Returns **CameraNotOpened** when the device is not opened.

### Sample Code:

```
void GetTOFIRGain_()
{
    uint16_t* TOFIRGain;
    DeviceHandle handle; //Handle we got from
    OpenDevice()
    if(GetTOFIRGain(handle, TOFIRGain)>0)
    {
        printf("TOF IR Gain value is %d\r\n",
TOFIRGain);
    }
    else
    {
        printf("GetTOFIRGain failed\r\n");
    }
}
```

## Result GetUVCControl(DeviceHandle devHandle, int32\_t controlID, UVCCProp\* controlValue)

This function gets the UVC control value controlValue of control specified by controlID from the device that is corresponding to the given devHandle. UVC controls supported by DepthVista are as follows:

- Brightness
- Contrast
- Saturation

- Gamma
- Gain
- Sharpness
- White Balance
- Exposure
- Power line frequency

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
int32_t controlID	[IN] controlID for the specific UVC Control. Each UVC control have a specific control ID listed in UVCProp enum.
UVCProp* controlValue	[OUT] pointer to the UVCProp structure. UVCProp structure contains ID, minimum value, maximum value, current value, step value and the default value

#### Return:

- Returns **Ok** when the UVC property is obtained successfully.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **NoPropertyValueGet** when the specified control cannot be obtained.

#### Sample Code:

```
void GetUVCControl_()
{
    int32_t controlID;
    UVCProp controlValue;
    DeviceHandle handle; //Handle we got from
OpenDevice()
    if (GetUVCControl(handle, controlID, &controlValue)>0
)
    {
        printf("controlValue.id : %d\r\n",
controlValue.id);
        printf("controlValue.cur : %d\r\n",
controlValue.cur);
        printf("controlValue.min : %d\r\n",
controlValue.min);
        printf("controlValue.max : %d\r\n",
controlValue.max);
        printf("controlValue.step : %d\r\n",
controlValue.step);
        printf("controlValue.default_val : %d\r\n",
controlValue.Default_val);
    }
}
```

```

else
{
    printf("GetUVCControl failed\r\n");
}
}

```

## Result SetUVCControl(DeviceHandle devHandle, int32\_t controlID, int32\_t controlValue)

This function sets the current value of the UVC control specified by controlID to the device that is corresponding to the given devHandle. UVC controls supported by DepthVista are as follows:

- Brightness
- Contrast
- Saturation
- Gamma
- Gain
- Sharpness
- White Balance
- Exposure
- Power line frequency

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
int32_t controlID	[IN] controlID for the specific UVC control. Each UVC control have a specific control ID listed UVCProp enum.
int32_t controlValue	[IN] current value that is to be set for the UVC control.

### Return Type:

- Returns **Ok** when UVC property is set successfully.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **NoPropertyValueSet** when the specified control cannot be set.

### Sample Code:

```

void SetUVCControl_()
{
    int32_t controlID;
    int32_t controlValue;
}

```

```

    DeviceHandle handle; //Handle we got from
OpenDevice()
    if(SetUVCControl(handle, controlId, controlValue)>0
)
    {
        printf("SetUVCControl success\r\n");
    }
    else
    {
        printf("SetUVCControl failed\r\n");
    }
}

```

## Result SetIMUConfig(DeviceHandle devHandle, IMUCONFIG\_TypeDef IMUConfig)

This function sets the IMU Configuration mode of the IMU in the device that is corresponding to the given devHandle.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
IMUCONFIG_TypeDef IMUConfig	[IN] pointer to the IMUCONFIG_TypeDef structure.

### Return:

- Returns **Ok** when the IMU Configuration mode is set successfully.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **NoPropertyValueSet** when the specified control cannot be set.
- Returns **SystemCallFail** when the system call API failed.
- Returns **HidWriteFail** when the buffer cannot be written to the hid handle.
- Returns **HidReadFail** when the buffer cannot be read from the hid handle.
- Returns **TimeoutError** when the wait event timed out.

### Sample Code:

```

void SetIMUConfig_()
{
    IMUCONFIG_TypeDef IMUConfig;
    DeviceHandle handle; //Handle we got from
OpenDevice()

    //Set the IMUConfig to the required Configuration
    if(SetIMUConfig(handle, IMUConfig)>0 )

```

```

{
    printf("SetIMUConfig success\r\n");
}
else
{
    printf("SetIMUConfig Failed\r\n");
}
}

```

## Result GetIMUConfig(DeviceHandle devHandle, IMUCONFIG\_TypeDef\* IMUConfig)

This function gets the IMU Configuration mode of the IMU from the device that is corresponding to the given devHandle.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
IMUCONFIG_TypeDef* IMUConfig	[OUT] pointer to the <i>IMUCONFIG_TypeDef</i> structure.

### Return:

- Returns **Ok** when the IMU Configuration mode is obtained successfully.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **NoPropertyValueGet** when the specified control cannot be obtained.
- Returns **SystemCallFail** when the system call API failed.
- Returns **HidWriteFail** when the buffer cannot be written to the hid handle.
- Returns **HidReadFail** when the buffer cannot be read from the hid handle.
- Returns **TimeoutError** when the wait event timed out.

### Sample Code:

```

void GetIMUConfig_()
{
    IMUCONFIG_TypeDef IMUConfig;
    DeviceHandle handle; //Handle we got from
    OpenDevice()
    if(GetIMUConfig(handle, &IMUConfig)>0 )
    {
        printf("GetIMUConfig success\r\n");
    }
    else

```

```

{
    printf("GetIMUConfig Failed\r\n");
}
}

```

## Result ControlIMUCapture(DeviceHandle devHandle, IMUDATAINPUT\_TypeDef\* IMUInput)

This function sets the IMU Value update in the device that is corresponding to the given devHandle.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
IMUDATAINPUT_TypeDef* IMUInput	[IN] pointer to the <i>IMUDATAINPUT_TypeDef</i> structure.

### Return:

- Returns **Ok** when the IMU Value update is set successfully.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **NoPropertyValueSet** when the specified control cannot set.
- Returns **SystemCallFail** when the system call API failed.
- Returns **HidWriteFail** when the buffer cannot be written to the hid handle.
- Returns **HidReadFail** when the buffer cannot be read from the hid handle.
- Returns **TimeoutError** when the wait event timed out.

### Sample Code:

```

void ControlIMUCapture_()
{
    IMUDATAINPUT_TypeDef IMUInput;
    DeviceHandle handle; //Handle we got from
    OpenDevice()
    //Set the IMUInput to the required Mode
    if(ControlIMUCapture(handle, &IMUInput)>0 )
    {
        printf("ControlIMUCapture success\r\n");
    }
    else
    {
        printf("ControlIMUCapture Failed\r\n");
    }
}

```

## Result GetIMUValue(DeviceHandle devHandle, IMUDATAINPUT\_Typedef\* IMUAxes)

This function gets the IMU axis data from the device that is corresponding to the given devHandle.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
IMUDATAINPUT_Typedef* IMUAxes	[OUT] pointer to the <u>IMUDATAINPUT_Typedef</u> structure.

### Return:

- Returns **Ok** when the IMU axis data is received successfully.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **SystemCallFail** when the system call API failed.
- Returns **HidWriteFail** when the buffer cannot be written to the hid handle.
- Returns **HidReadFail** when the buffer cannot be read from the hid handle.
- Returns **TimeoutError** when the wait event timed out.
- Returns **Failed** when get operation fails.

### Sample Code:

```
void GetIMUValue_()
{
    IMUDATAINPUT_Typedef IMUAxes;
    DeviceHandle handle; //Handle we got from
    OpenDevice()
    if(GetIMUValue(handle, &IMUAxes)>0 )
    {
        printf("GetIMUValue success\r\n");
    }
    else
    {
        printf("GetIMUValue Failed\r\n");
    }
}
```

## Result GetAntiFlickerDetection(DeviceHandle devHandle, uint8\_t\* AFDetect)

This function gets the anti-flicker detection mode from the device that is corresponding to the given devHandle. The flicker detection is used to avoid flicker in the video preview due to AC light source. There are 3 modes of Anti-Flicker

detection. There is off mode for flicker avoidance, auto mode, or manual mode (50 Hz and 60 Hz).

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
uint8_t* AFDetect	[OUT] pointer to uint8_t variable that is filled with the Anti-Flicker detection mode on success of the API. The resultant value is between 0 to 3. <ul style="list-style-type: none"> <li>• 0 represents Auto Mode.</li> <li>• 1 represents Manual 50 Hz.</li> <li>• 2 represents Manual 60 Hz.</li> <li>• 3 represents Disabled.</li> </ul>

#### Return:

- Returns **Ok** when the Anti-Flicker detection was obtained successfully.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **NoPropertyValueGet** when the specified control cannot be obtained.
- Returns **HidWriteFail** when the buffer cannot be written to the hid handle.
- Returns **HidReadFail** when the buffer cannot be read from the hid handle.
- Returns **TimeoutError** when the wait event timed out.

#### Sample Code:

```
void GetAntiFlickerDetection_()
{
    uint8_t AFDetect;
    DeviceHandle handle; //Handle we got from OpenDevice()
    if(GetAntiFlickerDetection(handle, &AFDetect)>0 )
    {
        switch(AFDetect)
        {
            case 0:
                printf("Anti_flicker detection is
in Auto Mode\r\n");
            case 1:
                printf("Anti_flicker detection is
in Manual 50 Hz mode\r\n");
            case 2:
                printf("Anti_flicker detection is
in Manual 60 Hz mode\r\n");
            case 3:
                printf("Anti_flicker detection is
in Disabled mode\r\n");
        }
    }
}
```



```

    }
}
else
{
    printf("GetAntiFlickerDetection failed\r\n");
}
}

```

## Result SetAntiFlickerDetection(DeviceHandle devHandle, uint8\_t AFDetect)

This function sets an anti-flicker detection mode specified by AFDetect to the device that is corresponding to the given devHandle. The flicker detection is used to avoid flicker in the video preview due to AC light source. There are 3 modes of anti-flicker detection. There is off mode for flicker avoidance, auto mode, or manual mode (50 Hz and 60 Hz).

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
uint8_t AFDetect	[IN] uint8_t variable containing the Anti-Flicker mode to be set. The input values are between 0 to 3. <ul style="list-style-type: none"> <li>• 0 represents Auto Mode.</li> <li>• 1 represents Manual 50 Hz.</li> <li>• 2 represents Manual 60 Hz.</li> <li>• 3 represents Disabled.</li> </ul>

### Return:

- Returns **Ok** when the Anti-Flicker detection is set successfully.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **NoPropertyValueSet** when the specified control cannot be set.
- Returns **HidWriteFail** when the buffer cannot be written to the hid handle.
- Returns **HidReadFail** when the buffer cannot be read from the hid handle.
- Returns **TimeoutError** when the wait event timed out.

### Sample Code:

```

void SetAntiFlickerDetection_()
{
    uint8_t AFDetect;
    DeviceHandle handle; //Handle we got from
    OpenDevice()
    if(SetAntiFlickerDetection(handle, AFDetect)>0 )
    {
        printf("SetAntiFlickerDetection success\r\n");
    }
}

```

```

    }
else
{
    printf("SetAntiFlickerDetection failed\r\n");
}
}

```

## Result GetSceneMode(DeviceHandle devHandle, uint8\_t\* sceneMode)

This function gets the scene mode from the device that is corresponding to the given devHandle. Scene mode is used to Flip the Scene. There are 3 scene modes.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
uint8_t* sceneMode	[OUT] pointer to uint8_t variable that is filled with the Scene mode on success of the API. The resultant value is between 0 to 3. <ul style="list-style-type: none"> <li>• 1 represents Horizontal Flip.</li> <li>• 2 represents Vertical Flip.</li> <li>• 3 represents both Horizontal and Vertical Flip.</li> </ul>

### Return:

- Returns **Ok** when the scene mode is obtained successfully.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **NoPropertyValueGet** when the specified control cannot be obtained.
- Returns **HidWriteFail** when the buffer cannot be written to the hid handle.
- Returns **HidReadFail** when the buffer cannot be read from the hid handle.
- Returns **TimeoutError** when the wait event timed out.

### Sample Code:

```

void GetSceneMode_()
{
    uint8_t sceneMode;
    DeviceHandle handle; //Handle we got from
OpenDevice()
    if(GetSceneMode(handle, &sceneMode)>0 )
    {
        switch(sceneMode)
        {
            case 0:

```

```

                                printf("The scene is Flipped
Horizontally\r\n");
                                case 1:
                                    printf("The scene is Flipped
Vertically \r\n");
                                case 2:
                                    printf("The scene is Flipped both
Horizontally and Vertically\r\n");
                                }
                            }
                        else
                        {
                            printf("GetSceneMode failed\r\n");
                        }
                    }
}

```

## Result SetSceneMode(DeviceHandle devHandle, uint8\_t sceneMode)

This function sets the scene mode from the device that is corresponding to the given devHandle. Scene mode is used to Flip the Scene. There are 3 scene modes.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
uint8_t sceneMode	[IN] uint8_t variable containing the Scene mode to be set. The resultant value is between 0 to 3. <ul style="list-style-type: none"> <li>• 1 represents Horizontal Flip.</li> <li>• 2 represents Vertical Flip.</li> <li>• 3 represents both Horizontal and Vertical Flip.</li> </ul>

### Return:

- Returns **Ok** when the scene mode is set successfully.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **NoPropertyValueSet** when the specified control cannot be set.
- Returns **HidWriteFail** when the buffer cannot be written to the hid handle.
- Returns **HidReadFail** when the buffer cannot be read from the hid handle.
- Returns **TimeoutError** when the wait event timed out.

### Sample Code:

```

void SetSceneMode_()
{

```

```

uint8_t sceneMode;

DeviceHandle handle; //Handle we got from
OpenDevice()

if (SetSceneMode(handle, sceneMode)>0 )
{
    printf("SetSceneMode success\r\n");
}
else
{
    printf("SetSceneMode failed\r\n");
}
}

```

## Result GetSpecialEffect(DeviceHandle devHandle, uint8\_t\* splEffect)

This function gets the processed image special effect filters applied to the preview from the device that is corresponding to the given devHandle. The four special effects are normal, black and white, grayscale, negative and sketch.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
uint8_t* splEffect	[OUT] pointer to uint8_t variable that is filled with the special effect mode on success of the API. The resultant values are: <ul style="list-style-type: none"> <li>• 1 for Normal Mode.</li> <li>• 4 for Black and White.</li> <li>• 7 for Grayscale.</li> <li>• 8 for Negative.</li> <li>• 16 for Sketch.</li> </ul>

### Return:

- Returns **Ok** when the special effect was obtained successfully.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **NoPropertyValueGet** when the specified control cannot be obtained.
- Returns **HidWriteFail** when the buffer cannot be written to the hid handle.
- Returns **HidReadFail** when the buffer cannot be read from the hid handle.
- Returns **TimeoutError** when the wait event timed out.

### Sample Code:

```

void GetSpecialEffect_()
{
    uint8_t splEffect;
}

```

```

    DeviceHandle handle; //Handle we got from
    OpenDevice()
    if(GetSpecialEffect(handle, &splEffect)>0 )
    {
        switch(splEffect)
        {
            case 1:
                printf("Preview is in Normal mode.
It doesn't have any added processing. \r\n");
            case 4:
                printf("Black and White special
effect is applied to the preview. The image stream is
composed of black and white pixels \r\n");
            case 7:
                printf("Grayscale special effect is
applied to the preview. The image stream is composed of
gray shades. r\n");
            case 8:
                printf("Negative special effect is
applied to the preview. The normal preview is color
inverted. \r\n");
            case 16:
                printf("Sketch special effect is
applied to the preview. r\n");
        }
    }
    else
    {
        printf("GetSpecialEffect failed\r\n");
    }
}

```

## Result SetSpecialEffect(DeviceHandle devHandle, uint8\_t splEffect)

This function applies the special effect filter specified by splEffect to the preview of the device that is corresponding to the given devHandle. The four special effects are normal, black and white, grayscale, negative and sketch.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
uint8_t splEffect	[IN] uint8_t variable containing the special effect to be set.

- 1 for Normal Mode.
- 4 for Black and White.
- 7 for Grayscale.
- 8 for Negative.
- 16 for Sketch.

#### Return:

- Returns **Ok** when the special effect was set successfully.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **NoPropertyValueSet** when the specified control cannot be set.
- Returns **HidWriteFail** when the buffer cannot be written to the hid handle.
- Returns **HidReadFail** when the buffer cannot be read from the hid handle.
- Returns **TimeoutError** when the wait event timed out.

#### Sample Code:

```
void SetSpecialEffect_()
{
    uint8_t splEffect;
    DeviceHandle handle; //Handle we got from
    OpenDevice()
    if(SetSpecialEffect(handle, splEffect)>0 )
    {
        printf("SetSpecialEffect failed\r\n");
    }
    else
    {
        printf("SetSpecialEffect failed\r\n");
    }
}
```

### Result GetDenoise(DeviceHandle devHandle, uint8\_t\* denoise)

This function gets the De-noise value from the device that is corresponding to the given devHandle.

De-Noise is to blur the effect of noise in the image preview.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
uint8_t* denoise	[OUT] address of a uint8_t variable in which the De-noise value is filled. The range is 0 to 15.

#### Return:

- Returns **Ok** when the De-noise value is obtained successfully.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **NoPropertyValueGet** when the specified control cannot be obtained.
- Returns **HidWriteFail** when the buffer cannot be written to the hid handle.
- Returns **HidReadFail** when the buffer cannot be read from the hid handle.
- Returns **TimeoutError** when the wait event timed out.

#### Sample Code:

```
void GetDenoise_()
{
    uint8_t denoise;
    DeviceHandle handle; //Handle we got from
    OpenDevice()
    if(GetDenoise(handle, &denoise)>0)
    {
        printf("De-noise value is %d\r\n", denoise);
    }
    else
    {
        printf("GetDenoise failed\r\n");
    }
}
```

### Result SetDenoise(DeviceHandle devHandle, uint8\_t denoise)

This function sets the De-noise value specified by the denoise to the device that is corresponding to the given devHandle.

De-Noise is to blur the effect of noise in the image preview.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
uint8_t denoise	[IN] uint8_t variable with De-noise value that is to be set. The range is 0 to 15.

#### Return:

- Returns **Ok** when the De-noise value is set successfully.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **NoPropertyValueSet** when the specified control cannot be set.
- Returns **HidWriteFail** when the buffer cannot be written to the hid handle.
- Returns **HidReadFail** when the buffer cannot be read from the hid handle.
- Returns **TimeoutError** when the wait event timed out.

### Sample Code:

```
void SetDenoise_()
{
    uint8_t denoise;
    DeviceHandle handle; //Handle we got from
    OpenDevice()
    if(SetDenoise(handle, denoise)>0)
    {
        printf("SetDenoise success\r\n");
    }
    else
    {
        printf("SetDenoise failed\r\n");
    }
}
```

### Result GetAutoExposureROI(DeviceHandle devHandle, uint8\_t\* ROI Mode, uint8\_t\* gWinSize)

This function gets an auto exposure algorithm from the device that is corresponding to the given devHandle.

The modes in auto exposure ROI are as follows:

- **Full ROI:** In this mode, full region-based exposure value will be applied to the frame.
- **Manual ROI:** In this mode, you can select the ROI and at that region, the exposure value will be applied to the entire frame.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
uint8_t* ROI Mode	[OUT] pointer to uint8_t variable which is filled with the Auto Exposure ROI Mode on success of this API. The values obtained are: <ul style="list-style-type: none"> <li>• 1 for Face ROI.</li> <li>• 2 for Full ROI.</li> <li>• 3 for Manual ROI.</li> <li>• 4 for Disabled Auto Exposure.</li> </ul>
uint8_t* gWinSize	[OUT] pointer to uint8_t variable, which is filled with the window size, on success of this API. The values can be from 1 to 8.

For frame size 1280 x 720, the exposure region based on the window size is listed in below table.



**Table 1: Window Size vs Exposure Region**

Window Size	Exposure Region (1280 x 720)
1	1/8 (160 x 90)
2	2/8 (320 x 180)
3	3/8 (480 x 270)
4	4/8 (640 x 360)
5	5/8 (800 x 450)
6	6/8 (960 x 540)
7	7/8 (1120 x 630)
8	1 (1280 x 720)

**Return:**

- Returns **Ok** when the Auto-Exposure ROI mode and the window size is obtained successfully.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **NoPropertyValueGet** when the specified control cannot be obtained.
- Returns **HidWriteFail** when the buffer cannot be written to the hid handle.
- Returns **HidReadFail** when the buffer cannot be read from the hid handle.
- Returns **TimeoutError** when the wait event timed out.

**Sample Code:**

```
void GetAutoExposureROI_()
{
    uint8_t ROIMode;
    uint8_t gWinSize;
    DeviceHandle handle; //Handle we got from
    OpenDevice()
    if(GetAutoExposureROI(handle, & ROIMode,
    &gWinSize)>0)
    {
        switch(ROIMode)
        {
            case 0:
                printf("Auto Exposure ROI Mode is
                Face ROI\r\n");
            case 1:
                printf("Auto Exposure ROI Mode is
                Full ROI\r\n");
            case 2:
                printf("Auto Exposure ROI Mode is
                Manual ROI\r\n");
        }
    }
}
```

```

        case 3:
            printf("Auto Exposure ROI Mode is
Disbaled\r\n");
        }
        printf("Window size is %d\r\n", gWinSize);
    }
    else
    {
        printf("GetAutoExposureROI failed\r\n");
    }
}

```

## Result SetAutoExposureROI(DeviceHandle devHandle, uint8\_t ROI Mode, uint32\_t Width, uint32\_t Height, uint32\_t XCor, uint32\_t YCor, uint8\_t winSize)

This function sets an auto exposure algorithm specified by ROI Mode and the window size specified by sWinSize to the device that is corresponding to the given devHandle.

The modes in auto exposure ROI are as follows:

- **Full ROI:** In this mode, full region-based exposure value will be applied to the frame.
- **Manual ROI:** In this mode, you can select the ROI and at that region, the exposure value will be applied to the entire frame.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
uint8_t ROI Mode	[IN] uint8_t variable with an auto exposure ROI Mode that is to be set. The values can be: <ul style="list-style-type: none"> <li>• 1 for Face ROI.</li> <li>• 2 for Full ROI.</li> <li>• 3 for Manual ROI.</li> <li>• 4 for Disabled Auto Exposure.</li> </ul>
uint32_t Width	[IN] This parameter is valid only when the ROI Mode parameter is in Manual ROI. Specifies the width of the ROI.
uint32_t Height	[IN] This parameter is valid only when the ROI Mode parameter is in Manual ROI. Specifies the height of the ROI.
uint32_t XCor	[IN] This parameter is valid only when the ROI Mode parameter is in Manual ROI. Specifies the x co-ordinate of the top left corner in ROI.

<code>uint32_t YCor</code>	[IN] This parameter is valid only when the ROI mode parameter is in Manual ROI. Specifies the y co-ordinate of the top left corner in ROI.
<code>uint8_t* winSize</code>	[OUT] pointer to <code>uint8_t</code> variable, which is filled with the window size, on success of this API. The values can be from 1 to 8.

#### Return:

- Returns **Ok** when the Auto-Exposure ROI mode and the window size along with the width, height, x and y co-ordinates of the top left corner of the ROI is set successfully.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **NoPropertyValueSet** when the specified control cannot be set.
- Returns **HidWriteFail** when the buffer cannot be written to the hid handle.
- Returns **HidReadFail** when the buffer cannot be read from the hid handle.
- Returns **TimeoutError** when the wait event timed out.

#### Sample Code:

```
void SetAutoExposureROI_()
{
    uint8_t ROIMode = 3, winSize = 1;
    uint32_t Width = 160, Height = 90;
    uint32_t XCor = 100, YCor = 200;
    DeviceHandle handle; //Handle we got from
    OpenDevice()
    if(SetAutoExposureROI(handle, ROIMode, Width,
    Height, XCor, YCor, WinSize)>0)
    {
        printf("SetAutoExposureROI success\r\n");
    }
    else
    {
        printf("SetAutoExposureROI failed\r\n");
    }
}
```

## Result GetOrientation(DeviceHandle devHandle, uint8\_t\* orientation)

This function gets the orientation of the scene from the camera. Horizontal flip, vertical flip and both flips simultaneously are supported from the device that is corresponding to the given devHandle.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
uint8_t* orientation	[OUT] pointer to uint8_t variable, which is filled with the type of orientation, on success of this API. The values obtained are: <ul style="list-style-type: none"> <li>• 1 for Horizontal Flip.</li> <li>• 2 for Vertical Flip.</li> <li>• 3 for both Horizontal and Vertical Flip.</li> </ul>

#### Return:

- Returns **Ok** when the Orientation is obtained successfully.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **NoPropertyValueGet** when the specified control cannot be obtained.
- Returns **HidWriteFail** when the buffer cannot be written to the hid handle.
- Returns **HidReadFail** when the buffer cannot be read from the hid handle.
- Returns **TimeoutError** when the wait event timed out.

#### Sample Code:

```
void GetOrientation_()
{
    uint8_t orientation;
    DeviceHandle handle; //Handle we got from
    OpenDevice()
    if(GetOrientation(handle, &orientation)>0)
    {
        switch(Orientation)
        {
            case 1;
                printf("Scene is flipped
Horizontally\r\n");
            case 2;
                printf("Scene is flipped
Vertically\r\n");
            case 3;
                printf("Scene is flipped
Horizontally and Vertically\r\n");
        }
    }
    else
    {
```

```
printf("GetOrientation failed\r\n");
}
}
```

## Result SetOrientation(DeviceHandle devHandle, uint8\_t orientation)

This function sets the orientation of the scene specified by orientation. Horizontal flip, vertical flip and both flips simultaneously are supported to the device that is corresponding to the given devHandle.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
uint8_t orientation	[IN] uint8_t variable with the type of orientation that is to be set. The values can be: <ul style="list-style-type: none"> <li>• 1 for Horizontal Flip.</li> <li>• 2 for Vertical Flip.</li> <li>• 3 for both Horizontal and Vertical Flip.</li> </ul>

### Return:

- Returns **Ok** when the Orientation is set successfully.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **NoPropertyValueSet** when the specified control cannot be set.
- Returns **HidWriteFail** when the buffer cannot be written to the hid handle.
- Returns **HidReadFail** when the buffer cannot be read from the hid handle.
- Returns **TimeoutError** when the wait event timed out.

### Sample Code:

```
void SetOrientation_()
{
    uint8_t orientation = 2;
    DeviceHandle handle; //Handle we got from
    OpenDevice()
    if(SetOrientation(handle, orientation)>0)
    {
        printf("SetOrientation success\r\n");
    }
    else
    {
        printf("SetOrientation failed\r\n");
    }
}
```

## Result GetFaceDetection(DeviceHandle devHandle, uint8\_t\* facedet, uint8\_t\* statusStruct, uint8\_t\* overlayRect)

This function gets the face detection option from the device that is corresponding to the given devHandle.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
uint8_t* facedet	[OUT] pointer to uint8_t variable, which is filled with the face detection option, on success of this API. The values obtained are: <ul style="list-style-type: none"> <li>• 0 for face detection disabled.</li> <li>• 1 for face detection enabled.</li> </ul>
uint8_t* statusStruct	[OUT] pointer to uint8_t variable, which is filled with status structure option, on success of this API. The values obtained are: <ul style="list-style-type: none"> <li>• 0 for status structure disabled.</li> <li>• 1 for status structure enabled.</li> </ul>
uint8_t* overlayRect	[OUT] pointer to uint8_t variable, which is filled with overlay rectangle option, on success of this API. The values obtained are: <ul style="list-style-type: none"> <li>• 0 for overlay rectangle disabled.</li> <li>• 1 for overlay rectangle enabled.</li> </ul>

The Overlay Rectangle option allows you to enable or disable the overlay rectangle around the faces during face detection, and when Embed Data option is enabled, the last part of the frame will be replaced with face details.

### Return:

- Returns **Ok** when the face detection option is obtained successfully.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **NoPropertyValueGet** when the specified control cannot be obtained.
- Returns **HidWriteFail** when the buffer cannot be written to the hid handle.
- Returns **HidReadFail** when the buffer cannot be read from the hid handle.
- Returns **TimeoutError** when the wait event timed out.

### Sample Code:

```
void GetFaceDetection_()
{
    uint8_t facedet, statusStruct, overlayRect;
    DeviceHandle handle; //Handle we got from
    OpenDevice()

    if(GetFaceDetection(handle, &facedet, &statusStruct,
    &overlayRect)>0)
    {
        if(!facedet)
```

```

        printf("Face detection mode is
disabled\r\n");
    else
        printf("Face detection mode is
enabled\r\n");
    if(!StatusStruct)
        printf("Status structure mode is
disabled\r\n");
    else
        printf("Status structure mode is
enabled\r\n");
    if(!OverlayRect)
        printf("Overlay rectangle mode is
disabled\r\n");
    else
        printf("Overlay rectangle mode is
enabled\r\n");
}
else
    printf("GetFaceDetection failed\r\n");
}

```

## Result SetFaceDetection(DeviceHandle devHandle, uint8\_t facedet, uint8\_t statusStruct, uint8\_t overlayRect)

This function sets the face detection option specified by facedet, statusStruct and overlayRect to the device that is corresponding to the given devHandle.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
uint8_t facedet	[IN] uint8_t variable with the face detection option that is to be set. The values can be: <ul style="list-style-type: none"> <li>• 0 for face detection disabled.</li> <li>• 1 for face detection enabled.</li> </ul>
uint8_t statusStruct	[IN] uint8_t variable with the status structure option that is to be set. The values can be: <ul style="list-style-type: none"> <li>• 0 for status structure disabled.</li> <li>• 1 for status structure enabled.</li> </ul>
uint8_t overlayRect	[IN] uint8_t variable with the overlay rectangle option that is to be set. The values can be: <ul style="list-style-type: none"> <li>• 0 for overlay rectangle disabled.</li> <li>• 1 for overlay rectangle enabled.</li> </ul>

The Overlay Rectangle option allows you to enable or disable the overlay rectangle around the faces during face detection, and when Embed Data option is enabled, the last part of the frame will be replaced with face details.

#### Return:

- Returns **Ok** when the face detection option is set successfully.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **NoPropertyValueSet** when the specified control cannot be set.
- Returns **HidWriteFail** when the buffer cannot be written to the hid handle.
- Returns **HidReadFail** when the buffer cannot be read from the hid handle.
- Returns **TimeoutError** when the wait event timed out.

#### Sample Code:

```
void SetFaceDetection_()
{
    uint8_t facedet = 1, statusStruct = 0, overlayRect = 1;

    DeviceHandle handle; //Handle we got from OpenDevice()

    if(SetFaceDetection(handle, facedet, statusStruct, overlayRect)>0)
    {
        printf("SetFaceDetection success\r\n");
    }
    else
    {
        printf("SetFaceDetection failed\r\n");
    }
}
```

### Result GetSmileDetection(DeviceHandle devHandle, uint8\_t\* smiledet, uint8\_t\* statusStruct)

This function gets the smile detection option from the device that is corresponding to the given devHandle.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
uint8_t* smiledet	[OUT] pointer to uint8_t variable which is filled with the smile detection option, on success of this API. The values obtained are: <ul style="list-style-type: none"> <li>• 0 for smile detection disabled.</li> </ul>



- 1 for smile detection enabled.

```
uint8_t*
statusStruct
```

[OUT] pointer to uint8\_t variable which is filled with status structure option, on success of this API. The values obtained are:

- 0 for status structure disabled.
- 1 for status structure enabled.

When status structure option is enabled, the last part of the frame will be replaced with face details.

#### Return:

- Returns **Ok** when the smile detection option is obtained successfully.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **NoPropertyValueGet** when the specified control cannot be obtained.
- Returns **HidWriteFail** when the buffer cannot be written to the hid handle.
- Returns **HidReadFail** when the buffer cannot be read from the hid handle.
- Returns **TimeoutError** when the wait event timed out.

#### Sample Code:

```
void GetSmileDetection_()
{
    uint8_t smiledet, statusStruct;
    DeviceHandle handle; //Handle we got from
    OpenDevice()
    if(GetSmileDetection(handle, &smiledet,
    &statusStruct)>0)
    {
        if(!smiledet)
            printf("Smile detection mode is
disabled\r\n");
        else
            printf("Smile detection mode is
enabled\r\n");
        if(!statusStruct)
            printf("Status structure mode is
disabled\r\n");
        else
            printf("Status structure mode is
enabled\r\n");
    }
    else
    {
        printf("GetSmileDetection failed\r\n");
    }
}
```

```

    }
}

```

## Result SetSmileDetection(DeviceHandle devHandle, uint8\_t smiledet, uint8\_t statusStruct)

This function sets the smile detection option specified by sSmiledet, sStatusStruct to the device that is corresponding to the given devHandle.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
uint8_t smiledet	[IN] uint8_t variable with the smile detection option that is to be set. The values can be: <ul style="list-style-type: none"> <li>• 0 for face detection disabled.</li> <li>• 1 for face detection enabled.</li> </ul>
uint8_t statusStruct	[IN] uint8_t variable with the status structure option that is to be set. The values can be: <ul style="list-style-type: none"> <li>• 0 for status structure disabled.</li> <li>• 1 for status structure enabled.</li> </ul>

When status structure option is enabled, the last part of the frame will be replaced with face details.

### Return:

- Returns **Ok** when the smile detection option is set successfully.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **NoPropertyValueSet** when the specified control cannot be set.
- Returns **HidWriteFail** when the buffer cannot be written to the hid handle.
- Returns **HidReadFail** when the buffer cannot be read from the hid handle.
- Returns **TimeoutError** when the wait event timed out.

### Sample Code:

```

void SetSmileDetection_()
{
    uint8_t smiledet = 1, statusStruct = 0;
    DeviceHandle handle; //Handle we got from
    OpenDevice()
    if(SetSmileDetection(handle, smiledet,
    statusStruct)>0)
    {
        printf("SetSmileDetection success\r\n");
    }
    else
    {

```

```
printf("SetSmileDetection failed\r\n");
}
}
```

## Result GetIMUEmbeddedData(DeviceHandle devHandle, uint8\_t\* IMUData)

Returns the IMU embedded option from the device that is corresponding to the given devHandle.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
uint8_t* IMUData	[OUT] pointer to uint8_t data, that will be filled based on whether the IMU Embedded Data is enabled or disabled. <ul style="list-style-type: none"> <li>• 0 for face detection disabled.</li> <li>• 1 for face detection enabled.</li> </ul>

When IMU embedded data option is enabled, the last part of the frame will be replaced with IMU data.

### Return:

- Returns **Ok** when the smile detection option is obtained successfully.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **NoPropertyValueGet** when the specified control cannot be obtained.
- Returns **HidWriteFail** when the buffer cannot be written to the hid handle.
- Returns **HidReadFail** when the buffer cannot be read from the hid handle.
- Returns **TimeoutError** when the wait event timed out.

### Sample Code:

```
void GetIMUEmbeddedData_()
{
    uint8_t IMUData;
    DeviceHandle handle; //Handle we got from
    OpenDevice()
    if(GetIMUEmbeddedData(handle, &IMUData)>0)
    {
        printf("GetIMUEmbeddedData success\n");
        if(IMUData == 1)
            printf("IMU Embedded data is enabled
\r\n");
        else
            printf("IMU Embedded data is
disabled\r\n");
    }
}
```

```

    }
    else
    {
        printf("GetIMUEmbeddedData failed\r\n");
    }
}

```

## Result SetIMUEmbeddedData(DeviceHandle devHandle, uint8\_t IMUData)

This function sets the IMU embedded option to the device that is corresponding to the given devHandle.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
uint8_t IMUData	[IN] uint8_t data, that is to be set for IMU Embedded Data. <ul style="list-style-type: none"> <li>• 0 for face detection disabled.</li> <li>• 1 for face detection enabled.</li> </ul>

When IMU embedded data option is enabled, the last part of the frame will be replaced with IMU data.

### Return:

- Returns **Ok** when the smile detection option is obtained successfully.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **NoPropertyValueSet** when the specified control cannot be set.
- Returns **HidWriteFail** when the buffer cannot be written to the hid handle.
- Returns **HidReadFail** when the buffer cannot be read from the hid handle.
- Returns **TimeoutError** when the wait event timed out.

### Sample Code:

```

void SetIMUEmbeddedData_(
{
    uint8_t IMUData;
    DeviceHandle handle; //Handle we got from
OpenDevice()
    if(SetIMUEmbeddedData(handle, IMUData)>0)
    {
        printf("SetIMUEmbeddedData success\r\n");
    }
    else
    {

```

```
printf("SetIMUEmbeddedData failed\r\n");
}
}
```

## Result GetExposureCompensation(DeviceHandle devHandle, uint32\_t\* expoComp)

This function gets the exposure compensation from the device that is corresponding to the given devHandle.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
uint32_t* expoComp	[OUT] pointer to uint32_t variable which is filled with the exposure compensation value, on success of this API.

### Return:

- Returns **Ok** when the Exposure compensation value is obtained successfully.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **NoPropertyValueGet** when the specified control cannot be obtained.
- Returns **HidWriteFail** when the buffer cannot be written to the hid handle.
- Returns **HidReadFail** when the buffer cannot be read from the hid handle.
- Returns **TimeoutError** when the wait event timed out.

### Sample Code:

```
void GetExposureCompensation_()
{
    uint32_t expoComp;
    DeviceHandle handle; //Handle we got from
    OpenDevice()
    if(GetExposureCompensation(handle, &expoComp)>0)
    {
        printf("Exposure compensation value is %d\r\n",
expoComp);
    }
    else
    {
        printf("GetExposureCompensation failed\r\n");
    }
}
```

## Result SetExposureCompensation(DeviceHandle devHandle, uint32\_t expoComp)

This function sets the exposure compensation specified by expoComp to the device that is corresponding to the given devHandle.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
uint32_t expoComp	[IN] uint32_t variable with the exposure compensation value that is to be set.

### Return:

- Returns **Ok** when the Exposure compensation value is set successfully.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **NoPropertyValueSet** when the specified control cannot be set.
- Returns **HidWriteFail** when the buffer cannot be written to the hid handle.
- Returns **HidReadFail** when the buffer cannot be read from the hid handle.
- Returns **TimeoutError** when the wait event timed out.

### Sample Code:

```
void SetExposureCompensation_()
{
    uint32_t expoComp;
    DeviceHandle handle; //Handle we got from
    OpenDevice()
    if(SetExposureCompensation(handle, expoComp)>0)
    {
        printf("SetExposureCompensation failed\r\n");
    }
    else
    {
        printf("SetExposureCompensation failed\r\n");
    }
}
```

## Result GetFrameRateCtrl(DeviceHandle devHandle, uint8\_t\* frameRateCtrl)

This function gets the frame rate control value from the device that is corresponding to the given devHandle.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.

```
uint8_t*
frameRateCtrl
```

[OUT] pointer to uint8\_t variable, which is filled with the frame rate control value, on success of this API.

#### Return:

- Returns **Ok** when the Frame Rate Control value is obtained successfully.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **NoPropertyValueGet** when the specified control cannot be obtained.
- Returns **HidWriteFail** when the buffer cannot be written to the hid handle.
- Returns **HidReadFail** when the buffer cannot be read from the hid handle.
- Returns **TimeoutError** when the wait event timed out.

#### Sample Code:

```
void GetFrameRateCtrl_()
{
    uint8_t frameRateCtrl;
    DeviceHandle handle; //Handle we got from
OpenDevice()
    if(GetFrameRateCtrl(handle, &frameRateCtrl)>0)
    {
        printf("Frame Rate control value is %d\r\n",
frameRateCtrl);
    }
    else
    {
        printf("GetFrameRateCtrl failed\r\n");
    }
}
```

### Result SetFrameRateCtrl(DeviceHandle devHandle, uint8\_t frameRateCtrl)

This function sets the frame rate control value that is specified by frameRateCtrl to the device that is corresponding to the given devHandle.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
uint8_t frameRateCtrl	[IN] uint8_t variable with the frame rate control value that is to be set.

#### Return:

- Returns **Ok** when the Frame Rate Control value is set successfully.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **NoPropertyValueSet** when the specified control cannot be set.

- Returns **HidWriteFail** when the buffer cannot be written to the hid handle.
- Returns **HidReadFail** when the buffer cannot be read from the hid handle.
- Returns **TimeoutError** when the wait event timed out.

#### Sample Code:

```
void SetFrameRateCtrl_()
{
    uint8_t frameRateCtrl;
    DeviceHandle handle; //Handle we got from
    OpenDevice()
    if(SetFrameRateCtrl(handle, frameRateCtrl)>0)
        printf("SetFrameRateCtrl success\r\n");
    else
        printf("SetFrameRateCtrl failed\r\n");
}
```

## Result SetDefault(DeviceHandle devHandle)

This function sets all the HID control to default values on the device that is corresponding to the given devHandle.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.

#### Return:

- Returns **Ok** when all the HID controls are set to default successfully.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **NoPropertyValueSet** when the specified control cannot be set.
- Returns **HidWriteFail** when the buffer cannot be written to the hid handle.
- Returns **HidReadFail** when the buffer cannot be read from the hid handle.
- Returns **TimeoutError** when the wait event timed out.

#### Sample Code:

```
void SetDefault_()
{
    DeviceHandle handle; //Handle we got from
    OpenDevice()
    if(SetDefault(handle)>0)
        printf("SetDefault success\r\n");
    else
        printf("SetDefault failed\r\n");
}
```



## Result GetUniqueID(DeviceHandle devHandle, uint64\_t\* uniqueID)

This function gets the unique ID from the device that is corresponding to the given devHandle.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
uint64_t* uniqueID	[OUT] pointer to uint64_t variable, which is filled with the unique ID, on success of this API.

### Return:

- Returns **Ok** when Unique ID is obtained successfully.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **NoPropertyValueGet** when the specified control cannot be obtained.
- Returns **HidWriteFail** when the buffer cannot be written to the hid handle.
- Returns **HidReadFail** when the buffer cannot be read from the hid handle.
- Returns **TimeoutError** when the wait event timed out.

### Sample Code:

```
void GetUniqueID_()
{
    uint64_t uniqueID;
    DeviceHandle handle; //Handle we got from
    OpenDevice()
    if(GetUniqueID(handle, &uniqueID)>0)
        printf("Unique ID of this device is %d\r\n",
uniqueID);
    else
        printf("GetUniqueID failed\r\n");
}
```

## Result SetAvgRegion(DeviceHandle devHandle, AvgRegion region)

This function sets the average depth and IR calculation area specified by region to the device that is corresponding to the given devHandle.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
AvgRegion region	[IN] AvgRegion variable with the region at which the average depth and IR is to be calculated.

- **Center** - Tells that the average depth and IR is calculated at the center of the scene.
- **MouseLivePtr** – Tells that the average depth and IR is calculated along the mouse pointer.

#### Return:

- Returns **Ok** when the region at which the average depth and IR is to be calculated is set successfully.
- Returns **CameraNotOpened** when the device is not opened.

#### Sample Code:

```
void SetAvgRegion_()
{
    AvgRegion region = MouseLivePtr;
    DeviceHandle handle; //Handle we got from
    OpenDevice()
    if(SetAvgRegion(handle, region)>0)
        printf("SetAvgRegion success\r\n");
    else
        printf("SetAvgRegion failed\r\n");
}
```

### Result SetFilterType(DeviceHandle devHandle, int ctrlID, bool selected)

This function sets or resets the type of filter specified by **filterID**, based on **selected** variable to the device that is corresponding to the given devHandle.

Parameters	Description
<code>DeviceHandle devHandle</code>	[IN] Device Handle of the device.
<code>int ctrlID</code>	[IN] int variable with specific filter ID which is to be enabled or disabled. Filter ID is mentioned below. <ul style="list-style-type: none"> <li>• 0 for Spatial Filter.</li> <li>• 1 for Temporal Filter.</li> <li>• 2 for Edge detection.</li> </ul>
<code>bool selected</code>	[IN] <b>true</b> when the filter mentioned by ctrlID is to be enabled. <b>False</b> when the filter mentioned by ctrlID is to be disabled.

#### Return:

- Returns **Ok** when the filter type mentioned in **ctrlID** is enabled or disabled based on **selected** is success.
- Returns **CameraNotOpened** when the device is not opened.

#### Sample Code:

```
void SetAvgRegion_()
{
    AvgRegion region = MouseLivePtr;
```

```

    DeviceHandle handle; //Handle we got from
OpenDevice()
    if(SetAvgRegion(handle, region)>0)
    {
        printf("SetAvgRegion success\r\n");
    }
    else
    {
        printf("SetAvgRegion failed\r\n");
    }
}

```

## Result SetDepthPos(DeviceHandle devHandle, DepthPtr pos)

This function sets the average depth and IR calculation area specified by pos to the device that is corresponding to the given devHandle.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
DepthPtr pos	[IN] Object of the structure <i>DepthPtr</i> , containing the x and y co-ordinates.

### Return:

- Returns **Ok** when the average Depth and IR calculation area specified by **pos** is success.
- Returns **CameraNotOpened** when the device is not opened.

### Sample Code:

```

void SetDepthPos_()
{
    DepthPtr pos;
    pos.X = pos.Y = 100;
    DeviceHandle handle; //Handle we got from
OpenDevice()
    if(SetDepthPos(handle, pos)>0)
    {
        printf("SetDepthPos success\r\n");
    }
    else
    {
        printf("SetDepthPos failed\r\n");
    }
}

```

```

    }
}

```

## Result SetCursorColor(DeviceHandle devHandle, int color)

This function sets the cursor color to the white or black specified by color parameter to the device that is corresponding to the given devHandle.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
int color	[IN] Non-Zero value for black cursor color. Zero for white cursor color.

### Return:

- Returns **Ok** when the cursor color specified by color parameter is set successfully.
- Returns **CameraNotOpened** when the device is not opened.

### Sample Code:

```

void SetCursorColor_()
{
    DeviceHandle handle; //Handle we got from
    OpenDevice()
    if(SetCursorColor(handle, 0)>0)
        printf("Cursor color set to white\r\n");
    else
        printf("SetCursorColor failed\r\n");
}

```

## Result SetUnDistortion(DeviceHandle devHandle, int undistort)

This function enables or disables undistortion based on the undistort parameter to the device that is corresponding to the given devHandle.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
int undistort	[IN] Non-Zero value to enable undistortion. Zero to disable undistortion.

### Return:

- Returns **Ok** when undistortion is disabled or enabled based on undistort parameter is success.
- Returns **CameraNotOpened** when the device is not opened.

- Returns **RGB\_DCalibNotFound** when RGB\_D calibration values are not found.
- Returns **InvalidDataMode** when current data mode does not support the feature.

#### Sample Code:

```
void SetUnDistortion()
{
    DeviceHandle handle; //Handle we got from
    OpenDevice()
    if(SetUnDistortion(handle, 0)>0)
    {
        printf("Undistortion is disbaled\r\n");
    }
    else
    {
        printf("SetUnDistortion failed\r\n");
    }
}
```

## Result SetRGBDMapping (DeviceHandle devHandle, int rgbdMapping)

This function enables or disables RGB-D Mapping based on the calibration parameter to the device that is corresponding to the given devHandle.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
int rgbdmapping	[IN] Non-Zero value to enable RGB-D Mapping. Zero to disable RGB-D Mapping.

#### Return:

- Returns **Ok** when RGB-D Mapping is disabled or enabled based on calibration parameter is success.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **RGB\_DCalibNotFound** when RGB\_D calibration values are not found.
- Returns **InvalidDataMode** when current data mode does not support the feature.

#### Sample Code:

```
void SetRGBDMapping()
{
    DeviceHandle handle; //Handle we got from
    OpenDevice()
```

```

if(SetRGBDMapping(handle, 0)>0)
{
    printf("RGB-D Mapping is disbaled\r\n");
}
else
{
    printf("SetRGBDMapping failed\r\n");
}
}

```

## Result SetPlanarization(DeviceHandle devHandle, int planarize)

This function enables or disables planarization based on the planarize parameter to the device that is corresponding to the given devHandle.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
int planarize	[IN] Non-Zero value to enable planarization. Zero to disable planarization.

TOF camera gets the distance between a target object and the camera as depth. The depth data from the targets in the center of the frame is correct, but depth data for the targets away from the center of the frame is the diagonal distance from the target to the camera, but we should know the distance based on the world co-ordinates. Hence depth planarization is used.

### Return:

- Returns **Ok** when planarization is disabled or enabled based on **planarize** parameter is success.
- Returns **CameraNotOpened** when the device is not opened.

### Sample Code:

```

void SetPlanarization_()
{
    DeviceHandle handle; //Handle we got from
    OpenDevice()
    if(SetPlanarization(handle, 0)>0)
    {
        printf("Planarization is disbaled\r\n");
    }
}

```

```

else
{
    printf("SetPlanarization failed\r\n");
}
}

```

## Result Result SetFlyingPixelFilter(DeviceHandle devHandle, int state)

This function enables or disables flying pixel filter based on the state parameter to the device that is corresponding to the given devHandle.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
int state	[IN] Non-Zero value to enable flying pixel filter. Zero to disable flying pixel filter.

Flying pixels are false pixels which are non-existent but exist in 3D point cloud. The lateral resolution of TOF sensors is generally less. Hence one pixel covers a large area in 3D space. If this area has objects at different depths, they combine into a single pixel causing flying pixels. This effect is more prominent at the boundaries of the objects. This is because at the edges, light from foreground and background may combine to produce intermediate erroneous depth values, which can be seen as a wave of pixels known as flying pixels. If a pixel has large depth discontinuity from its neighboring pixels, it is considered as flying pixel. This pixel will be termed as invalid and ignored in 3D point cloud.

### Return:

- Returns **Ok** when flying pixel filter is disabled or enabled based on **state** parameter is success.
- Returns **CameraNotOpened** when the device is not opened.

### Sample Code:

```

void SetFlyingPixelFilter_()
{
    DeviceHandle handle; //Handle we got from
OpenDevice()
    if(SetFlyingPixelFilter(handle, 0)>0)
    {
        printf("SetFlyingPixelFilter is
disbaled\r\n");
    }
    else

```

```

    {
        printf("SetFlyingPixelFilter failed\r\n");
    }
}

```

## Result GetDepthIRValues(DeviceHandle devHandle, int\* avgDepth, int\* stdDepth, int\* avgIR, int\* stdIR)

This function gets the average depth and IR and sigma depth and IR from the device that is corresponding to the given devHandle.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
int* avgDepth	[OUT] pointer to int variable that is to be filled with the average depth value.
int* stdDepth	[OUT] pointer to int variable that is to be filled with the sigma depth value.
int* avgIR	[OUT] pointer to int variable that is to be filled with the average IR value.
int* stdIR	[OUT] pointer to int variable that is to be filled with the sigma IR value.

### Return:

- Returns **Ok** when average depth and IR, and sigma depth and IR is obtained successfully.
- Returns **CameraNotOpened** when the device is not opened.

### Sample Code:

```

void GetDepthIRValues_()
{
    int avgDepth, stdDepth, avgIR, stdIR;
    DeviceHandle handle; //Handle we got from
    OpenDevice()
    if(GetDepthIRValues(handle, &avgDepth, &stdDepth,
    &avgIR, &stdIR)>0)
    {
        printf("Average depth value is %d\r\n",
        avgDepth);
        printf("sigma depth value is %d\r\n",
        stdDepth);
        printf("Average IR value is %d\r\n", avgIR);
        printf("sigma IR value is %d\r\n", stdIR);
    }
}

```



```

    }
    else
    {
        printf("GetDepthIRValues failed\r\n");
    }
}

```

## Void RegisterFrameCallback(DeviceHandle devHandle, function<void(int)> cb)

Register a callback function specified by cb when frame is received from the device that is corresponding to the given devHandle.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
function<void(int)> cb	[IN] Function pointer of the callback function.

### Sample Code

```

void callback_func(int a)
{
    printf("Frames received from device\r\n");
}

void RegisterFrameCallback_()
{
    DeviceHandle handle; //Handle we got from
    OpenDevice()
    RegisterFrameCallback(handle, &callback_func);
}

```

## Void RegisterNotificationCallback(DeviceHandle devHandle, function<void(int)> cb)

This function registers a callback function specified by cb when notification is received from the device that is corresponding to the given devHandle. This is used to receive device removal notification.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
function<void(int)> cb	[IN] Function pointer of the callback function.

### Sample Code :

```
void callback_func(int a)
{
    printf("Device removed notification received from
device\r\n");
}
void RegisterNotificationCallback_()
{
    DeviceHandle handle; //Handle we got from
OpenDevice()
    RegisterNotificationCallback(handle,
&callback_func);
}
```

### Result UpdateAvgXandY(DeviceHandle devHandle, int avg\_x, int avg\_y)

This function updates the ROI for which the average depth and IR is being calculated of the device that is corresponding to the given devHandle.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
int avg_x	[IN] width of the average depth calculating ROI
int avg_y	[IN] height of the average depth calculating ROI

#### Return:

- Returns **Ok** when the ROI for average depth and IR is set successfully.
- Returns **CameraNotOpened** when the device is not opened.

### Sample Code:

```
void UpdateAvgXandY_()
{
    int avg_x = 64, avg_y = 64;
    DeviceHandle handle; //Handle we got from
OpenDevice()
    if(UpdateAvgXandY(handle, avg_x, avg_y)>0)
    {
        printf("UpdateAvgXandY success\r\n");
    }
    else
    {
        printf("UpdateAvgXandY failed\r\n");
    }
}
```

```
    }  
}
```

## Result UpdateColorMap(DeviceHandle devHandle, int min, int max, int colormap)

This function updates the colormap that is being applied on the raw depth data to the device that is corresponding to the given devHandle, along with the minimum and maximum limit of depth data to be covered in the depth colormap scene.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
int min	[IN] minimum limit of depth data that is to be included in the colormap image.
int max	[IN] maximum limit of depth data that is to be included in the colormap image.
int colormap	[IN] specific number for a colormap. There are 12 colormap and the values are from 0 to 11.

### Return:

- Returns **Ok** when the minimum and maximum limit of depth, and the colormap is updated successfully.
- Returns **CameraNotOpened** when the device is not opened.

### Sample Code:

```
void UpdateColorMap_()
{
    int min = 200, max = 1200, colormap = 4;
    DeviceHandle handle; //Handle we got from
    OpenDevice()
    if(UpdateColorMap(handle, min, max, colormap)>0)
    {
        printf("UpdateColorMap success\r\n");
    }
    else
    {
        printf("UpdateColorMap failed\r\n");
    }
}
```

## Result SetAvgIRDisplay(DeviceHandle devHandle, uint8\_t avgIRDisplay)

This function sets whether to display the IR data along with the average Depth Value.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
uint8_t avgIRDisplay	[IN] uint8_t variable to set or unset the IR display along with Average depth. <ul style="list-style-type: none"> <li>• 0 to unset</li> <li>• 1 to set</li> </ul>

### Return:

- Returns **Ok** when the display IR feature is set successfully.
- Returns **CameraNotOpened** when the device is not opened.

### Sample Code:

```
void SetAvgIRDisplay_()
{
    int avgIRDisplay = 0;
    DeviceHandle handle; //Handle we got from
    OpenDevice()
    if(SetAvgIRDisplay(handle, avgIRDisplay)>0)
    {
        printf("SetAvgIRDisplay success\r\n");
    }
    else
    {
        printf("SetAvgIRDisplay failed\r\n");
    }
}
```

## Void pError(std::string message = "")

This function produces an error message on standard error describing the last error encountered during a call to the TOF library function.

Parameters	Description
std::string message	[IN] Error message string to be printed during failure condition.

### Sample Code:

```
void pError_()
{
```

```

    pError(<content>)
}

```

## Result ReadFirmwareVersion(DeviceHandle devHandle, uint8\_t\* majorVersion, uint8\_t\* minorVersion1, uint16\_t\* minorVersion2, uint16\_t\* minorVersion3)

This function gets the firmware version from the device that is corresponding to the given devHandle.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
uint8_t* majorVersion	[OUT] pointer to uint8_t variable to store the Major version of firmware.
uint8_t* minorVersion1	[OUT] pointer to uint8_t variable to store the Minor version 1 of firmware.
uint16_t* minorVersion2	[OUT] pointer to uint16_t variable to store the Minor version 2 of firmware.
uint16_t* minorVersion3	[OUT] pointer to uint16_t variable to store the Minor version 3 of firmware.

### Return:

- Returns **Ok** when the firmware version is read successfully.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **NoPropertyValueGet** when the specified control cannot be obtained.
- Returns **HidWriteFail** when the buffer cannot be written to the hid handle.
- Returns **HidReadFail** when the buffer cannot be read from the hid handle.
- Returns **TimeoutError** when the wait event timed out.

### Sample Code:

```

void readFirmwareVersion_()
{
    uint8_t majorVersion = 0, minorVersion1 = 0;
    uint16_t minorVersion2 = 0, minorVersion3 = 0;
    DeviceHandle handle; //Handle we got from
    OpenDevice()
    if(ReadFirmwareVersion(handle, &majorVersion,
    &minorVersion1, &minorVersion2, &minorVersion3)>0)
    {
        printf("Firmware version is %d.%d.%d.%d\r\n",
        majorVersion, minorVersion1, minorVersion2,
        minorVersion3);
    }
    else

```

```

    {
        printf("readFirmwareVersion failed\r\n");
    }
}

```

## Result CalibReadReqDepthIntrinsic(DeviceHandle devHandle, int\* depthIntFileLength)

This function requests the device to read intrinsic calibration data of the depth camera that is corresponding to the given devHandle.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
int* depthIntFileLength	[OUT] Pointer to int data that contains the length of the valid calibration Data on success.

### Return:

- Returns **Ok** when the request to read intrinsic calibration data of the depth camera is success.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **NoPropertyValueGet** when the specified control cannot be obtained.
- Returns **HidWriteFail** when the buffer cannot be written to the hid handle.
- Returns **HidReadFail** when the buffer cannot be read from the hid handle.
- Returns **TimeoutError** when the wait event timed out.

### Sample Code:

```

void calibReadReqDepthIntrinsic_()
{
    int depthIntFileLength;
    DeviceHandle handle; //Handle we got from
    OpenDevice()

    if (CalibReadReqDepthIntrinsic(handle,
    &depthIntFileLength)>0)
    {
        printf("CalibReadReqDepthIntrinsic
    success\r\n");
    }
    else
    {
        printf("CalibReadReqDepthIntrinsic
    failed\r\n");
    }
}

```

```
}
```

## Result CalibReadDepthIntrinsic(DeviceHandle devHandle, int\* depthIntFileLength, unsigned char\* DepthIntrinsicBuffer)

This function reads the intrinsic calibration data of the depth camera that is corresponding to the given devHandle.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
int* depthIntFileLength	[OUT] Pointer to int data that contains the length of the valid calibration Data that is read.
unsigned char* DepthIntrinsicBuffer	[OUT] Pointer to an unsigned char that will contain the intrinsic calibration Data of depth camera on success of the API.

### Return:

- Returns **Ok** when the intrinsic calibration data of the depth camera is read successfully.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **NoPropertyValueGet** when the specified control cannot be obtained.
- Returns **HidWriteFail** when the buffer cannot be written to the hid handle.
- Returns **HidReadFail** when the buffer cannot be read from the hid handle.
- Returns **TimeoutError** when the wait event timed out.

### Sample Code:

```
void calibReadDepthIntrinsic_()
{
    DeviceHandle handle; //Handle we got from
    OpenDevice()
    int depthIntFileLength;
    unsigned char* DepthIntrinsicBuffer;
    int read_length_depth; //read_length_data must be
    the length requested using calibReadReqDepthIntrinsic API
    DepthIntrinsicBuffer = (unsigned
    char*)malloc(read_length_depth);
    if(CalibReadDepthIntrinsic(handle,
    &depthIntFileLength, DepthIntrinsicBuffer)>0)
    {
        printf("CalibReadDepthIntrinsic success\r\n");
    }
    else
```

```

    {
        printf("CalibReadDepthIntrinsic failed\r\n");
    }
    free(DepthIntrinsicBuffer);
}

```

## Result CalibReadReqRGBIntrinsic (DeviceHandle devHandle, int\* RGBIntFileLength)

This function requests the device to read intrinsic calibration data of the RGB camera that is corresponding to the given devHandle.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
int* RGBIntFileLength	[OUT] Pointer to int data that contains the length of the valid calibration Data on success.

### Return:

- Returns **Ok** when the request to read intrinsic calibration data of the RGB camera is success.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **NoPropertyValueGet** when the specified control cannot be obtained.
- Returns **HidWriteFail** when the buffer cannot be written to the hid handle.
- Returns **HidReadFail** when the buffer cannot be read from the hid handle.
- Returns **TimeoutError** when the wait event timed out.

### Sample Code:

```

void CalibReadReqRGBIntrinsic_()
{
    DeviceHandle handle; //Handle we got from
    OpenDevice()
    int RGBIntFileLength;
    if(CalibReadReqRGBIntrinsic(handle,
    &RGBIntFileLength)>0)
    {
        printf("CalibReadReqRGBIntrinsic
    success\r\n");
    }
    else
    {
        printf("CalibReadReqRGBIntrinsic failed\r\n");
    }
}

```



```
}
```

## Result CalibReadRGBIntrinsic(DeviceHandle devHandle, int\* RGBIntFileLength, unsigned char\* RGBIntrinsicBuffer)

This function reads the intrinsic calibration data of the RGB camera that is corresponding to the given devHandle.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
int* RGBIntFileLength	[OUT] Pointer to int data that contains the length of the valid calibration Data that is read.
unsigned char* RGBIntrinsicBuffer	[OUT] Pointer to an unsigned char that will contain the intrinsic calibration Data of RGB camera on success of the API.

### Return:

- Returns **Ok** when the intrinsic calibration data of the RGB camera is read successfully.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **NoPropertyValueGet** when the specified control cannot be obtained.
- Returns **HidWriteFail** when the buffer cannot be written to the hid handle.
- Returns **HidReadFail** when the buffer cannot be read from the hid handle.
- Returns **TimeoutError** when the wait event timed out.

### Sample Code:

```
void CalibReadRGBIntrinsic_()
{
    DeviceHandle handle; //Handle we got from
    OpenDevice()
    int RGBIntFileLength;
    unsigned char* RGBIntrinsicBuffer;
    int read_length_depth; //read_length_data must be
    the length requested using CalibReadReqRGBIntrinsic API
    RGBIntrinsicBuffer = (unsigned
    char*)malloc(read_length_depth);
    if(CalibReadRGBIntrinsic(handle, &RGBIntFileLength,
    RGBIntrinsicBuffer)>0)
    {
        printf("CalibReadRGBIntrinsic success\r\n");
    }
    else
```

```

{
    printf("CalibReadRGBIntrinsic failed\r\n");
}
free(RGBIntrinsicBuffer);
}

```

## Result CalibReadReqExtrinsic (DeviceHandle devHandle, int\* extFileLength)

This function requests the device to read extrinsic calibration data of the device that is corresponding to the given devHandle.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
int* extFileLength	[OUT] Pointer to int data that contains the length of the valid calibration Data on success.

### Return:

- Returns **Ok** when the request to extrinsic calibration data of the device is success.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **NoPropertyValueGet** when the specified control cannot be obtained.
- Returns **HidWriteFail** when the buffer cannot be written to the hid handle.
- Returns **HidReadFail** when the buffer cannot be read from the hid handle.
- Returns **TimeoutError** when the wait event timed out.

### Sample Code:

```

void CalibReadReqExtrinsic_()
{
    DeviceHandle handle; //Handle we got from
    OpenDevice()
    int extFileLength;
    if(CalibReadReqExtrinsic(handle, &extFileLength)>0)
    {
        printf("CalibReadReqExtrinsic success\r\n");
    }
    else
    {
        printf("CalibReadReqExtrinsic failed\r\n");
    }
}

```

## Result CalibReadExtrinsic(DeviceHandle devHandle, int\* extFileLength, unsigned char\* ExtrinsicBuffer)

This function reads the extrinsic calibration data of the device that is corresponding to the given devHandle.

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
int* extFileLength	[OUT] Pointer to int data that contains the length of the valid calibration Data that is read.
unsigned char* ExtrinsicBuffer	[OUT] Pointer to an unsigned char that will contain the extrinsic calibration Data of the device on success of the API.

### Return:

- Returns **Ok** when the extrinsic calibration Data of the device is read successfully.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **NoPropertyValueGet** when the specified control cannot be obtained.
- Returns **HidWriteFail** when the buffer cannot be written to the hid handle.
- Returns **HidReadFail** when the buffer cannot be read from the hid handle.
- Returns **TimeoutError** when the wait event timed out.

### Sample Code:

```
void CalibReadExtrinsic_()
{
    DeviceHandle handle; //Handle we got from
    OpenDevice()
    int extFileLength;
    unsigned char* ExtrinsicBuffer;
    int read_length_depth; //read_length_data must be
    the length requested using CalibReadReqExtrinsic API
    ExtrinsicBuffer = (unsigned
    char*)malloc(read_length_depth);
    if(CalibReadExtrinsic(handle, &extFileLength,
    ExtrinsicBuffer)>0)
    {
        printf("CalibReadExtrinsic success\r\n");
    }
    else
    {
        printf("CalibReadExtrinsic failed\r\n");
    }
    free(ExtrinsicBuffer);
}
```

```
}
```

## Result GetSDKVersion (uint8\_t\* majorVersion, uint8\_t\* minorVersion1, uint16\_t\* minorVersion2)

This function gets the SDK version.

Parameters	Description
uint8_t* majorVersion	[OUT] pointer to uint8_t variable to store the Major version of SDK.
uint8_t* minorVersion1	[OUT] pointer to uint8_t variable to store the Minor version 1 of SDK.
uint16_t* minorVersion2	[OUT] pointer to uint16_t variable to store the Minor version 2 of SDK.

### Return:

- Returns **Ok** when the firmware version is read successfully.

### Sample Code:

```
void GetSDKVersion_()
{
    uint8_t majorVersion = 0, minorVersion1 = 0;
    uint16_t minorVersion2 = 0;
    if (GetSDKVersion(&majorVersion, &minorVersion1,
&minorVersion2) > 0)
    {
        printf("SDK version is %d.%d.%d\r\n",
majorVersion, minorVersion1, minorVersion2);
    }
    else
    {
        printf("Get SDK version failed\r\n");
    }
}
```

## Result SetLogLevel (LogLevel logState)

Sets the logging level to SDK.

Parameters	Description
LogLevel logState	[IN] State of Log, any value from LogLevel enum

### Return:

- Returns **Ok** when logging level is set successfully.
- Returns **Failed** when setting log state failed.

#### Sample Code:

```
void SetLogLevel_()
{
    LogLevel logState;
    if (SetLogLevel (LogLevel)>0)
    {
        printf("SetLogLevel success\r\n");
    }
    else
    {
        printf("SetLogLevel failed\r\n");
    }
}
```

## Result SetLogging(Logging log)

Sets the logging to SDK.

Parameters	Description
Logging log	[IN] Place of Log, any value from Logging enum

#### Return:

- Returns **Ok** when logging is set successfully.
- Returns **Failed** when setting log failed.

#### Sample Code:

```
void SetLogging_()
{
    Logging log;
    if (SetLogging(log)>0)
    {
        printf("SetLogging success\r\n");
    }
    else
    {

```

```
        printf("SetLogging failed\r\n");
    }
}
```

## Result GetDeviceCalibrationParams(DeviceHandle devHandle, CalibrationParams\* calibparams)

This function gets the Calibration parameters of the device that is corresponding to the given devHandle and fills the values in calibparams

Parameters	Description
DeviceHandle devHandle	[IN] Device Handle of the device.
CalibrationParams* calibparams	[OUT] pointer to CalibrationParams structure to store the intrinsic and extrinsic calibration parameters.

### Return:

- Returns **Ok** when CalibrationParams Data read is successfully.
- Returns **RGB\_DCalibNotFound** when RGB\_D calibration values are not found.

### Sample Code:

```
void GetDeviceCalibrationParams_()
{
    DeviceHandle handle; //Handle we got from
    OpenDevice()
    CalibrationParams calibparams;
    if(GetDeviceCalibrationParams(handle,
    &calibparams)>0)
    {
        printf("Get Calibration parameters
        success\r\n");
    }
    else
    {
        printf("Get Calibration parameters
        failed\r\n");
    }
}
```

# Enum

---

The details regarding the enum used will be discussed below.

## DataMode

Different datamode supported by DepthVista is listed in the DataMode enum.

```
typedef enum {  
    Depth_IR_Mode = 0,  
    Depth_Mode = 2,  
    IR_Mode = 3,  
    Depth_IR_RGB_VGA_Mode = 4,  
    Depth_IR_RGB_HD_Mode = 5,  
    RGB_VGA_Mode = 6,  
    RGB_HD_Mode = 7,  
    RGB_Full_HD_Mode = 8,  
    RGB_1200p_Mode = 9,  
}DataMode;
```

Where,

**Depth\_IR\_Mode:** Output depth frame (640 x 480) and IR frame (640 x 480) at 30 FPS.

**Depth\_Mode:** Output depth frame (640 x 480) at 30 FPS.

**IR\_Mode:** Output IR frame (640 x 480) at 30 FPS.

**Depth\_IR\_RGB\_VGA\_Mode:** Output depth frame (640 x 480), IR frame (640 x 480) and RGB frame (640 x 480) at 30 FPS.

**Depth\_IR\_RGB\_HD\_Mode:** Output depth frame (640 x 480), IR frame (640 x 480) and RGB frame (1280 x 720) at 30 FPS.

**RGB\_VGA\_Mode:** Output RGB frame (640 x 480) at 60 FPS.

**RGB\_HD\_Mode:** Output RGB frame (1280 x 720) at 60 FPS.

**RGB\_Full\_HD\_Mode:** Output RGB frame (1920 x 1080) at 30 FPS.

**RGB\_1200p\_Mode:** Output RGB frame (1920 x 1200) at 30 FPS.

## LogLevel

LogLevel enum contains the level of the log that is to be stored in log file or to print in console.

```
typedef enum{  
    Low = 0,  
    Critical = 1,  
    High = 2,  
    Disable = 3  
}LogLevel;
```

Where,

**Low:** Enable All logs.

**Critical:** Enable very critical logs.

**High:** Enables high priority logs.

**Disable:** Enable all logs.

## Logging

Logging enum contains the options for logging debug prints.

```
typedef enum{  
    Console = 1,  
    LogFile = 2,  
    BothConsoleAndLogFile = 3,  
}Logging;
```

Where,

**Console:** Prints the log only in Console.

**LogFile:** Prints the log only in Log File.

**BothConsoleAndLogFile:** Prints the log in both Console and Log File.

## CameraMode

Different camera mode supported by DepthVista is listed in the CameraMode enum

```
typedef enum {  
    DualCameraMode = 0,
```



```
    RGBCameraMode = 1,  
    TOFCameraMode = 2,  
} CameraMode;
```

Where,

**DualCameraMode:** Both RGB and TOF camera enabled mode.

**RGBCameraMode:** Only RGB camera enabled mode.

**TOFCameraMode:** Only TOF camera enabled mode.

## DepthRange

Different depth range supported by DepthVista is listed in the DepthRange enum

```
typedef enum {  
    NearRange = 0,  
    FarRange = 1,  
} DepthRange;
```

Where,

**NearRange:** Near range mode.

**FarRange:** Far range mode.

## FrameType

Types of frames that can be obtained from DepthVista is listed in the FrameType enum.

```
typedef enum {  
    IRPreviewFrame,  
    DepthColorMap,  
    RGBFrame,  
    DepthRawFrame,  
} FrameType;
```

Where,

**IRPreviewFrame:** Separate IR frame.

**DepthColorMap:** Depth data that is applied with Color map for preview purpose.

**RGBFrame:** Separate RGB frame.

**DepthRawFrame:** Raw depth frame without applying color map. Hence this frame cannot be used for preview.

## Result

Result enum contain the values that will be returned from supported APIs.

```
typedef enum {  
    Ok = 1,  
    NotInitialized = -1,  
    NotDeInitialized = -2,  
    InvalidFrameType = -3,  
    NoStreaming = -4,  
    AlreadyStreaming = -5,  
    InvalidNode = -6,  
    CameraNotOpened = -7,  
    InvalidDeviceIndex = -8,  
    NoDeviceConnected = -9,  
    NoPropertyValueGet = -10,  
    NoPropertyValueSet = -11,  
    SystemCallFail = -12,  
    InvalidValue = -13,  
    HidWriteFail = -14,  
    HidReadFail = -15,  
    UVCOpenFail = -16,  
    TimeoutError = -17,  
    InvalidBuffer = -18,  
    RGB_DCalibNotFound = -19,  
    FrameSeparationFailed = -20,  
    RegisterCallBackFailed = -21,  
    CameraAlreadyOpen = -22,  
    InvalidDataMode = -23,  
    Failed = -24,  
    Others = -255,  
}Result;
```

Where,

**Ok:** The API completed successfully.

**NotInitialized:** The APIs are not initialized.

**NotDeInitialized:** The APIs are not deinitialized.

**InvalidFrameType:** The input frame type is invalid.

**NoStreaming:** The camera is not streaming.

**AlreadyStreaming:** The camera is already streaming.

**InvalidNode:** The device Node is invalid.

**CameraNotOpened:** The camera has not been opened.

**InvalidDeviceIndex:** The input device index is invalid.

**NoDeviceConnected:** There is no depth camera connected or the camera has not been connected correctly.

**NoPropertyValueGet:** Cannot get the value for the specified property.

**NoPropertyValueSet:** Cannot set the value for the specified property.

**SystemCallFail:** System call API failed.

**InvalidValue:** One or more of the parameter values provided are invalid.

**HidWriteFail:** Cannot write the buffer to the hid handle.

**HidReadFail:** Cannot read the buffer from the hid handle.

**UVCOpenFail:** Open UVC Failed.

**TimeoutError:** Read on file descriptor timed out.

**InvalidBuffer:** Frame from the camera is invalid.

**RGB\_DCalibNotFound:** RGB-D Calibration Data not found in Camera.

**FrameSeparationFailed:** Frame separation Failed.

**RegisterCallBackFailed:** Registering frame call back Failed.

**CameraAlreadyOpen:** Camera is already open.

**InvalidDataMode:** Invalid Data Mode for intended change.

**Failed:** Operation Failed.

**Others:** An unknown error occurred.

## AvgRegion

Contains two values that tells whether the Average of Depth and IR is calculated at the center of the scene or along the mouse pointer.

```
typedef enum {  
    Center = 0,
```

```
MouseLivePtr = 1,  
}AvgRegion;
```

Where,

**Center:** Tells that the average depth and IR is calculated at the center of the scene.

**MouseLivePtr:** Tells that the average depth and IR is calculated along the mouse pointer.

## UVCPropID

UVC controls for DepthVista is listed in UVCPropID enum.

```
typedef enum {  
    TOF_UVC_CID_BRIGHTNESS,  
    TOF_UVC_CID_CONTRAST,  
    TOF_UVC_CID_SATURATION,  
    TOF_UVC_CID_WB_AUTO,  
    TOF_UVC_CID_GAMMA,  
    TOF_UVC_CID_GAIN,  
    TOF_UVC_CID_PWR_LINE_FREQ,  
    TOF_UVC_CID_WB_TEMP,  
    TOF_UVC_CID_SHARPNESS,  
    TOF_UVC_CID_EXPOSURE_AUTO,  
    TOF_UVC_CID_EXPSOURE_ABS,  
}UVCPropID;
```

Where,

**TOF\_UVC\_CID\_BRIGHTNESS** : Brightness Control ID.

**TOF\_UVC\_CID\_CONTRAST** : Contrast Control ID.

**TOF\_UVC\_CID\_SATURATION** : Saturation Control ID.

**TOF\_UVC\_CID\_WB\_AUTO** : Auto White balance Control ID.

**TOF\_UVC\_CID\_GAMMA** : Gamma Control ID.

**TOF\_UVC\_CID\_GAIN** : Gain Control ID.

**TOF\_UVC\_CID\_PWR\_LINE\_FREQ** : Powerline Frequency Control ID.

**TOF\_UVC\_CID\_WB\_TEMP** : White balance Control ID.

**TOF\_UVC\_CID\_SHARPNESS** : Sharpness Control ID.

**TOF\_UVC\_CID\_EXPOSURE\_AUTO** : Auto Exposure Control ID.

**TOF\_UVC\_CID\_EXPSOURE\_ABS** : Absolute Exposure Control ID.

## CalibValidFlag

CalibValidFlag enum Contain flag values to specify about the board's Calibration status.

```
typedef enum {  
    Not_Calibrated = -1,  
    Only_VGA = 0,  
    Only_HD = 1,  
    Both_VGA_HD = 2  
}CalibValidFlag;
```

Where,

**Not\_Calibrated:** Camera module is not calibrated.

**Only\_VGA:** Camera module is calibrated only for VGA mode.

**Only\_HD:** Camera module is calibrated only for HD mode.

**Both\_VGA\_HD:** Camera module is calibrated for both VGA and HD modes.

# Structure

The details regarding Structures used will be discussed below.

## DeviceInfo

**Description:** This structure contains the information of a device that is connected to the host PC. It holds device name, VID, PID, device path and the serial number of the camera.

### Members:

```
typedef struct
{
    char deviceName[50];
    char vid[5];
    char pid[5];
    char devicePath[500];
    char serialNo[50];
}DeviceInfo;
```

### Member Description:

- **deviceName:** This member holds the deviceName of the camera.
- **pid:** This member holds the Product ID specific for DepthVista.
- **vid:** This member holds the Vendor ID specific for e-con Systems.
- **devicePath:** This member holds the path in which the device is enumerated.
- **serialNo:** This holds the serial number of the device.

## IntrinsicCalibParams

**Description:** This structure contains the intrinsic calibration values of a device that is connected to the host PC. It holds fx, fy, cx, cy, k1, k2, p1, p2, k3 values of the camera.

### Members:

```
typedef struct
{
    double fx, fy, cx, cy;
    double k1, k2, p1, p2, k3;
}IntrinsicCalibParams;
```

### Member Description:

- **fx**: Focal length of the device in the x-direction.
- **fy**: Focal length of the device in the y-direction.
- **cx**: x-coordinate of the principal point.
- **cy**: y-coordinate of the principal point.
- **k1**: Radial distortion coefficient.
- **k2**: Radial distortion coefficient.
- **p1**: Tangential distortion coefficient.
- **p2**: Tangential distortion coefficient.
- **k3**: Radial distortion coefficient.

## ExtrinsicCalibParams

**Description:** This structure contains the extrinsic calibration values of a device that is connected to the host PC. It holds rotationalVec, translationalVec vectors of the camera.

### Members:

```
typedef struct
{
    double rotationalVec[3][3];
    double translationalVec[3];
}ExtrinsicCalibParams;
```

### Member Description:

- **rotationalVec**: Rotational Matrix of the device.
- **translationalVec**: Translational Vector of the device.

## CalibrationParams

**Description:** This structure contains all the calibration values of the device. It holds depthCamVGAIntrinsic, rgbCamVGAIntrinsic, depthCamHDItrinsic, rgbCamHDItrinsic, VGAextrinsic, HDextrinsic.

### Members:

```
typedef struct
{
    IntrinsicCalibParams depthCamVGAIntrinsic,
    rgbCamVGAIntrinsic, depthCamHDItrinsic,
    rgbCamHDItrinsic;
    ExtrinsicCalibParams VGAextrinsic, HDextrinsic;
}calibrationParams;
```

### Member Description:

- **depthCamVGAIntrinsic:** IntrinsicCalibParams structure which contains Intrinsic calibration values of depth camera in VGA mode.
- **rgbCamVGAIntrinsic:** IntrinsicCalibParams structure which contains Intrinsic calibration values of RGB camera in VGA mode.
- **depthCamHDItrinsic:** IntrinsicCalibParams structure which contains Intrinsic calibration values of depth camera in HD mode.
- **rgbCamHDItrinsic:** IntrinsicCalibParams structure which contains Intrinsic calibration values of RGB camera in HD mode.
- **VGAextrinsic:** ExtrinsicCalibParams structure which contains Extrinsic calibration matrix values in VGA mode.
- **HDextrinsic:** IntrinsicCalibParams structure which contains Extrinsic calibration matrix values in HD mode.

## DeviceHandle

**Description:** This structure contains the DeviceHandle of a device that is connected to the host PC. It holds serialNo of the camera. This structure is used to differentiate multiple device.

### Members:

```
typedef struct
{
    char serialNo[50];
}DeviceHandle;
```

### Member Description:

- **serialNo:** Serial Number of the device

## UVCProp

**Description:** This structure contains the details of the UVC properties queried from the device. It holds the UVC Property ID, minimum, maximum, current, step and the default value of a particular UVC property.

### Members:

```
typedef struct
{
    int id;
    int min;
    int max;
    int cur;
    int step;
```



```
        int default_val;  
    }UVCProp;
```

**Member Description:**

- **int id:** UVC Property ID.
- **int min:** Minimum value of that particular UVC property.
- **int max:** Maximum value of that particular UVC property.
- **int cur:** Current value of that particular UVC property.
- **int step:** Step value of that particular UVC property.
- **int default\_val:** Default value of that particular UVC property.

## DepthPtr

**Description:** This structure contains the x co-ordinate and y co-ordinate of the depth position.

**Members:**

```
typedef struct  
{  
    int X;  
    int Y;  
}DepthPtr;
```

**Member Description:**

- **X:** This member holds the x co-ordinate of the depth position.
- **Y:** This member holds the y co-ordinate of the depth position.

## ToFFrame

**Description:** This structure contains the details of the frame obtained from camera. It holds the frame buffer, width, height, and size of the frame along with the pixel format.

**Members:**

```
typedef struct  
{  
    unsigned char* frame_data;  
    uint16_t width;  
    uint16_t height;  
    uint8_t pixel_format;  
    uint32_t total_size;  
    uint64_t time_stamp;
```

```
}ToFFrame;
```

#### Member Description:

- **unsigned char\* frame\_data:** This unsigned char pointer holds the address of the memory that holds the frame data.
- **uint16\_t width:** This holds the width of the frame.
- **uint16\_t height:** This holds the height of the frame.
- **uint8\_t pixel\_format:** 0 for UYVY, 1 for Y16 and 2 for RGB pixel format.
- **uint32\_t size:** This holds the size of the image buffer in bytes.
- **Uint64\_t time\_stamp:** This holds the time stamp of the frame.

## Frames

**Description:** This structure contains ToFFrame of all frame types. It holds rgb, ir, raw\_depth, depth\_colormap.

#### Members:

```
typedef struct
{
    ToFFrame rgb, ir, raw_depth, depth_colormap;
}Frames;
```

#### Member Description:

- **rgb:** ToFFrame structure which contains frame data of rgb frame.
- **ir:** ToFFrame structure which contains frame data of ir frame.
- **raw\_depth:** ToFFrame structure which contains frame data of raw depth frame.
- **depth\_colormap:** ToFFrame structure which contains frame data of depth colormap frame.

## IMUCONFIG\_TypeDef

**Description:** This structure contains the configuration details for IMU.

#### Members:

```
typedef struct
{
    uint8_t IMU_MODE;
    uint8_t ACC_AXIS_CONFIG;
    uint8_t ACC_SENSITIVITY_CONFIG;
    uint8_t GYRO_AXIS_CONFIG;
    uint8_t GYRO_SENSITIVITY_CONFIG;
    uint8_t IMU_ODR_CONFIG;
```

```
} IMUCONFIG_TypeDef;
```

### Member Description:

1. **IMU\_MODE** – The operating mode of the IMU.
  - IMU\_ACC\_GYRO\_DISABLE (0x00) - Accelerometer and Gyroscope Disabled.
  - IMU\_ACC\_ENABLE (0x01) - Accelerometer Enabled and Gyroscope Disabled.
  - IMU\_GYRO\_ENABLE (0x02) - Accelerometer Disabled and Gyroscope Enabled.
  - IMU\_ACC\_GYRO\_ENABLE (0x03) - Accelerometer and Gyroscope Enabled.
2. **ACC\_AXIS\_CONFIG** - The configuration for enabling/disabling an axis of Accelerometer, it is a bitmap. For example, to enable X axis alone use the code 0x01, to enable X and Z axis use 0x05 (0x01 | 0x04).
  - IMU\_ACC\_X\_Y\_Z\_ENABLE (0x07) - All 3 axis (X, Y, Z) enabled.
  - IMU\_ACC\_X\_ENABLE (0x01) - Enable X Axis.
  - IMU\_ACC\_Y\_ENABLE (0x02) - Enable Y Axis.
  - IMU\_ACC\_Z\_ENABLE (0x04) - Enable Z Axis.
3. **ACC\_SENSITIVITY\_CONFIG** - The configuration to set the sensitivity of the Accelerometer.
  - IMU\_ACC\_SENS\_2G (0x00) - Set sensitivity of 2g.
  - IMU\_ACC\_SENS\_4G (0x02) - Set sensitivity of 4g.
  - IMU\_ACC\_SENS\_8G (0x03) - Set sensitivity of 8g.
  - IMU\_ACC\_SENS\_16G (0x01) - Set sensitivity of 16g.
4. **GYRO\_AXIS\_CONFIG** - The configuration for enabling/disabling an axis of Gyroscope, it is a bitmap. For example, to enable X axis alone use the code 0x01, to enable X and Z axis use 0x05 (0x01 | 0x04).
  - IMU\_GYRO\_X\_Y\_Z\_ENABLE (0x07) - All 3 axis (X, Y, Z) enabled.
  - IMU\_GYRO\_X\_ENABLE (0x01) - Enable X Axis.
  - IMU\_GYRO\_Y\_ENABLE (0x02) - Enable Y Axis.
  - IMU\_GYRO\_Z\_ENABLE (0x04) - Enable Z Axis.

5. **GYRO\_SENSITIVITY\_CONFIG** - The configuration to set the sensitivity of the Gyroscope.
  - IMU\_GYRO\_SENS\_245DPS (0x00) - Set sensitivity of 245 degrees per second.
  - IMU\_GYRO\_SENS\_500DPS (0x02) - Set sensitivity of 500 degrees per second.
  - IMU\_GYRO\_SENS\_2000DPS (0x03) - Set sensitivity of 2000 degrees per second.
6. **IMU\_ODR\_CONFIG** - The configuration of Output Data Rate (ODR) of the IMU.
  - IMU\_ODR\_12\_5HZ (0x01) - ODR is 12.5 Hz
  - IMU\_ODR\_26HZ (0x02) - ODR is 26 Hz
  - IMU\_ODR\_52HZ (0x03) - ODR is 52 Hz
  - IMU\_ODR\_104HZ (0x04) - ODR is 104 Hz.
  - IMU\_ODR\_208HZ (0x05) - ODR is 208 Hz.
  - IMU\_ODR\_416HZ (0x06) - ODR is 416 Hz.
  - IMU\_ODR\_833HZ (0x07) – ODR is 833 Hz.
  - IMU\_ODR\_1666HZ (0x08) – ODR is 1666 Hz.

## IMUDATAINPUT\_TypeDef

**Description:** This structure contains the configuration details of IMU for Control IMU Capture.

### Members:

```
typedef struct
{
    uint8_t imu_update_mode;
    uint16_t imu_num_of_values;
} IMUDATAINPUT_TypeDef;
```

### Member Description:

1. **IMU\_UPDATE\_MODE** – The value update mode of the IMU.
  - IMU\_CONT\_UPDT\_EN (0x01) - Continuous Update of IMU Values Enabled.

- IMU\_CONT\_UPDT\_DIS (0x02) - Continuous Update of IMU Values Disabled.
2. **IMU\_NUM\_OF\_VALUES** – The number of values required, currently need to be set to one, other values support will be added later.

## IMUDATAOUTPUT\_TypeDef

**Description:** This structure contains the output data from IMU.

**Members:**

```
typedef struct
{
    uint16_t imu_value_id;
    double accX;
    double accY;
    double accZ;
    double gyroX;
    double gyroY;
    double gyroZ;
} IMUDATAOUTPUT_TypeDef;
```

**Member Description:**

- **IMU\_VALUE\_ID** – The ID for each set of values generated from the IMU, range 1-65535 with reload.
- **accX** – The accelerometer g value for the X axis.
- **accY** – The accelerometer g value for the Y axis.
- **accZ** – The accelerometer g value for the Z axis.
- **gyroX**– The gyroscope DPS (Degree per second) value for the X axis.
- **gyroY**– The gyroscope DPS value for the Y axis.
- **gyroZ**– The gyroscope DPS value for the Z axis.

**1. We have already installed OpenCV version in PC. Will it make conflict with the DepthVista SDK?**

Pre-installed OpenCV version will not make any conflict with DepthVistaSDK, because the libs and header files required by DepthVistaSDK are statically bound. This also allows the user to use any version of **OpenCV above 4.5.4**.

# Support

---

## **Contact Us**

If you need any support on DepthVista product, please contact us using the Live Chat option available on our website - <https://www.e-consystems.com/>

## **Creating a Ticket**

If you need to create a ticket for any type of issue, please visit the ticketing page on our website - <https://www.e-consystems.com/create-ticket.asp>

## **RMA**

To know about our Return Material Authorization (RMA) policy, please visit the RMA Policy page on our website - <https://www.e-consystems.com/RMA-Policy.asp>

## **General Product Warranty Terms**

To know about our General Product Warranty Terms, please visit the General Warranty Terms page on our website - <https://www.e-consystems.com/warranty.asp>

## Revision History

Rev	Date	Description	Author
1.0	21-April-2022	Initial Draft	Camera Products
1.1	13-May-2022	Application name changed	Camera Products
1.2	25-August-2022	Added IMU embedded data control API's.	Camera Products
1.3	30-September-2022	Added calibration read API's	Camera Products
1.4	02-November-2022	Added IMU APIs	Camera Products
1.5	04-November-2022	Added FAQ	Camera Products
1.6	18-May-2023	Added SDK changes	Camera Products
1.7	05-June-2023	Document Update	Camera Products
1.8	15-June-2023	Changed GetIMUValue API Parameter	Camera Products
1.9	10-October-2023	Added RGB-D mapping and Logging API's	Camera Products
1.10	20-January -2024	Added Flying pixel filter	Camera Products
1.11	16-March-2024	Added SDK changes	Camera Products