

DepthVista

DepthVista SDK API Manual



e-con Systems

Think Camera. Think e-con.

Version 1.2

e-con Systems

8/25/2022

Disclaimer

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

Contents

INTRODUCTION TO DEPTHVISTA	4
PREREQUISITES	4
WINDOW	4
LINUX	4
DESCRIPTION	4
SUPPORTED APIS	6
RESULT INITIALIZE()	6
RESULT DEINITIALIZE()	6
RESULT GETDEVICECOUNT(UINT32_T *GDEVICECOUNT)	7
RESULT GETDEVICELISTINFO(UINT32_T DEVICECOUNT, DEVICEINFO* GDEVICESLIST)	8
RESULT GETDEVICEINFO(UINT32_T DEVICEINDEX, DEVICEINFO* GDEVICE)	9
RESULT OPENDevice(UINT32_T DEVICEINDEX)	10
RESULT CLOSEDevice()	11
RESULT ISOPENED()	11
RESULT GETNEXTFRAME()	12
RESULT GETTOFFRAME(FRAMETYPE FRAMETYPE, TOFFRAME* FRAME)	12
INT GETFRAMESPERSECOND()	14
RESULT SETDATAMODE(DATAMODE SDATAMODE)	14
RESULT GETDATAMODE(DATAMODE* GDATAMODE)	16
RESULT SETDEPTHRANGE(UINT16_T SDEPTHRANGE)	17
RESULT GETDEPTHRANGE(UINT16_T *GDEPTHRANGE)	18
RESULT SETTOFCORING(UINT16_T STOFCORE)	19
RESULT GETTOFCORING(UINT16_T* GTOFCORE)	19
RESULT SETTOFIRGAIN(UINT16_T STOFIRGAIN)	20
RESULT GETTOFIRGAIN(UINT16_T* GTOFIRGAIN)	21
RESULT GETUVCCONTROL(INT32_T GCONTROLID, UVCPROP* GCONTROLVALUE)	22
RESULT SETUVCCONTROL(INT32_T SCONTROLID, INT32_T SCONTROLVALUE)	23
RESULT GETANTIFLICKERDETECTION(UINT8_T* GAFDETECT)	24
RESULT SETANTIFLICKERDETECTION(UINT8_T SAFDETECT)	25
RESULT GETSPECIALEFFECT(UINT8_T* GSPEFFECT)	26
RESULT SETSPECIALEFFECT(UINT8_T SSPEFFECT)	27
RESULT GETDENOISE(UINT8_T* GDENOISE)	28
RESULT SETDENOISE(UINT8_T SDENOISE)	29
RESULT GETAUTOEXPOSUREROI(UINT8_T* GROIMODE, UINT8_T* GWINSIZE)	30
RESULT SETAUTOEXPOSUREROI(UINT8_T SROIMODE, UINT32_T SWIDTH, UINT32_T SHEIGHT, UINT32_T SXCOR, UINT32_T SYCOR, UINT8_T SWINSIZE)	32
RESULT GETORIENTATION(UINT8_T* GORIENTATION)	33
RESULT SETORIENTATION(UINT8_T SORIENTATION)	34
RESULT GETFACEDETECTION(UINT8_T* GFACEDET, UINT8_T* GSTATUSSTRUCT, UINT8_T* GOVERLAYRECT)	35

RESULT SETFACEDETECTION(UINT8_T SFACEDET, UINT8_T SSTATUSSTRUCT, UINT8_T SOVERLAYRECT)	37
RESULT GETSMILEDETECTION(UINT8_T* GSMILEDET, UINT8_T* GSTATUSSTRUCT)	38
RESULT SETSMILEDETECTION(UINT8_T GSMILEDET, UINT8_T SSTATUSSTRUCT)	39
RESULT GETEXPOSURECOMPENSATION(UINT32_T* GEXPOCOMP)	40
RESULT SETEXPOSURECOMPENSATION(UINT32_T SEXPOCOMP)	41
RESULT GETFRAMERATECTRL(UINT8_T* GFRAMERATECTRL)	42
RESULT SETFRAMERATECTRL(UINT8_T SFRAMERATECTRL)	42
RESULT SETDEFAULT()	43
RESULT GETUNIQUEID(UINT64_T* GUNIQUEID)	44
RESULT READFIRMWAREVERSION(UINT8_T* GMAJORVERSION, UINT8_T* GMINORVERSION1, UINT16_T* GMINORVERSION2, UINT16_T* GMINORVERSION3)	44
RESULT GETBASEBOARDTEMPERATUREDATA(FLOAT* GBASEBOARDTEMPDATA)	45
RESULT GETLASERBOARDTEMPERATUREDATA(FLOAT* GLASERBOARDTEMPDATA)	46
RESULT SETAVGREGION(AVGREGION REGION)	47
RESULT SETFILTERTYPE(INT FILTERID, BOOL SELECTED)	47
RESULT SETMOUSEPOS(MOUSEPTR POS)	48
RESULT SETCURSORCOLOR(INT COLOR)	49
RESULT SETPLANARIZATION(INT PLANARIZE)	49
RESULT GETDEPTHIRVALUES(INT *GAVGDEPTH, INT *GSTDEPTH, INT *GAVGIR, INT *GSTDIR)	50
VOID REGISTERFRAMECALLBACK(FUNCTION<VOID(INT)> CB)	51
VOID REGISTERNOTIFICATIONCALLBACK(FUNCTION<VOID(INT)> CB)	51
RESULT UPDATEAVGXANDY(INT AVG_X, INT AVG_Y)	52
RESULT UPDATECOLORMAP(INT MIN, INT MAX, INT COLORMAP)	53
RESULT GETIMUEMBEDDEDATA(UINT8_T* GIMUDATA)	53
RESULT SETIMUEMBEDDEDATA(UINT8_T SIMUDATA)	54
ENUM	56
RESULT	56
UVCPROPID	57
FRAMETYPE	58
DATAMODE	58
AVGREGION	59
STRUCTURE	60
TOFFFRAME	60
UVCPROP	60
DEVICEINFO	61
MOUSEPTR	61
SUPPORT	63

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, a leading Embedded Product Design Services Company which specializes in advanced camera solutions.

DepthVista is a 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 onsemiTM. 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.

This document highlights the APIs that are currently used in the sample application of DepthVista.

Prerequisites

The prerequisites are described below.

Window

The Visual C++ redistributable packages install runtime components of Visual C++ libraries that are required to run applications developed using Visual Studio 2017 on a computer. These packages install runtime components of the C Runtime (CRT) and Standard C++.

- Visual C++ redistributable for Visual Studio 2017.
- Build environment support from Visual Studio 2017 and higher Versions.

Linux

To use the APIs in SDK in Linux, either cmake or Qt Creator is required.

- Cmake version 3.5 and above. Cmake GUI can also be used.
- Qt Creator 4.11.1 and higher version is required.

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.

Supported APIs

The details regarding the supported APIs are explained below.

Result Initialize()

This function initializes all the other APIs and it must be called first before calling any other APIs.

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 on the device and clears 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 *gDeviceCount)

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 *gDeviceCount	[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 gdeviceCount;
    if(GetDeviceCount(&gdeviceCount)>0)
    {
        printf("Number of devices connected is %d \r\n\n",
gdeviceCount );
    }
    else
    {
        printf("GetDeviceCount Failed\r\n\n");
    }
}
```


Result GetDeviceListInfo(uint32_t deviceCount, DeviceInfo* gDevicesList)

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

Parameters	Description
uint32_t gDeviceCount	[IN] number of devices connected.
DeviceInfo *gDevicesList	[OUT] list of DeviceInfo whose size is gDeviceCount

Member Description:

- char deviceName[50]: This member holds the deviceName of the camera.
- char pid[5]: This member holds the Product ID specific for DepthVista.
- char vid[5]: This member holds the Vendor ID specific for e-con Systems.
- char devicePath[250]: This member holds the path in which the device is enumerated.
- char serialNo[50]: This holds the serial number of the device.

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 gDeviceCount; // get device count from
    GetDeviceCount

    DeviceInfo *gDevicesList = new
    DeviceInfo[gDeviceCount];

    if (GetDeviceListInfo(gDeviceCount, gDevicesList )>0)
    {
        uint16_t index;
        for(index = 0; index < gDeviceCount; index++)
        {
            printf("Device name is %s",
            gDeviceList[index].deviceName);
            printf("Device PID is %s",
            gDeviceList[index].pid);
            printf("Device VID is %s",
            gDeviceList[index].vid);
        }
    }
}
```

```

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

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

    }

}

else
{

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

}

}

```

Result GetDeviceInfo(uint32_t deviceIndex, DeviceInfo* gDevice)

This function returns the information of the devices at the index mentioned in deviceIndex connected to the host PC. The information is contained in a *DeviceInfo* structure. DeviceInfo structure holds the device name, VID, PID, device path and the serial number of the camera.

Parameters	Description
uint32_t deviceIndex	[IN] device index whose device information is to be known
DeviceInfo *gDevicesList	[OUT] DeviceInfo struct that contains the information of the device at index deviceIndex

Member Description:

- char deviceName[50]: This member holds the deviceName of the camera.
- char pid[5]: This member holds the Product ID specific for DepthVista.
- char vid[5]: This member holds the Vendor ID specific for e-con Systems.
- char devicePath[250]: This member holds the path in which the device is enumerated.
- char serialNo[50]: This holds the serial number of the device.

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 GetDeviceInfo_(
{

    uint32_t deviceIndex; // 0 to gDeviceCount -1

```

```
DeviceInfo *gDevicesInfo;
if (GetDeviceInfo(deviceIndex, gDevicesInfo )>0 )
{
    printf("GetDeviceInfo Success\r\n");
}
else
{
    printf("GetDevicetInfo failed\r\n");
}
}
```

Result OpenDevice(uint32_t deviceIndex)

This function opens the device specified by deviceIndex. The device must be subsequently closed using CloseDevice().

Parameters	Description
uint32_t deviceIndex	[IN] device index of the device which is to be opened

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 in invalid.

Sample Code:

```
void OpenDevice_()
{
    uint32_t deviceIndex; // 0 to gDeviceCount -1
    if (OpenDevice(deviceIndex) >0)
    {
        printf("OpenDevice Success\r\n");
    }
    else
    {
        printf("OpenDevice failed\r\n");
    }
}
```

Result CloseDevice()

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

Parameter	Description
Nil	N/A

Return:

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

Sample Code:

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

Result IsOpened()

This function return whether the device is opened or not.

Parameter	Description
Nil	N/A

Return:

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

Sample Code:

```
void IsOpened_()
{
    if(IsOpened()>0 )
    {
        printf("Device is opened\r\n");
    }
    else
```

```

    {
        printf("Device is closed \r\n");
    }
}

```

Result GetNextFrame()

This function retrieves the image frame from the device that was opened using `OpenDevice()`. This API must be invoked before retrieving frame data using `GetToFFrame()`.

Parameter	Description
Nil	N/A

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.

Sample Code:

```

void GetNextFrame_()
{
    if(GetNextFrame ()>0)
    {
        printf("GetNextFrame success\r\n");
        // call to GetToFFrame();
    }
    else
    {
        printf("GetNextFrame failed\r\n");
    }
}

```

Result GetToFFrame(FrameType frameType, ToFFrame* frame)

This function returns the image data of *ToFFrame* structure for the current frame specified by *FrameType* from the device that was opened using `OpenDevice()`. Before invoking this API, invoke `GetNextFrame()` to capture one image frame from the device.

Parameters	Description
<div>FrameType</div> <div>frameType</div>	[IN]Type of frame for which the image data is needed. The frame types supported are as follows:

- **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.

[OUT]Address of the ToFFrame structure in which the image data is filled. The structure member are as follows:

```
ToFFrame*  
Frame
```

- **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.

Return:

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

Sample Code:

```
void GetToFFrame_  
{  
    ToFFrame* frame;  
    uint8_t frameType;  
    if(GetToFFrame(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");
}
}

```

int getFramesPerSecond()

This function returns the number of frames obtained per second.

Parameters	Description
Nil	N/A

Return:

Int: returns the frames per second.

Sample Code:

```

void GetFramesPerSecond_()
{
    printf("FPS : %d\r\n", getFramesPerSecond());
}

```

Result SetDataMode(DataMode sDataMode)

This function sets the stream mode specified sDataMode. All the supported data modes are listed in the enum *DataMode*.

Parameters	Description
<div style="border: 1px solid #ccc; padding: 5px; width: fit-content;"> DataMode sDataMode </div>	<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"> ModeUnknown: This datamode is used as default in the application.

- **Depth_IR_Mode:** Output depth frame (640 x 480) and IR frame (640 x 480) at 30 FPS.
- **Depth_Mode:** Output depth frame(640x480) 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.

Return:

- Returns **Ok** when the Data mode is set successfully.
- Returns **CameraNotOpened** when the device is not opened.
- Returns **SysCallFail** when the system call API failed.
- Returns **HidWriteFail** when the buffer cannot be written to the hid handle.
- Returns **TimeoutError** when the wait event timed out.
- Returns **InvalidHidHandle** when the buffer obtained is invalid.
- Returns **Others** when an unknown error occurred.

Sample Code:

```
void SetDataMode_()
{
    DataMode sDataMode;
    if(SetDataMode(sDataMode)>0)
    {
        printf("SetDataMode success\r\n");
    }
    else
    {
        printf("SetDataMode failed\r\n");
    }
}
```


Result GetDataMode(DataMode* gDataMode)

This function gets the stream mode in which the device is streaming currently. All the supported data modes are listed in the enum *DataMode*.

Parameters	Description
<code>DataMode* gDataMode</code>	[OUT]gDataMode 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 **SysCallFail** when the system call API failed.

Sample Code:

```
void GetDataMode_()
{
    DataMode *gDataMode;
    if (GetDataMode(gDataMode) > 0)
    {
        switch (gDataMode)
        {
            case ModeUnknown:
                printf("Current data modes is ModeUnkown\r\n");
            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(uint16_t sDepthRange)

This function sets the depth range specified by sDepthRange. The device supports two depth range as follows:

- **Far Mode:** Effective depth range in this mode is between 1000 mm to 6500 mm.
- **Near Mode:** Effective depth range in this mode is between 200 mm to 1200 mm.

Parameters	Description
uint16_t sDepthRange	[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 **SysCallFail** when the system call API failed.

Sample Code:

```

void SetDepthMode_()
{
    uint16_t sDepthRange;
    if (SetDepthRange (sDepthRange)>0)
    {
        printf("SetDepthRange success\r\n");
    }
}

```

```

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

```

Result GetDepthRange(uint16_t *gDepthRange)

This function gets the depth range in which the device is streaming currently. The device supports two depth range as follows:

- **Far Mode:** Effective depth range in this mode is between 1000 mm to 6500 mm.
- **Near Mode:** Effective depth range in this mode is between 200 mm to 1200 mm.

Parameters	Description
uint16_t* gDepthRange	[OUT]gDepthRange 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 **SysCallFail** when the system call API failed.

Sample Code:

```

void GetDepthMode_()
{
    uint16_t* gDepthRange;
    if (GetDepthRange(gDepthRange)>0)
    {
        if(gDepthRange == 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(uint16_t sTOFCore)

This function sets the TOF coring value specified by sTOFCore.

Coring is function API that facilitates removal of signal values if they are less than the threshold value. This is used to random optical noises or pixel noises.

Parameters	Description
uint16_t sTOFCore	[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 **SysCallFail** when the system call API failed.

Sample Code:

```
void SetTOFCoring_  
{  
    uint16_t sTOFCore;  
    if(SetTOFCoring(sTOFCore)>0)  
    {  
        printf("SetTOFCoring success\r\n");  
    }  
    else  
    {  
        printf("SetTOFCoring failed\r\n");  
    }  
}
```

Result GetTOFCoring(uint16_t* gTOFCore)

This function gets the TOF coring value from the device.

Coring is a function that facilitates removal of signal values if they are less than the threshold value. This is used to random optical noises or pixel noises.

Parameters	Description
<code>uint16_t* gTOFCore</code>	[OUT]gTOFCore 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 **SysCallFail** when the system call API failed.

Sample API:

```
void GetTOFCoring_()
{
    uint16_t gTOFCore;
    if(GetTOFCoring(&gTOFCore)>0)
    {
        printf("The Coring value is %d\r\n",gTOFCore);
    }
    else
    {
        printf("GetTOFCoring failed\r\n");
    }
}
```

Result SetTOFIRGain(uint16_t sTOFIRGain)

This function sets the TOF IR gain value specified by sTOFIRGain.

TOF gain is a software gain added to the IR display to increase the brightness of the IR scene.

Parameters	Description
<code>uint16_t sTOFIRGain</code>	[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 sTOFIRGain;
```

```

if(SetTOFIRGain(sTOFIRGain)>0)
{
    printf("SetTOFIRGain success\r\n");
}
else
{
    printf("SetTOFIRGain failed\r\n");
}
}

```

Result GetTOFIRGain(uint16_t* gTOFIRGain)

This function gets the TOF IR gain value. The default value is 16.

TOF gain is a software gain added to the IR display to increase the brightness of the IR scene.

Parameters	Description
uint16_t* gTOFIRGain	[OUT] gTOFIRGain is filled with the TOF IR Gain value.

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* gTOFIRGain;
    if(GetTOFIRGain(gTOFIRGain)>0)
    {
        printf("TOF IR Gain value is %d\r\n",
gTOFIRGain);
    }
    else
    {
        printf("GetTOFIRGain failed\r\n");
    }
}

```

Result GetUVCControl(int32_t gControlID, UVCCProp* gControlValue)

This function gets the UVC control value gControlValue of control specified by gControlID. UVC controls supported by DepthVista are as follows:

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

Parameters	Description
<code>int32_t gControlID</code>	[IN] gControlID for the specific UVC Control. Each UVC control have a specific control ID listed in <i>UVCCPropID</i> enum.
<code>UVCCProp* gControlValue</code>	[OUT] pointer to the <i>UVCCPropID</i> structure. <i>UVCCPropID</i> 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 gControlID;
    UVCCProp gControlValue;
    if(GetUVCControl(gControlID, &gControlValue)>0 )
    {
        printf("gControlValue.id : %d\r\n",
gControlValue.id);
        printf("gControlValue.cur : %d\r\n",
gControlValue.cur);
        printf("gControlValue.min : %d\r\n",
gControlValue.min);
        printf("gControlValue.max : %d\r\n",
gControlValue.max);
    }
}
```

```

        printf("gControlValue.step : %d\r\n",
gControlValue.step);

        printf("gControlValue.default_val : %d\r\n",
gControlValue.Default_val);

    }

    else

    {

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

    }

}

```

Result SetUVCControl(int32_t sControlID, int32_t sControlValue)

This function sets the current value of the UVC control specified by sControlID. UVC controls supported by DepthVista are as follows:

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

Parameters	Description
<code>int32_t sControlID</code>	[IN] sControlID for the specific UVC control. Each UVC control have a specific control ID listed in <i>UVCCPropID</i> enum.
<code>int32_t sControlValue</code>	[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 sControlID;
int32_t sControlValue;

```



```

if(SetUVCControl(sControlID, sControlValue)>0 )
{
    printf("SetUVCControl success\r\n");
}
else
{
    printf("SetUVCControl failed\r\n");
}
}

```

Result GetAntiFlickerDetection(uint8_t* gAFDetect)

This function gets the anti-flicker detection mode from the device. 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 are off mode for flicker avoidance, auto mode, or manual mode (50 Hz and 60 Hz).

Parameters	Description
<code>uint8_t* gAFDetect</code>	[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"> • 0 represents Auto Mode • 1 represents Manual 50 Hz • 2 represents Manual 60 Hz • 3 represents Disabled

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 **TimeoutError** when the wait event timed out.
- Returns **InvalidHidHandle** when the buffer obtained is invalid.
- Returns **Others** when an unknown error occurred.

Sample Code:

```

void GetAntiFlickerDetection_()
{
    uint8_t gAFDetect;
    if(GetAntiFlickerDetection(&gAFDetect)>0 )
    {
        switch(gAFDetect)

```

```

{
    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(uint8_t sAFDetect)

This function sets an anti-flicker detection mode specified by sAFDetect to the device. 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 are off mode for flicker avoidance, auto mode, or manual mode (50 Hz and 60 Hz).

Parameters	Description
<code>uint8_t sAFDetect</code>	[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"> • 0 represents Auto Mode • 1 represents Manual 50 Hz • 2 represents Manual 60 Hz • 3 represents Disabled

Return:

- Returns **Ok** when the Anti-Flicker detection is set 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 **TimeoutError** when the wait event timed out.
- Returns **InvalidHidHandle** when the buffer obtained is invalid.

- Returns **Others** when an unknown error occurred.

Sample Code:

```
void SetAntiFlickerDetection_()
{
    uint8_t sAFDetect;
    if (SetAntiFlickerDetection(sAFDetect)>0 )
    {
        printf("SetAntiFlickerDetection success\r\n");
    }
    else
    {
        printf("SetAntiFlickerDetection failed\r\n");
    }
}
```

Result GetSpecialEffect(uint8_t* gSplEffect)

This function gets the processed image special effect filters applied to the preview. The four special effects are normal, black and white, grayscale, negative and sketch.

Parameters	Description
<code>uint8_t* gSplEffect</code>	<p>[OUT] pointer to uint8_t variable that is filled with the special effect mode on success of the API. The resultant values are:</p> <ul style="list-style-type: none"> • 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 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 **TimeoutError** when the wait event timed out.
- Returns **InvalidHidHandle** when the buffer obtained is invalid.
- Returns **Others** when an unknown error occurred.

Sample Code:

```
void GetSpecialEffect_()
{
```

```
uint8_t gSplEffect;
    if (GetSpecialEffect(&gSplEffect)>0 )
    {
        switch(gSplEffect)
        {
            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(uint8_t sSplEffect)

This function applies the special effect filter specified by sSplEffect to the preview. The four special effects are normal, black and white, grayscale, negative and sketch.

Parameters	Description
<code>uint8_t sSplEffect</code>	[IN] uint8_t variable containing the special effect to be set. <ul style="list-style-type: none"> • 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 **NoPropertyValueGet** when the specified control cannot be obtained.
- Returns **HidWriteFail** when the buffer cannot be written to the hid handle.
- Returns **TimeoutError** when the wait event timed out.
- Returns **InvalidHidHandle** when the buffer obtained is invalid.
- Returns **Others** when an unknown error occurred.

Sample Code:

```
void SetSpecialEffect_()
{
    uint8_t sSplEffect;
    if(SetSpecialEffect(sSplEffect)>0 )
    {
        printf("SetSpecialEffect failed\r\n");
    }
    else
    {
        printf("SetSpecialEffect failed\r\n");
    }
}
```

Result GetDenoise(uint8_t* gDenoise)

This function gets the De-noise value from the device.

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

Parameters	Description
<code>uint8_t* gDenoise</code>	[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 **TimeoutError** when the wait event timed out.
- Returns **InvalidHidHandle** when the buffer obtained is invalid.
- Returns **Others** when an unknown error occurred.

Sample Code:

```
void GetDenoise_()
{
    uint8_t gDenoise;
    if(GetDenoise(&gDenoise)>0)
    {
        printf("De-noise value is %d\r\n", gDenoise);
    }
    else
    {
        printf("GetDenoise failed\r\n");
    }
}
```

Result SetDenoise(uint8_t sDenoise)

This function sets the De-noise value specified by the sDenoise.

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

Parameters	Description
<code>uint8_t sDenoise</code>	[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 **NoPropertyValueGet** when the specified control cannot be obtained.
- Returns **HidWriteFail** when the buffer cannot be written to the hid handle.
- Returns **TimeoutError** when the wait event timed out.
- Returns **InvalidHidHandle** when the buffer obtained is invalid.
- Returns **Others** when an unknown error occurred.

Sample Code:

```
void SetDenoise_()
{
```

```
uint8_t sDenoise;
if (SetDenoise(sDenoise) > 0)
{
    printf("SetDenoise success\r\n");
}
else
{
    printf("SetDenoise failed\r\n");
}
}
```

Result GetAutoExposureROI(uint8_t* gROIMode, uint8_t* gWinSize)

This function gets an auto exposure algorithm from the device.

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
<code>uint8_t* gROIMode</code>	<p>[OUT] pointer to uint8_t variable which is filled with the Auto Exposure ROI Mode on success of this API. The values obtained are:</p> <ul style="list-style-type: none"> • 1 for Face ROI • 2 for Full ROI • 3 for Manual ROI • 4 for Disabled Auto Exposure
<code>uint8_t* gWinSize</code>	<p>[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.</p>

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 **TimeoutError** when the wait event timed out.
- Returns **InvalidHidHandle** when the buffer obtained is invalid.
- Returns **Others** when an unknown error occurred.

Sample Code:

```
void GetAutoExposureROI_()
{
    uint8_t gROIMode;
    uint8_t gWinSize;
    if (GetAutoExposureROI(&gROIMode, &gWinSize)>0)
    {
        switch(gROIMode)
        {
            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(uint8_t sROIMode, uint32_t sWidth, uint32_t sHeight, uint32_t sXCor, uint32_t sYCor, uint8_t sWinSize)

This function sets an auto exposure algorithm specified by sROIMode and the window size specified by sWinSize.

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
uint8_t sROIMode	[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"> • 1 for Face ROI • 2 for Full ROI • 3 for Manual ROI • 4 for Disabled Auto Exposure
uint32_t sWidth	[IN] This parameter is valid only when the sROIMode parameter is in Manual ROI. Specifies the width of the ROI.
uint32_t sHeight	[IN] This parameter is valid only when the sROIMode parameter is in Manual ROI. Specifies the height of the ROI.
uint32_t sXCor	[IN] This parameter is valid only when the sROIMode parameter is in Manual ROI. Specifies the x co-ordinate of the top left corner in ROI.
uint32_t sYCor	[IN] This parameter is valid only when the sROIMode parameter is in Manual ROI. Specifies the y co-ordinate of the top left corner in ROI.
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.

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 **NoPropertyValueGet** when the specified control cannot be obtained.
- Returns **HidWriteFail** when the buffer cannot be written to the hid handle.
- Returns **TimeoutError** when the wait event timed out.
- Returns **InvalidHidHandle** when the buffer obtained is invalid.
- Returns **Others** when an unknown error occurred.

Sample Code:

```
void SetAutoExposureROI_()
{
    uint8_t sROIMode = 3, sWinSize = 1;
    uint32_t sWidth = 160, sHeight = 90;
    uint32_t sXCor = 100, sYCor = 200;
    if(SetAutoExposureROI(sROIMode, sWidth, sHeight, sXCor,
        sYCor, gWinSize)>0)
    {
        printf("SetAutoExposureROI success\r\n");
    }
    else
    {
        printf("SetAutoExposureROI failed\r\n");
    }
}
```

Result GetOrientation(uint8_t* gOrientation)

This function gets the orientation of the scene from the camera. Horizontal flip, vertical flip and both flips simultaneously are supported.

Parameters	Description
<div>uint8_t*</div> <div>gOrientation</div>	[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"> • 1 for Horizontal Flip • 2 for Vertical Flip • 3 for both Horizontal and Vertical Flip

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 **TimeoutError** when the wait event timed out.
- Returns **InvalidHidHandle** when the buffer obtained is invalid.
- Returns **Others** when an unknown error occurred.

Sample Code:

```
void GetOrientation_()
{
    uint8_t gOrientation;
    if(GetOrientation(&gOrientation)>0)
    {
        switch(gOrientation)
        {
            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(uint8_t sOrientation)

This function sets the orientation of the scene specified by sOrientation. Horizontal flip, vertical flip and both flips simultaneously are supported.

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

Return:

- Returns **Ok** when the Orientation is set 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 **TimeoutError** when the wait event timed out.
- Returns **InvalidHidHandle** when the buffer obtained is invalid.
- Returns **Others** when an unknown error occurred.

Sample Code:

```
void SetOrientation_  
{  
    uint8_t sOrientation = 2;  
    if(SetOrientation(sOrientation)>0)  
    {  
        printf("SetOrientation success\r\n");  
    }  
    else  
    {  
        printf("SetOrientation failed\r\n");  
    }  
}
```

Result GetFaceDetection(uint8_t* gFacedet, uint8_t* gStatusStruct, uint8_t* gOverlayRect)

This function gets the face detection option from the device.

Parameters	Description
<code>uint8_t* gFacedet</code>	[OUT] pointer to <code>uint8_t</code> variable which is filled with the face detection option, on success of this API. The values obtained are <ul style="list-style-type: none"> 0 for face detection disabled 1 for face detection enabled
<code>uint8_t* gStatusStruct</code>	[OUT] pointer to <code>uint8_t</code> variable which is filled with status structure option, on success of this API. The values obtained are <ul style="list-style-type: none"> 0 for status structure disabled 1 for status structure enabled
<code>uint8_t* gOverlayRect</code>	[OUT] pointer to <code>uint8_t</code> variable which is filled with overlay rectangle option, on success of this API. The values obtained are <ul style="list-style-type: none"> 0 for overlay rectangle disabled 1 for overlay rectangle enabled

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 **TimeoutError** when the wait event timed out.
- Returns **InvalidHidHandle** when the buffer obtained is invalid.
- Returns **Others** when an unknown error occurred.

Sample Code:

```
void GetFaceDetection_()
{
uint8_t gFacedet, gStatusStruct, gOverlayRect;
if (GetFaceDetection(&gFacedet, &gStatusStruct,
&gOverlayRect)>0)
{
    if(!gFacedet)
        printf("Face detection mode is disabled\r\n");
    else
        printf("Face detection mode is enabled\r\n");
    if(!gStatusStruct)
        printf("Status structure mode is
disabled\r\n");
}
```

```

    else
        printf("Status structure mode is enabled\r\n");
        if(!gOverlayRect)
            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(uint8_t sFacedet, uint8_t sStatusStruct, uint8_t sOverlayRect)

This function sets the face detection option specified by sFacedet, sStatusStruct and sOverlayRect.

Parameters	Description
uint8_t sFacedet	[IN] uint8_t variable with the face detection option that is to be set. The values can be: <ul style="list-style-type: none"> 0 for face detection disabled 1 for face detection enabled
uint8_t sStatusStruct	[IN] uint8_t variable with the status structure option that is to be set. The values can be: <ul style="list-style-type: none"> 0 for status structure disabled 1 for status structure enabled
uint8_t sOverlayRect	[IN] uint8_t variable with the overlay rectangle option that is to be set. The values can be: <ul style="list-style-type: none"> 0 for overlay rectangle disabled 1 for overlay rectangle enabled

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 **NoPropertyValueGet** when the specified control cannot be obtained.
- Returns **HidWriteFail** when the buffer cannot be written to the hid handle.
- Returns **TimeoutError** when the wait event timed out.

- Returns **InvalidHidHandle** when the buffer obtained is invalid.
- Returns **Others** when an unknown error occurred.

Sample Code:

```
void SetFaceDetection_()
{
    uint8_t sFacedet = 1, sStatusStruct = 0, sOverlayRect = 1;
    if(SetFaceDetection(sFacedet, sStatusStruct, sOverlayRect)>0)
    {
        printf("SetFaceDetection success\r\n");
    }
    else
    {
        printf("SetFaceDetection failed\r\n");
    }
}
```

Result GetSmileDetection(uint8_t* gSmiledet, uint8_t* gStatusStruct)

This function gets the smile detection option from the device.

Parameters	Description
uint8_t* gSmiledet	[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"> • 0 for smile detection disabled • 1 for smile detection enabled
uint8_t* gStatusStruct	[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"> • 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 **TimeoutError** when the wait event timed out.

- Returns **InvalidHidHandle** when the buffer obtained is invalid.
- Returns **Others** when an unknown error occurred.

Sample Code:

```
void GetSmileDetection_()
{
    uint8_t gSmiledet, gStatusStruct;
    if(GetSmileDetection(&gSmiledet, &gStatusStruct)>0)
    {
        if(!gSmiledet)
            printf("Smile detection mode is disabled\r\n");
        else
            printf("Smile detection mode is enabled\r\n");
        if(!gStatusStruct)
            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(uint8_t sSmiledet, uint8_t sStatusStruct)

This function sets the smile detection option specified by sSmiledet, sStatusStruct.

Parameters	Description
uint8_t sSmiledet	[IN] uint8_t variable with the smile detection option that is to be set. The values can be: <ul style="list-style-type: none"> • 0 for face detection disabled • 1 for face detection enabled
uint8_t sStatusStruct	[IN] uint8_t variable with the status structure option that is to be set. The values can be: <ul style="list-style-type: none"> • 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 set 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 **TimeoutError** when the wait event timed out.
- Returns **InvalidHidHandle** when the buffer obtained is invalid.
- Returns **Others** when an unknown error occurred.

Sample Code:

```
void SetSmileDetection_()
{
    uint8_t sSmiledet = 1, sStatusStruct = 0;
    if (SetSmileDetection(sSmiledet, sStatusStruct)>0)
    {
        printf("SetSmileDetection success\r\n");
    }
    else
    {
        printf("SetSmileDetection failed\r\n");
    }
}
```

Result GetExposureCompensation(uint32_t* gExpoComp)

This function gets the exposure compensation from the device.

Parameters	Description
<div>uint32_t*</div> <div>gExpoComp</div>	[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 **TimeoutError** when the wait event timed out.
- Returns **InvalidHidHandle** when the buffer obtained is invalid.
- Returns **Others** when an unknown error occurred.

Sample Code:

```
void GetExposureCompensation_()
{
    uint32_t gExpoComp;
    if (GetExposureCompensation(&gExpoComp)>0)
    {
        printf("Exposure compensation value is
%d\r\n", gExpoComp);
    }
    else
    {
        printf("GetExposureCompensation failed\r\n");
    }
}
```

Result SetExposureCompensation(uint32_t sExpoComp)

This function sets the exposure compensation specified by sExpoComp.

Parameters	Description
uint32_t sExpoComp	[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 **NoPropertyValueGet** when the specified control cannot be obtained.
- Returns **HidWriteFail** when the buffer cannot be written to the hid handle.
- Returns **TimeoutError** when the wait event timed out.
- Returns **InvalidHidHandle** when the buffer obtained is invalid.
- Returns **Others** when an unknown error occurred.

Sample Code:

```
void SetExposureCompensation_()
{
    uint32_t sExpoComp;
    if (SetExposureCompensation(sExpoComp)>0)
    {
        printf("SetExposureCompensation failed\r\n");
    }
    else
    {
        printf("SetExposureCompensation failed\r\n");
    }
}
```

```
}
}
```

Result GetFrameRateCtrl(uint8_t* gFrameRateCtrl)

This function gets the frame rate control value from the device.

Parameters	Description
uint8_t* gFrameRateCtrl	[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 **TimeoutError** when the wait event timed out.
- Returns **InvalidHidHandle** when the buffer obtained is invalid.
- Returns **Others** when an unknown error occurred.

Sample Code:

```
void GetFrameRateCtrl_()
{
    uint8_t gFrameRateCtrl;
    if (GetFrameRateCtrl (&gFrameRateCtrl) > 0)
    {
        printf("Frame Rate control value is %d\r\n",
            gFrameRateCtrl);
    }
    else
    {
        printf("GetFrameRateCtrl failed\r\n");
    }
}
```

Result SetFrameRateCtrl(uint8_t sFrameRateCtrl)

This function sets the frame rate control value that is specified by sFrameRateCtrl.

Parameters	Description
uint8_t sFrameRateCtrl	[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 **NoPropertyValueGet** when the specified control cannot be obtained.
- Returns **HidWriteFail** when the buffer cannot be written to the hid handle.
- Returns **TimeoutError** when the wait event timed out.
- Returns **InvalidHidHandle** when the buffer obtained is invalid.
- Returns **Others** when an unknown error occurred.

Sample Code:

```
void SetFrameRateCtrl_()
{
    uint8_t sFrameRateCtrl;
    if(SetFrameRateCtrl(sFrameRateCtrl)>0)
        printf("SetFrameRateCtrl success\r\n");
    else
        printf("SetFrameRateCtrl failed\r\n");
}
```

Result SetDefault()

This function sets all the HID control to default values on the device.

Parameters	Description
Nil	N/A

Return:

- Returns **Ok** when all the HID controls are set to default 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 **TimeoutError** when the wait event timed out.
- Returns **InvalidHidHandle** when the buffer obtained is invalid.
- Returns **Others** when an unknown error occurred.

Sample Code:

```
void SetDefault_()
{
```

```

if (SetDefault() > 0)
    printf("SetDefault success\r\n");
else
    printf("SetDefault failed\r\n");
}

```

Result GetUniqueID(uint64_t* gUniqueID)

This function gets the unique ID from the device.

Parameters	Description
uint64_t* gUniqueID	[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 **TimeoutError** when the wait event timed out.
- Returns **InvalidHidHandle** when the buffer obtained is invalid.
- Returns **Others** when an unknown error occurred.

Sample Code:

```

void GetUniqueID_()
{
    uint64_t gUniqueID;
    if (GetUniqueID(&gUniqueID) > 0)
        printf("Unique ID of this device is %d\r\n",
gUniqueID);
    else
        printf("GetUniqueID failed\r\n");
}

```

Result readFirmwareVersion(uint8_t* gMajorVersion, uint8_t* gMinorVersion1, uint16_t* gMinorVersion2, uint16_t* gMinorVersion3)

This function gets the firmware version from the device.

Parameters	Description
uint8_t* gMajorVersion	[OUT] pointer to uint8_t variable to store the Major version of firmware.
uint8_t* gMinorVersion1	[OUT] pointer to uint8_t variable to store the Minor version 1 of firmware.

<code>uint16_t* gMinorVersion2</code>	[OUT] pointer to uint16_t variable to store the Minor version 2 of firmware.
<code>uint16_t* gMinorVersion3</code>	[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 **TimeoutError** when the wait event timed out.
- Returns **InvalidHidHandle** when the buffer obtained is invalid.
- Returns **Others** when an unknown error occurred.

Sample Code:

```
void readFirmwareVersion_()
{
    uint8_t gMajorVersion = 0, gMinorVersion1 = 0;
    uint16_t gMinorVersion2 = 0, gMinorVersion3 = 0;
    if(readFirmwareVersion(&gMajorVersion, &gMinorVersion1,
    &gMinorVersion2, &gMinorVersion3)>0)
    {
        printf("Firmware version is %d.%d.%d.%d\r\n",
        gMajorVersion, gMinorVersion1, gMinorVersion2,
        gMinorVersion3);
    }
    else
    {
        printf("readFirmwareVersion failed\r\n");
    }
}
```

Result GetBaseBoardTemperatureData(float* gBaseBoardTempData)

This function gets the base board temperature data from the device. The device consists of two boards, base board, and laser board. Temperature of the base board is read in this API.

Parameters	Description
<code>float* gBaseBoardTempData</code>	[OUT] Pointer to a float variable that will be filled with the base board temperature data, on success of this API.

Return:

- Returns **Ok** when the base board temperature 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 **TimeoutError** when the wait event timed out.
- Returns **InvalidHidHandle** when the buffer obtained is invalid.
- Returns **Others** when an unknown error occurred.

Sample Code:

```
void GetBaseBoardTemperatureData_()
{
    float gBaseBoardTempData;
    if (GetBaseBoardTemperatureData (&gBaseBoardTempData) > 0)
        printf("The base board temperature is %f\r\n",
            gBaseBoardTempData);
    else
        printf("GetBaseBoardTemperatureData failed\r\n");
}
```

Result GetLaserBoardTemperatureData(float* gLaserBoardTempData)

This function gets the Laser board temperature data from the device. The device consists of two boards, base board, and laser board. Temperature of the laser board is read in this API.

Parameters	Description
float* gLaserBoardTempData	[OUT] Pointer to a float variable that will be filled with the laser board temperature data, on success of this API.

Return:

- Returns **Ok** when the laser board temperature 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 **TimeoutError** when the wait event timed out.
- Returns **InvalidHidHandle** when the buffer obtained is invalid.
- Returns **Others** when an unknown error occurred.

Sample Code:

```
void GetLaserBoardTemperatureData_()
{
    float gLaserBoardTempData;
```

```

if (GetLaserBoardTemperatureData (&gLaserBoardTempData) > 0)
    printf("The laser board temperature is %f\r\n",
gLaserBoardTempData);
else
    printf("GetLaserBoardTemperatureData failed\r\n");
}

```

Result SetAvgRegion(AvgRegion region)

This function sets the average depth and IR calculation area specified by region.

Parameters	Description
<u>AvgRegion</u> region	[IN] <i>AvgRegion</i> 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;
    if (SetAvgRegion(region) > 0)
        printf("SetAvgRegion success\r\n");
    else
        printf("SetAvgRegion failed\r\n");
}

```

Result SetFilterType(int filterID, bool selected)

This function sets or resets the type of filter specified by **filterID**, based on **selected**.

Parameters	Description
<u>int</u> filterID	[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"> • 0 for Spatial Filter

- 1 for Temporal Filter
- 2 for Edge detection

```
bool
selected
```

[IN] **true** when the filter mentioned by filterID is to be enabled
False when the filter mentioned by filterID is to be disabled.

Return:

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

Sample Code:

```
void SetAvgRegion_()
{
    AvgRegion region = MouseLivePtr;
    if (SetAvgRegion(region) > 0)
    {
        printf("SetAvgRegion success\r\n");
    }
    else
    {
        printf("SetAvgRegion failed\r\n");
    }
}
```

Result SetMousePos(MousePtr pos)

This function sets the average depth and IR calculation area specified by pos.

Parameters	Description
<code>MousePtr pos</code>	[IN] Object of the structure <i>MousePtr</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 SetMousePos_()
{
    MousePtr pos;
```

```

pos.X = pos.Y = 100;
if (SetMousePos (pos)>0)
{
    printf("SetMousePos success\r\n");
}
else
{
    printf("SetMousePos failed\r\n");
}
}

```

Result SetCursorColor(int color)

This function sets the cursor color to the white or black specified by color parameter.

Parameters	Description
<code>int color</code>	[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_()
{
    if (SetCursorColor (0)>0)
        printf("Cursor color set to white\r\n");
    else
        printf("SetMousePos failed\r\n");
}

```

Result SetPlanarization(int planarize)

This function enables or disables planarization based on the planarize parameter.

Parameters	Description
<code>int planarize</code>	[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_()
{
    if(SetPlanarization(0)>0)
    {
        printf("Planarization is disbaled\r\n");
    }
    else
    {
        printf("SetPlanarization failed\r\n");
    }
}
```

Result GetDepthIRValues(int *gAvgDepth, int *gStdDepth, int *gAvgIR, int *gStdIR)

This function gets the average depth and IR and sigma depth and IR.

Parameters	Description
int *gAvgDepth	[OUT] pointer to int variable that is to be filled with the average depth value.
int *gStdDepth	[OUT] pointer to int variable that is to be filled with the sigma depth value.
int *gAvgIR	[OUT] pointer to int variable that is to be filled with the average IR value.
int *gStdIR	[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 gAvgDepth, gStdDepth, gAvgIR, gStdIR;
    if (GetDepthIRValues(&gAvgDepth, &gStdDepth, &gAvgIR,
        &gStdIR) > 0)
    {
        printf("Average depth value is %d\r\n",
            gAvgDepth);
        printf("sigma depth value is %d\r\n", gStdDepth);
        printf("Average IR value is %d\r\n", gAvgIR);
        printf("sigma IR value is %d\r\n", gStdIR);
    }
    else
    {
        printf("GetDepthIRValues failed\r\n");
    }
}
```

void RegisterFrameCallback(function<void(int)> cb)

Register a callback function specified by cb when frame is received from the device.

Parameters	Description
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_()
{
    RegisterFrameCallback(&callback_func);
}
```

void RegisterNotificationCallback(function<void(int)> cb)

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

Parameters	Description
<code>function<void(int)> cb</code>	[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_()
{
    RegisterNotificationCallback(&callback_func);
}
```

Result UpdateAvgXandY(int avg_x, int avg_y)

This function updates the ROI for which the average depth and IR is being calculated.

Parameters	Description
<code>int avg_x</code>	[IN] width of the average depth calculating ROI
<code>int avg_y</code>	[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;
    if (UpdateAvgXandY(avg_x, avg_y) > 0)
    {
        printf("UpdateAvgXandY success\r\n");
    }
    else
    {
        printf("UpdateAvgXandY failed\r\n");
    }
}
```

Result UpdateColorMap(int min, int max, int colormap)

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

Parameters	Description
<code>int min</code>	[IN] minimum limit of depth data that is to be included in the colormap image.
<code>int max</code>	[IN] maximum limit of depth data that is to be included in the colormap image.
<code>int colormap</code>	[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;
    if(UpdateColorMap(min, max, colormap)>0)
    {
        printf("UpdateColorMap success\r\n");
    }
    else
    {
        printf("UpdateColorMap failed\r\n");
    }
}
```

Result GetIMUEmbeddedData(uint8_t* gIMUData)

Returns the IMU embedded option from the device.

Parameters	Description
<code>uint8_t* gIMUData</code>	[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"> • 0 for face detection disabled • 1 for face detection enabled

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 **TimeoutError** when the wait event timed out.
- Returns **InvalidHidHandle** when the buffer obtained is invalid.
- Returns **Others** when an unknown error occurred.

Sample Code:

```
void GetIMUEmbeddedData_()
{
    uint8_t gIMUData;
    if (GetIMUEmbeddedData (&gIMUData) > 0)
    {
        printf("GetIMUEmbeddedData success\n");
        if (gIMUData == 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(uint8_t sIMUData)

This function sets the IMU embedded option to the device.

Parameters	Description
uint8_t sIMUData	[IN] uint8_t data, that is to be set for IMU Embedded Data. <ul style="list-style-type: none"> • 0 for face detection disabled • 1 for face detection enabled

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 **TimeoutError** when the wait event timed out.
- Returns **InvalidHidHandle** when the buffer obtained is invalid.
- Returns **Others** when an unknown error occurred.

Sample Code:

```
void SetIMUEmbeddedData_()
{
    uint8_t sIMUData;
    if(SetIMUEmbeddedData(sIMUData)>0)
    {
        printf("SetIMUEmbeddedData success\r\n");
    }
    else
    {
        printf("SetIMUEmbeddedData failed\r\n");
    }
}
```


Enum

The details regarding the enum used will be discussed below.

Result

Description: Contains the error values that can 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,  
    SysCallFail = -12,  
    InvalidValue = -13,  
    HidWriteFail = -14,  
    HidReadFail = -15,  
    UVCOpenFail = -16,  
    TimeoutError = -17,  
    InvalidBuffer = -18,  
    InvalidHidHandle = -19,  
    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.

SysCallFail: 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.

InvalidHidHandle: Invalid HID Handle.

Others: An unknown error occurred.

UVCPropID

UVC controls for DepthVista is contained within the following 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;
```

FrameType

Types of frames that can be obtained from DepthVista is contained within the following enum.

```
typedef enum {
    IRPreviewFrame = 2,
    DepthColorMap = 3,
    RGBFrame = 4,
    DepthRawFrame = 7,
}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.

DataMode

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

```
typedef enum {
    ModeUnknown = 255,
    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,

ModeUnknown: This data mode is used as default in the application.

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.

AvgRegion

Contains two members 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.

Structure

The details regarding Structures used will be discussed below.

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;
}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.

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 the 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.

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[250];  
    char serialNo[50];  
}DeviceInfo;
```

Member Description:

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

MousePtr

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

Members:

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

Member Description:

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

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