NaturalPoint

OptiTrack API

Version: 1.83

For SDK Version 1.1.037

Date: 01/14/2010

NaturalPoint Corporation
33872 SE Eastgate Circle
Corvallis OR 97339

Table of Contents

1 Architecture

1.1 Overview

The OptiTrack API is written as a series of COM automation interfaces. This type of interface was chosen due to the flexibility it provides. COM automation interfaces are supported by nearly every language, including VBScript, JavaScript, Visual Basic and C/C++. Support also exists for Python, Delphi and many others. Check your favorite language for more details.

1.2 Component Model

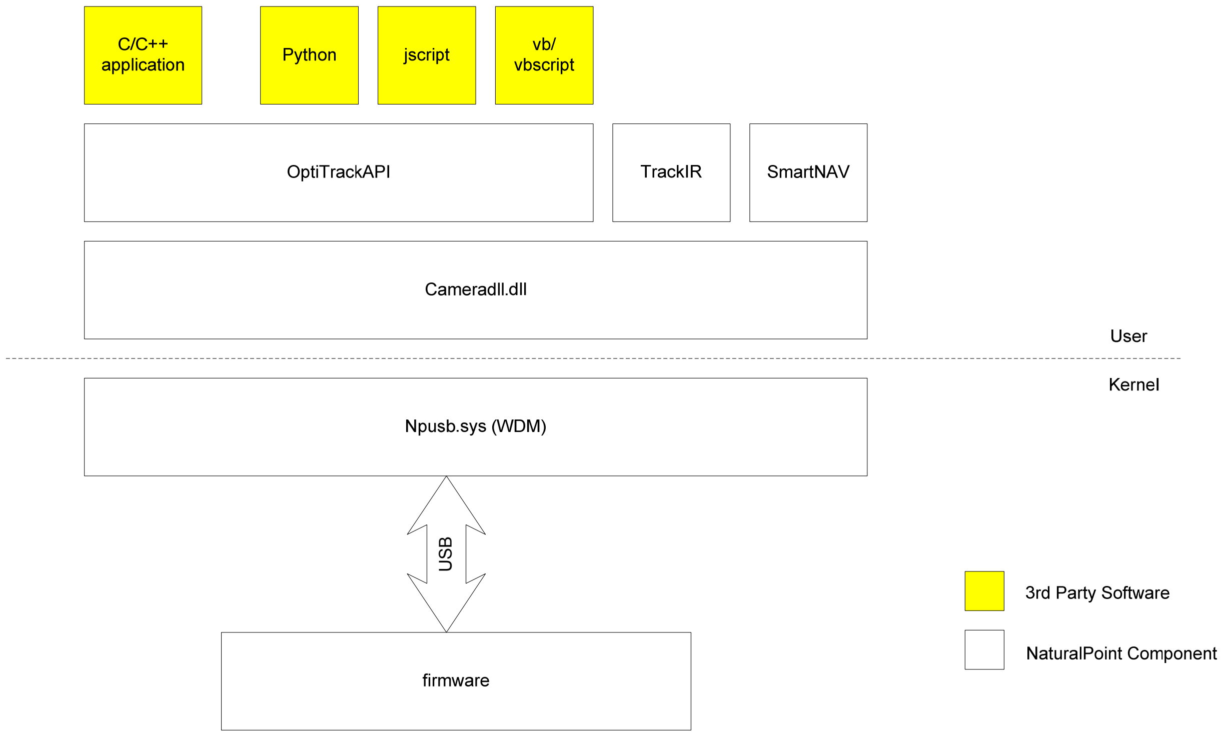

Figure 1

1.3 Interface Layout

Figure 2 below shows the interfaces in the OptiTrack API and their relationships. The main interface is INPCameraCollection. This interface contains a list of all cameras attached to the system. As cameras are added and removed to the system, notifications will be sent using the _INPCameraCollectionEvents interface.

The INPCamera interface contains most of the functionality involving the device. Information about the device can be queried, the device can be started and stopped, and data can be collected.

When a frame is captured from the device, a client will obtain an INPCameraFrame interface. This interface is simply a collection of tracked objects in the frame. The ranking of the objects in the frame can be changed by setting options on the INPCamera interface. Notifications for events involving the camera are received through the _INPCameraEvents interface.

The INPSmoothing and INPVector interfaces are provided as helpers to manipulate the frame data. The data from the camera can contain spatial noise, so some sort of smoothing needs to be applied. The INPSmoothing interface is an example of a smoothing algorithm. Apply smoothing to the single dot object positions or the vector calculations.

The INPVector interface will calculate the 6 degrees of freedom (6DOF) values when using Vector Expansion. A Vector clip is required in order for the calculations to work correctly. When calculating Vector positions, be sure to apply some sort of smoothing to the outputs. Smoothing is not required for the frame inputs to the INPVector interface.

(Note : Vector will remain in the SDK, but support will no longer be provided for it)

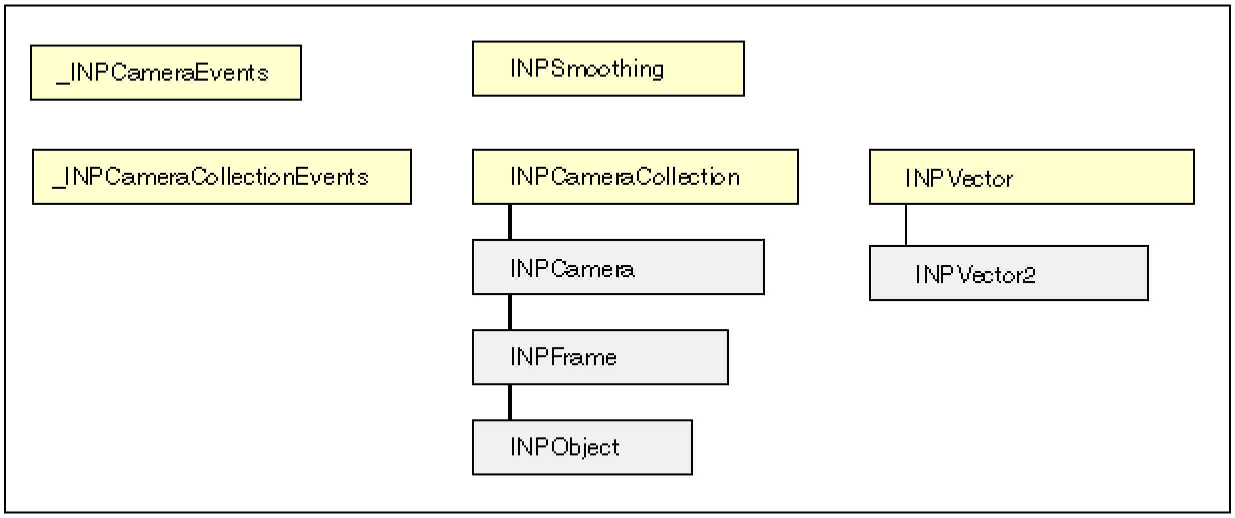

Figure 2

2 Functionality

2.1 npusb.sys

Npusb.sys is the USB driver for the NaturalPoint camera. The driver resides in kernel space and is loaded whenever a device is connected to the system. The driver handles communication with the device from the rest of the NaturalPoint components. No interaction with the device driver is required.

2.2 cameradll.dll

The camera DLL resides in user space and handles communication to the USB driver. This library contains routines to handle camera specific data. No external interfaces are provided.

2.3 optitrack.dll

This DLL is the set of COM automation interfaces provided by OptiTrack.

2.4 smartnav.exe

SmartNAV is an application written by NaturalPoint that provides cursor control for the ergonomics and assistive technologies market. It is an example of an application that can be written using the camera system. See http://www.naturalpoint.com/smartnav for more information.

2.5 trackir.exe

TrackIR is the gaming application written by NaturalPoint. TrackIR provides view control to many game titles. See http://www.naturalpoint.com/trackir for more information.

3 Design Considerations

3.1 Getting Started

Only a small amount of code needs to be written in order to communicate with the camera. The following outlines the procedure for initializing the camera:

1. Create the INPCameraCollection object.
2. Call the Enum() method to enumerate the devices in the system.
3. Walk the list of cameras in the collection.
4. If a camera is detected, initialize the camera:
    a. Call the Open() method on INPCamera.
    b. Query any information about the device—hardware type, serial number, etc. (optional)
    c. Apply any options that are relevant to your application (optional).
    d. Turn on any LEDs (optional).
    e. Call the Start() method. This will start the collection of frame information from the camera.

At this point, the camera should be initialized and collecting frame information. The main loop of the application should either handle frame callbacks using the _INPCameraEvents interface or poll for frames using the GetFrame() method of INPCamera.

When data processing is complete, call the Stop() method of INPCamera and then close the device by calling Close().

The following code shows how to communicate with the camera using callbacks in VBScript.

```
dim objNPCameras
dim objCamera

Sub SINK_DeviceArrival(objCamera)
    WScript.Echo "Device Arrival."
End Sub

Sub SINK_DeviceRemoval(objCamera)
    WScript.Echo "Device Removal."
End Sub

Sub CAMERA_FrameAvailable(objCamera)
    Set objFrame = objCamera.GetFrame(0)

    ‘ TODO: process the frame information here

    objFrame.Free()
End Sub

' create the main config object
set objNPCameras = WScript.CreateObject("OptiTrack.NPCameraCollection", "SINK_")

‘ enumerate the cameras
objNPCameras.Enum()
WScript.Echo "Num of cameras: " & objNPCameras.Count

for each objCamera in objNPCameras

    ' register callbacks
    WScript.ConnectObject objCamera, "CAMERA_"

    ' Open the camera
    objCamera.Open()

    ‘ start the camera
    objCamera.Start()

    ‘ process data until stopped
    WScript.Echo "hit OK to stop data."

    ' close the camera
    objCamera.Close()

    ' disconnect callbacks
    WScript.DisconnectObject objCamera

Next
```

3.2 Single Dot Tracking

The basic OptiTrack API provides functionality to help track a single object in the camera's view. Several factors go into determining which object is to be considered the tracked object. These options are exposed through the SetOption method of the INPCamera interface.

There are 6 variables that go into determining the tracked object. They are: object size, object ratio, proximity to other objects, static count (non-movement), distance from the center of the imager, and whether or not the current object was the last tracked object.

Object size is the number of imager pixels in the object. Ratio is the ratio of the width to the height of the object. The static count measures whether or not the object is moving. Most often, static objects are not good candidates for being tracked. Distance from the center measures how far an object is located from the center of the imager. The farther from the center of the imager, the less likely it is to be considered as the preferred tracked object. The last tracked object has more weight in the calculations based on the fact that the most recent object is probably going to be the object tracked in the future.

All tracking variables have a minimum, maximum, ideal, weight and penalty if the value is outside the minimum or maximum. Each of the 6 variables is scaled from 0 to 1.0 based on its position within the minimum and maximum range. The scaled value is multiplied by its weight and then the sum of all variables is calculated. The objects are then ranked based on their weightings. The object with the highest weighting is ranked number 1.

Changing the options will affect the ranking of the objects returned in the INPCameraFrame interface. The object with the highest ranking will be known as the tracked object. This is the object applications should use to determine movement when only a single object is preferred.

The output from the camera can contain spatial noise. When using single dot tracking, be sure to smooth the data using an appropriate algorithm. A sample smoothing algorithm is provided in the INPSmoothing interface.

The dot tracking functionality provided by the OptiTrack API is entirely optional. The algorithms provided in the API are the same set of rules that other NaturalPoint applications use. 3rd party applications may want to take the camera object information and rank the objects manually if the desired results are not obtained through the default tracking parameters.

3.3 Vector Tracking

(Vector will remain in the SDK, but support will no longer be provided for it)

Vector tracking is optional and can be used to determine the position and orientation of a users head in 3D space. The single dot tracking rankings do not have any affect on the Vector tracking algorithms. The INPVector interface will rank the objects it thinks are valid. This is done by the position and size of the objects. When using Vector tracking, it is important keep extra objects in the field of view at a minimum.

If for some reason the tracking of all three objects in the Vector clip is lost, call the Reset method of INPVector to restart the calculations.

The Vector algorithms work best when the clip is placed directly in front of the camera, about 2-3 feet away. See the TrackIR FAQ on www.naturalpoint.com for more help on ideal positions when using Vector.

3.4 Connection Points

Standard COM connection points are used for callback notifications. Connection points were chosen because they are compatible with most languages that communicate with COM automation interfaces. See the COM documentation in the MSDN help for more information. Specifically, search for IConnectionPoint and IConnectionPointContainer.

There is slightly more overhead involved with setting up connection points in C/C++, but sample code is provided to assist in coding.

3.5 Threading Issues

The OptiTrack APIs are apartment threaded. Apartment threaded means that only one thread can be used to call into the DLL at any given time. Any other threads that are used to call into the DLL while another thread is executing in the DLL will be blocked until the first thread completes.

This also affects how callbacks are implemented. Connection point callbacks must be done on the same thread that created the object. This is accomplished by creating a hidden window whenever an object that has connection points is created. The hidden window uses the message queue of the calling thread. Callbacks are done by posting a message to the hidden window and then calling the connection point callback interface.

The drawback to this mechanism is that the message queue for the thread that created the object needs to be free to run if OptiTrack notifications are to be handled in the timely fashion. For instance, if the main thread of a GUI application is used to create the OptiTrack camera object, make sure that no blocking operations are run on that thread. If a blocking operation is required, use a message loop to handle messages.

This issue only affects connection point callbacks. If camera frame information is gathered by polling this INPCamera interface, this is not a concern.

3.6 Smoothing

The INPSmoothing interface assumes a correlation between the X and Y values. It is most commonly used for smoothing and filtering the data in single dot tracking. The smoothing algorithm can also be used when smoothing a single value (for instance when smoothing yaw from the Vector calculations). In this case, use only the X value and set the Y value 0.

3.7 Camera Commands

Issuing a high volume of commands to the camera device can reduce the camera’s video throughput performance. All commands that interact with the physical camera cause an interrupt to occur on the device which slows data collection and USB transmission rates. During normal operation, the device is busy handling the capture of tracking information. Commands from the client interrupt data flow and can produce undesirable results if called too frequently.

Limit the use of LED commands when capturing data. Turning on and off an LED periodically to indicate status is fine, but flashing LEDs in not recommended. In addition, it is recommended that the LED state be tracked internally to the user’s application with camera commands only issued during state changes.

3.8 Camera Frame Object Lifetime

Camera frame objects in the OptiTrack system are a limited resource. Be sure to call the Free() method to release the INPCameraFrame interface as quickly as possible.

3.9 Object Coordinates

Object coordinates are measured from the upper left corner of the device when the device is in its normal, upright position. Figure 3 shows the default configuration.

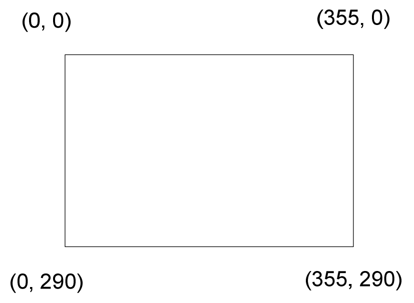

Figure 3

If transformations are applied, the dot positions will change slightly. Transformed coordinates are relative to the device center. Transformations include camera orientation and mirroring the X and Y axis. Transformations can be performed by calling the Transform method of the INPObject interface. Figure 4 shows the new configuration.

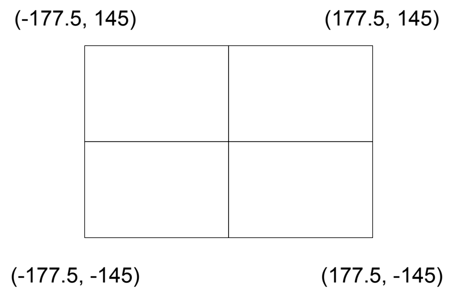

Figure 4

3.10 Color Structures

Color structures in Visual Basic are handled differently that in Win32 API. The RGB macro is used for most color specifications in the Win32 API. Visual Basic handles colors by using its own Color object. The two are not directly compatible.

The OptiTrack API requires colors to be specified using the RGB macro. Two structures are provided in the Visual Basic sample code to assist in the conversion. ARGBColor is the Visual Basic color structure and NPColor can be used to represent color structures passed to the OptiTrack API.

3.11 VARIANT_BOOL versus BOOL

One thing to note for those developers using C/C++ to create applications: VARIANT_BOOLs and BOOLs are not quite the same. -1 is defined as true for VARIANT_BOOLs. Depending on how your code is written, this may or may not affect your application.

Several macros are provided to help ease the situation. VARIANT_TRUE and VARIANT_FALSE should be used when comparing VARIANT_BOOLs. Also, the OptiTrack API provides two macros (B2VB and VB2B) for converting to and from VARIANT_BOOLs.

3.12 Switch States

The state of external ability switches connected to SmartNAV cameras is available to the software stack when each frame is captured. The SwitchState property of the INPCameraFrame interface can be queried for the state of the switches.

By default, the OptiTrack API suppresses empty camera frames to reduce the data flow in the system. Doing so also causes switch state information to be lost when no tracking objects are visible. Set the NP_OPTION_SEND_EMPTY_FRAMES to true to have all frames sent to the client.

NOTE: External switches are supported in the OptiTrack and TrackIR hardware.

4 Interfaces

4.1.1 INPCameraCollection

This is the main entry point for communicating with the cameras. This interface is a standard COM enumeration interface containing a list of all available cameras in the system.

4.1.1.1 Properties

4.1.1.1.1 INPCameraCollection::get__NewEnum

This property returns a copy of the enumerator. The returned object will be an INPCamera interface with the index pointing to the first object in the collection.

HRESULT get__NewEnum(LPUNKNOWN * ppunk);

Parameters

ppunk

[out, retval] Pointer to an IUnknown interface.

Return Values

Value	**Meaning**
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.1.1.2 INPCameraCollection::get_Count

Returns the number of objects in the collection.

HRESULT get_Count(LONG * pVal);

Parameters

pVal

[out, retval] Pointer to LONG that receives the count.

Return Values

Value	**Meaning**
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.1.2 Methods

4.1.1.2.1 INPCameraCollection::Item

The Item method returns the corresponding item in the collection. In this case Item will return an INPCamera interface.

HRESULT Item(LONG a_vlIndex, INPCamera ** ppCamera);

Parameters

A_vlIndex

[in] Index of the item to retrieve.

ppCamera

[out, retval] Pointer that receives an INPCamera interface.

Return Values

Value	Meaning
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.1.2.2 INPCameraCollection::Enum

The Enum method enumerates all cameras in the system. This method must be called before calling any other function of the INPCameraCollection interface.

HRESULT Enum();

Parameters

None

Return Values

Value	Meaning
S_OK	Method succeeded.

4.1.1.2.3 INPCameraCollection::Synchronize

The Synchronize method synchronizes the exposure timing and Frame ID of multiple cameras. The method only works with OptiTrack FLEX:C120 ~~and V100~~ cameras, inter-camera synchronization cables must be attached first in order for it to work successfully. When it is called, all cameras in the system are stopped, a master is elected and the cameras are then started.

Note : This call is deprecated for OptiTrack V100 cameras as of SDK version 1.1.033. If the synchronization cables are connected, then the Frame IDs for V100 cameras will automatically synchronize without software intervention once the cameras are started.

HRESULT Synchronize();

Parameters

None

Return Values

Value	Meaning
S_OK	Method succeeded.

4.1.2 INPCamera

4.1.2.1 Properties

4.1.2.1.1 INPCamera::get_SerialNumber

Read-only. Serial number of the camera.

HRESULT get_SerialNumber(LONG * pVal);

Parameters

pVal

[out, retval] Pointer to a LONG that receives the serial number of the camera.

Return Values

Value	Meaning
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.2.1.2 INPCamera::get_Model

Read-only. Model of the camera. Valid values are:

Name	Description
NP_HW_MODEL_OLDTRACKIR	Old TrackIR 1.
NP_HW_MODEL_SMARTNAV	SmartNAV
NP_HW_MODEL_TRACKIR	TrackIR2 or TrackIR3.
NP_HW_MODEL_OPTITRACK	OptiTrack.
NP_HW_MODEL_UNKNOWN	Unknown hardware

HRESULT get_Model(LONG * pVal);

Parameters

pVal

[out, retval] Pointer to a LONG that receives the model.

Return Values

Value	Meaning
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.2.1.3 INPCamera::get_Revision

Read-only. Hardware revision of the camera. Revisions vary depending on the model of the camera. The following table describes valid revision values:

Name	Description
NP_HW_REVISION_OLDTRACKIR_LEGACY	Old TrackIR / SmartNAV
NP_HW_REVISION_OLDTRACKIR_BASIC	Old TrackIR / SmartNAV
NP_HW_REVISION_OLDTRACKIR_EG	Old TrackIR / SmartNAV
NP_HW_REVISION_OLDTRACKIR_AT	Old TrackIR / SmartNAV
NP_HW_REVISION_OLDTRACKIR_GX	Old TrackIR / SmartNAV
NP_HW_REVISION_OLDTRACKIR_MAC	Old TrackIR / SmartNAV
NP_HW_REVISION_SMARTNAV_BASIC	SmartNAV
NP_HW_REVISION_SMARTNAV_EG	SmartNAV
NP_HW_REVISION_SMARTNAV_AT	SmartNAV
NP_HW_REVISION_SMARTNAV_MAC_BASIC	SmartNAV
NP_HW_REVISION_SMARTNAV_MAC_AT	SmartNAV
NP_HW_REVISION_TRACKIR_BASIC	TrackIR
NP_HW_REVISION_TRACKIR_PRO	TrackIR
NP_HW_REVISION_OPTITRACK_BASIC	OptiTrack Basic
NP_HW_REVISION_OPTITRACK_FLEX	OptiTrack Flex
NP_HW_REVISION_UNKNOWN	Unknown

HRESULT get_Revision(LONG * pVal);

Parameters

pVal

[out, retval] Pointer to a LONG that receives the revision.

Return Values

Value	Meaning
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.2.1.4 INPCamera::get_Width

Returns the width of the imager in pixels.

HRESULT get_Width(LONG * pVal);

Parameters

pVal

[out, retval] Pointer to a LONG that receives the imager width.

Return Values

Value	Meaning
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.2.1.5 INPCamera::get_Height

Returns the height of the imager in pixels.

HRESULT get_Height(LONG * pVal);

Parameters

pVal

[out, retval] Pointer to a long that receives the height.

Return Values

Value	Meaning
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.2.1.6 INPCamera::get_FrameRate

Returns the frame rate of the imager in frames per second.

HRESULT get_FrameRate(LONG * pVal);

Parameters

pVal

[out, retval] Pointer to a long that receives the Frame Rate.

Return Values

Value	Meaning
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.2.2 Methods

4.1.2.2.1 INPCamera::Open

Opens a connection with the device. This method must be called before making any other calls to the INPCamera interface.

HRESULT Open();

Parameters

None

Return Values

Value	**Meaning**
S_OK	Method succeeded.
NP_E_DEVICE_DISCONNECTED	The device has removed from the system.
E_POINTER	Pointer is invalid.

4.1.2.2.2 INPCamera::Close

Closes the connection with the device.

HRESULT Close();

Parameters

None

Return Values

Value	**Meaning**
S_OK	Method succeeded.

4.1.2.2.3 INPCamera::Start

Starts video on the camera. Data frames will be available after start is called.

HRESULT Start();

Parameters

None

Return Values

Value	**Meaning**
S_OK	Method succeeded.

NP_E_DEVICE_DISCONNECTED	The device has removed from the system.
NP_E_DEVICE_NOT_SUPPORTED	The connected device is an older hardware type not supported by this API.

4.1.2.2.4 INPCamera::Stop

Stops data flow from the camera.

HRESULT Stop();

Parameters

None

Return Values

Value	**Meaning**
S_OK	Method succeeded.
NP_E_DEVICE_NOT_SUPPORTED	The connected device is an older hardware type not supported by this API.

4.1.2.2.5 INPCamera::SetLED

Sets the state of the state of the specified LED. LEDs can be turned on or off using this method.

HRESULT SetLED(LONG lLED, VARIANT_BOOL fOn);

Parameters

lLED

[in] LED to be turned on or off. The LED must be specified by the NP_LED enumeration. LEDs have different mappings depending on the hardware revision. The following table outlines the differences.

TrackIR / SmartNAV / OptiTrack Basic

LED	**Meaning**
NP_LED_ONE	Illumination LEDs
NP_LED_TWO	Green LED (status)
NP_LED_THREE	Red LED
NP_LED_FOUR	Blue LED

OptiTrack Flex

LED	**Meaning**

NP_LED_ONE	Illumination LEDs
NP_LED_TWO	Left red LED (status)
NP_LED_THREE	Right red LED
NP_LED_FOUR	Not used

fOn

[in] The desired state of the LED. VARIANT_TRUE turns on the LED, VARIANT_FALSE turns the LED off.

Return Values

Value	**Meaning**
S_OK	Method succeeded.
E_INVALIDARG	Invalid LED number.
NP_E_DEVICE_DISCONNECTED	The device has removed from the system.
NP_E_DEVICE_NOT_SUPPORTED	The connected device is an older hardware type not supported by this API.

4.1.2.2.6 INPCamera::SetVideo

Turns the stream of video from the camera On and Off, this allows the video to be paused without shutting down the camera. The camera must be started before calling this method. Calling INPCamera::Stop() forces the video stream off regardless of it's current state.

HRESULT SetVideo(VARIANT_BOOL fOn);

Parameters

fOn

[in] The desired state of the Video stream. VARIANT_TRUE turns the video stream on, VARIANT_FALSE turns the video stream off.

Return Values

Value	**Meaning**
S_OK	Method succeeded.
NP_E_NOT_STARTED	The camera was not started
NP_E_DEVICE_DISCONNECTED	The device has removed from the system.
NP_E_DEVICE_NOT_SUPPORTED	The connected device is an older hardware type not supported by this API.

4.1.2.2.7 INPCamera::GetFrame

This method returns the most recent frame from the camera.

HRESULT GetFrame(LONG lTimeout, INPCameraFrame ** ppFrame);

Parameters

lTimeout

[in] The amount of time in milliseconds to wait for a frame. If no frame has arrived from the camera, the thread will block. Specify 0 in order for the thread to not block. Specifying INFINITE will cause the thread to wait indefinitely. Threads that are blocked will be signaled when the camera is stopped.

ppFrame

[out, retval] Pointer to an INPCameraFrame interface.

Return Values

Value	**Meaning**
S_OK	Method succeeded.
E_POINTER	ppFrame is NULL.
NP_E_DEVICE_DISCONNECTED	The device has removed from the system.
NP_E_DEVICE_NOT_SUPPORTED	The connected device is an older hardware type not supported by this API.
HRESULT_FROM_WIN32(ERROR_TIMEOUT)	A timeout occurred waiting for a frame from the camera.

4.1.2.2.8 INPCamera::DrawFrame

This method draws the camera image to the specified window handle.

HRESULT DrawFrame(INPCameraFrame * pFrame, LONG hwnd);

Parameters

pFrame

[in] Specifies the camera frame to draw. Camera frames can be obtained by calling the GetFrame method of INPCamera.

hwnd

[in] Window handle of the window in which the image is drawn. The size of the image can be changed by setting the NP_OPTION_DRAW_SCALE option.

Return Values

Value	**Meaning**

S_OK	Method succeeded.
E_INVALIDARG	Invalid hwnd.
NP_E_DEVICE_DISCONNECTED	The device has removed from the system.
NP_E_DEVICE_NOT_SUPPORTED	The connected device is an older hardware type not supported by this API.

4.1.2.2.9 INPCamera::ResetTrackedObject

This method resets the single dot tracking engine. The currently tracked object will have no weighting advantage against other objects.

HRESULT ResetTrackedObject();

Parameters

None

Return Values

Value	**Meaning**
S_OK	Method succeeded.
NP_E_DEVICE_DISCONNECTED	The device has removed from the system.
NP_E_DEVICE_NOT_SUPPORTED	The connected device is an older hardware type not supported by this API.

4.1.2.2.10 INPCamera::GetOption

This method returns the value of the specified camera option.

HRESULT GetOption(LONG lOption, VARIANT * pVal);

Parameters

lOption

[in] Specifies the option to set. Options are defined by the NP_OPTION enumeration.

pV*al*

[out, retval] Returns the value of the specified option. See the table below for a mapping of option values to VARIANT types.

Option	**Type**	**Meaning**
NP_OPTION_STATUS_GREEN_ON_TRACKING	VARIANT_BOOL	By default the OptiTrack API turns the status LED on and off when an object is being

		tracked. If this behavior is not desired, set this options to VARIANT_FALSE.
NP_OPTION_TRACKED_OBJECT_COLOR	VT_I4	When the DrawFrame method is called, the currently tracked object is draw using this color. The default value is green. Use the RGB macro to specify the color.
NP_OPTION_UNTRACKED_OBJECTS_COLOR	VT_I4	When the DrawFrame method is called, objects that are not tracked are drawn using this color. The default value is red. Use the RGB macro to specify the color.
NP_OPTION_OBJECT_COLOR_OPTION	VT_I4	The object color option specifies how the various objects are drawn when the DrawFrame method is called. Specify options using the NP_OBJECT_COLOR_OPTION enumeration.
NP_OPTION_DRAW_SCALE	VT_R8	The draw scale option is a multiplier applied to the camera image when DrawFrame is called. Value values are from 0.1 to 5.0.
NP_OPTION_THRESHOLD	VT_I4	Scene light filtering can be accomplished by changing the video threshold level. Lower values for the threshold allow objects of a lower light intensity to be captured. Higher values will filter out objects of a lower light intensity. Valid values are from NP_THRESHOLD_MIN to NP_THRESHOLD_MAX. Threshold values are not persisted in the device and are write-only. When the API starts or a device is inserted, the threshold is assumed to be the device's default value.
NP_OPTION_OBJECT_MASS_WEIGHT	VT_R8	Weight of the object mass in determining the tracked object. Default value: 1.0
NP_OPTION_OBJECT_RATIO_WEIGHT	VT_R8	Weight of the object ratio in determining the tracked object. Default value: 1.0
NP_OPTION_PROXIMITY_WEIGHT	VT_R8	Weight of the object's proximity to other objects in determining the tracked object. Default value: 1.0
NP_OPTION_STATIC_COUNT_WEIGHT	VT_R8	Weight of the object's non-movement in determining the tracked object. Default value: 1.0
NP_OPTION_SCREEN_CENTER_WEIGHT	VT_R8	Weight of the object's proximity to the center of the imager in determining the tracked object. Default value: 1.0

NP_OPTION_LAST_OBJECT_TRACKED_WEIGHT	VT_R8	Weight of the last tracked object in determining the tracked object. Default value: 2.0
NP_OPTION_OBJECT_MASS_MIN	VT_R8	Minimum number of pixels an object should have to be considered a tracked object. Default value: 3
NP_OPTION_OBJECT_MASS_MAX	VT_R8	Maximum number of pixels an object can have and still be considered a tracked object. Default value: 200
NP_OPTION_OBJECT_MASS_IDEAL	VT_R8	Ideal number of pixels an object should have to be considered the tracked object. Default value: 100
NP_OPTION_OBJECT_MASS_OUT_OF_RANGE	VT_R8	Score to assign the object mass if the value is outside the range of MIN and MAX. Default value: 0
NP_OPTION_OBJECT_RATIO_MIN	VT_R8	Minimum object ratio to be considered a tracked object. Ratio is Width / Height. Default value: 0.25
NP_OPTION_OBJECT_RATIO_MAX	VT_R8	Maximum object ratio to be considered a tracked object. . Ratio is Width / Height. Default value: 4.0
NP_OPTION_OBJECT_RATIO_IDEAL	VT_R8	Ideal object ratio to be considered a tracked object. . Ratio is Width / Height. Default value: 1.0
NP_OPTION_OBJECT_RATIO_OUT_OF_RANGE	VT_R8	Score to assign the ratio if the value is outside the range of MIN and MAX. Default value: 0
NP_OPTION_PROXIMITY_MIN	VT_R8	Minimum distance from other objects to be considered the tracked object. Default value: 3
NP_OPTION_PROXIMITY_MAX	VT_R8	Maximum distance from other objects to be considered the tracked object. Default value: 300
NP_OPTION_PROXIMITY_IDEAL	VT_R8	Ideal distance from other objects to be considered the tracked object. Default value: 20
NP_OPTION_PROXIMITY_OUT_OF_RANGE	VT_R8	Score to assign the proximity if the value is outside the range of MIN and MAX. Default value: 0
NP_OPTION_STATIC_COUNT_MIN	VT_R8	Minimum number of frames the object doesn’t move to be considered the tracked object. Default value: 0

NP_OPTION_STATIC_COUNT_MAX	VT_R8	Maximum number of frames the object doesn't move to not be considered the tracked object. Default value: 200
NP_OPTION_STATIC_COUNT_IDEAL	VT_R8	Ideal number of frames the object doesn't move to be considered the tracked object. Default value: 0
NP_OPTION_STATIC_COUNT_OUT_OF_RANGE	VT_R8	Score to assign the static count if the value is outside the range of MIN and MAX. Default value: 0
NP_OPTION_SCREEN_CENTER_MIN	VT_R8	Minimum distance from center of imager to be considered the tracked object. Default value: 0
NP_OPTION_SCREEN_CENTER_MAX	VT_R8	Maximum distance from center of imager to be considered the tracked object. Default value: 256
NP_OPTION_SCREEN_CENTER_IDEAL	VT_R8	Ideal distance from center of imager to be considered the tracked object. Default value: 0
NP_OPTION_SCREEN_CENTER_OUT_OF_RANGE	VT_R8	Score to assign the screen center if the value is outside the range of MIN and MAX. Default value: 0
NP_OPTION_LAST_OBJECT_MIN	VT_R8	Minimum number of frames to consider an object the last tracked object. Default value: 0
NP_OPTION_LAST_OBJECT_MAX	VT_R8	Maximum number of frames to consider an object the last tracked object. Default value: 20
NP_OPTION_LAST_OBJECT_IDEAL	VT_R8	Ideal number of frames to consider an object the last tracked object. Default value: 0
NP_OPTION_LAST_OBJECT_OUT_OF_RANGE	VT_R8	Score to assign the last object if the value is outside the range of MIN and MAX. Default value: 0
NP_OPTION_STATUS_LED_ON_START	VARIANT_BOOL	By default, the OptiTrack API will turn on the status LED when the Start method is called. If this behavior is not desired, set this option to VARIANT_FALSE before calling the Start method.
NP_OPTION_ILLUMINATION_LEDS_ON_START	VARIANT_BOOL	By default, the OptiTrack API will turn on the illumination LED when the Start method is called. If this behavior is not desired, set this option to VARIANT_FALSE before calling the Start method.
NP_OPTION_CAMERA_ROTATION	VT_I4	The camera may be mounted in an

		orientation other than the normal, upright position. If the camera is turned on its side or upside down, set this option. Use the NP_CAMERA_ROTATION enumeration to specify the camera rotation. To apply these settings, the INPObject::Transform method must be called per object which you wish to transform.
NP_OPTION_MIRROR_X	VARIANT_BOOL	At times, it is desirable to reverse the motion in the X plane. Set this option to VARIANT_TRUE to mirror movement in the X plane. To apply these settings, the INPObject::Transform method must be called per object which you wish to transform.
NP_OPTION_MIRROR_Y	VARIANT_BOOL	At times, it is desirable to reverse the motion in the Y plane. Set this option to VARIANT_TRUE to mirror movement in the Y plane. To apply these settings, the INPObject::Transform method must be called per object which you wish to transform.
NP_OPTION_SEND_EMPTY_FRAMES	VARIANT_BOOL	By default, the OptiTrack API will filter camera frames that have no data in them. To have the API send all camera frames, set this option to VARIANT_TRUE.
NP_OPTION_CAMERA_ID	VT_I4	Assigns an ID number to a camera, the camera will stamp this ID onto the frames that it sends to the PC. This feature is only supported by the OptiTrack FLEX:C120 and V100 cameras.
NP_OPTION_FRAME_RATE	VT_I4	Adjusts the cameras frame rate. Not all percentage values are available, the closest matching value will automatically be used when an unavailable value is chosen. Using a smaller percentage will slow down the frame rate, this causes the camera to expose longer resulting in brighter images. Range is from 3 to 100 (percent of the maximum frame rate). Default value is 100. Valid settings for C120 cameras : 3, 6, 10, 13, 16, 20, 33, 40,50, 66, 100 Valid settings for V100 cameras : 25, 50, 100 This feature is only supported by the OptiTrack FLEX:C120 and V100 cameras.
NP_OPTION_EXPOSURE	VT_I4	Adjusts the cameras electronic exposure control, allowing for brighter or darker images.

		C120 cameras : Range is from 0 to 399 (smaller values result in darker images). Default value is 150. V100 cameras : Range is from 0 to 479 (smaller values result in darker images). Default value is 55. This feature is only supported by the OptiTrack FLEX:C120 and V100 cameras.
NP_OPTION_VIDEO_TYPE	VT_I4	Selects the type of video data sent by the camera. Values : 0 = processed video with final objects built on the PC, 1 = raw greyscale video, 2 = processed video with final objects built in the camera (V100 cameras only). Default value is 0 (Processed with objects built on the PC) This feature is only supported by the OptiTrack FLEX:C120 and V100 cameras.
NP_OPTION_INTENSITY	VT_I4	Controls the intensity (brightness) of the IR illumination LEDs on the camera. Range is from 1 to 12 (smaller values result in less emitted IR illumination). Default value is 12 (maximum illumination power). V100 cameras only : Strobe illumination mode is enabled by using a value of 15. This mode activates the IR LEDs at full power during a brief period at the start of the frame. For best results a short exposure time (less than 100) is recommended when using strobe mode. This feature is only supported by the OptiTrack FLEX:C120 and V100 cameras.
NP_OPTION_FRAME_DECIMATION	VT_I4	Controls the cameras auto-frame discarding (decimation) feature, this allows the camera to expose at fast shutter speeds while delivering a reduced number of frames (every Nth) to the PC. Range is from 0 to 5 (larger values result in more auto-discarded frames). Default value is 0 (no frames discarded). 0=drop no frames, 1=send every other, 2=send every forth, 3=send every eighth, 4=send every sixteenth, 4=send every thirty-secondth This feature is only supported by the OptiTrack FLEX:C120 and V100 cameras.
NP_OPTION_MINIMUM_SEGMENT_LENGTH	VT_I4	Sets the minimum horizontal segment width for a thresholded slice of an object that the

		camera will accept, useful for filtering out small image noise. Segments which are shorter than the minimum length will get discarded. Range is from 0 to 1023 (in ½ pixel increments). Default value is 0 (allow all detected pixels through). This feature is only supported by the OptiTrack FLEX:C120 and V100 cameras.
NP_OPTION_MAXIMUM_SEGMENT_LENGTH	VT_I4	Sets the maximum horizontal segment width for a thresholded slice of an object that the camera will accept, useful for filtering out large image noise. Segments which are larger than the maximum length will get discarded. Range is from 0 to 1023 (in ½ pixel increments). Default value is 1023 (allow all detected pixels through). This feature is only supported by the OptiTrack FLEX:C120 and V100 cameras.
NP_OPTION_WINDOW_EXTENTS_X	VT_I4	Sets the left edge of the clipping window for the cameras image. Range is from 74 to 427. Default value is 74. This feature is only supported by the OptiTrack FLEX:C120 and V100 cameras.
NP_OPTION_WINDOW_EXTENTS_X_END	VT_I4	Sets the right edge of the clipping window for the cameras image. Range is from 74 to 427. Default value is 427. This feature is only supported by the OptiTrack FLEX:C120 and V100 cameras.
NP_OPTION_WINDOW_EXTENTS_Y	VT_I4	Sets the top edge of the clipping window for the cameras image. Range is from 11 to 299. Default value is 11. The resulting total vertical height of the image must a multiple of 4 in order for video to work. This feature is only supported by the OptiTrack FLEX:C120 and V100 cameras.
NP_OPTION_WINDOW_EXTENTS_Y_END	VT_I4	Sets the bottom edge of the clipping window for the cameras image. Range is from 11 to 299. Default value is 299. This feature is only supported by the OptiTrack FLEX:C120 and V100 cameras.
NP_OPTION_RESET_FRAME_COUNT	N/A	Resets the frame ID counter (incremented once per frame) in the camera to 0. Write only. Also see INPCameraFrame::get_Id() This feature is only supported by the

		OptiTrack FLEX:C120 and V100 cameras.
NP_OPTION_NUMERIC_DISPLAY_ON	VT_I4	Turns on the Cameras Numeric LED and displays a number. Range 0-99. This feature is only supported by the OptiTrack FLEX:C120 and V100 cameras.
NP_OPTION_NUMERIC_DISPLAY_OFF	N/A	Turns off the Cameras Numeric LED. This feature is only supported by the OptiTrack FLEX:C120 and V100 cameras.
NP_OPTION_SEND_FRAME_MASK	VT_I4	Extended frame delivery options, this controls whether Invalid (corrupt) and/or Empty (no objects detected) frames trigger callback notification. Default value is 0. The value passed is a bitwise set of flags, the following flags may be used : ● Enable callbacks for Invalid (corrupt) frames (note : frames will not be processed) : NP_FRAME_SENDINVALID (0x02) ● Enable callbacks for Empty (no objects detected) frames : NP_FRAME_SENDEMPTY (0x01)
NP_OPTION_TEXT_OVERLAY_OPTION	VT_I4	Controls whether useful information about the current frame will be overlaid on the frame image when it gets rendered. Default value is 0. The value passed is a bitwise set of flags, the following flags may be used ● Enable overlay information about the video mode and frame ID : NP_TEXT_OVERLAY_HEADER (0x01) ● Enable overlay information about the objects detected in the frame : NP_TEXT_OVERLAY_OBJECT (0x02) ● Enable overlay information which increases the visibility of objects detected in the frame : NP_TEXT_OVERLAY_OBJECT_HIGHLIGHT (0x04)
NP_OPTION_SCORING_ENABLED	VARIANT_BOOL	Default is TRUE Allows the built-in object scoring and ranking algorithms to be disabled. Disabling scoring can significantly reduce the system processing load when the camera is tracking

		large numbers of markers. To have the API disable scoring, set this option to VARIANT_FALSE.
NP_OPTION_GRAYSCALE_DECIMATION	VT_I4	Controls the cameras grayscale image down-sampling (resizing) feature. This feature can be enabled when there is not enough USB bandwidth to transfer the entire grayscale frame. Default value is 0 (no down-sampling 640x480). 4=1/4 size (160x120), 2=1/2 size(320x240) This feature is only supported by the OptiTrack V100/V120 based cameras.
NP_OPTION_OBJECT_CAP	VT_I4	Controls the maximum number of objects detected in a frame. Lowering the number of objects returned in can be used to increase performance. Default is 500
NP_OPTION_SET_IR_FITER	VARIANT_BOOL	Controls the on-camera Filter Switcher to select between the IR light pass filter and the Visible light pass filter. A value of TRUE selects the IR light pass filter, and value of FALSE selects the Visible light pass filter. This feature is only supported for cameras with the FilterSwitcher (FS) option installed.

Return Values

Value	**Meaning**
S_OK	Method succeeded.
E_INVALIDARG	Invalid hwnd.
NP_E_DEVICE_DISCONNECTED	The device has removed from the system.
NP_E_DEVICE_NOT_SUPPORTED	The connected device is an older hardware type not supported by this API.

4.1.2.2.11 INPCamera::SetOption

This method sets the value of the specified camera option. For a list of options, see the GetOption method.

HRESULT SetOption(LONG lOption, VARIANT Val);

Parameters

lOption

[in] Specifies the option to set. Options are defined by the NP_OPTION enumeration.

V*al*

[in] VARIANT that contains the value of the option. The variant type varies depending on the option.

Return Values

Value	**Meaning**
S_OK	Method succeeded.
E_INVALIDARG	Invalid option value.
NP_E_DEVICE_DISCONNECTED	The device has removed from the system.
NP_E_DEVICE_NOT_SUPPORTED	The connected device is an older hardware type not supported by this API.

4.1.2.2.12 INPCamera::GetFrameImage

This method renders an image of the specified frame into a user-provided buffer.

HRESULT GetFrameImage(INPCameraFrame *pFrame, INT PixelWidth, INT PixelHeight, INT ByteSpan, INT BitsPerPixel, BYTE* Buffer);

Parameters

pFrame

[in] Specifies the camera frame to draw. Camera frames can be obtained by calling the GetFrame method of INPCamera.

PixelWidth

[in] Specifies the width in pixels of the image to be drawn.

PixelHeight

[in] Specifies the height in pixels of the image to be drawn.

ByteSpan

[in] Specifies the width of a horizontal line of the image in bytes. (If this value is set to zero, it will be auto-calculated based on the BitsPerPixel value).

BitsPerPixel

[in] Specifies the colordepth of the image in bits per pixel.

Value	**Meaning**
8	Eight Bits Per Pixel. Display using each 8-bit value as luminosity.

16	Sixteen Bits Per Pixel. Format: RGB565
24	Twenty Four Bits Per Pixel. Format: RGB
32	Thirty Two Bits Per Pixel. Format: RGBA

Buffer

[in] Specifies the buffer into which the frame should be drawn. In order to prevent buffer overruns and memory corruption, the buffer should be of size (PixelHeight x ByteSpan) or (PixelWidth x PixelHeight x (BitsPerPixel / 8))

Return Values

Value	**Meaning**
S_OK	Method succeeded.

4.1.3 INPVector

(Vector will remain in the SDK, but support will no longer be provided for it)

4.1.3.1 Properties

4.1.3.1.1 INPVector::get_Yaw

Returns the amount of yaw calculated from the previous frame.

HRESULT get_Yaw(VARIANT * pVal);

Parameters

pVal

[out, retval] Pointer to VARIANT that receives the yaw.

Return Values

Value	**Meaning**
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.3.1.2 INPVector::get_Pitch

Returns the amount of pitch calculated from the previous frame.

HRESULT get_Pitch(VARIANT * pVal);

Parameters

pVal

[out, retval] Pointer to VARIANT that receives the pitch.

Return Values

Value	**Meaning**
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.3.1.3 INPVector::get_Roll

Returns the amount of roll calculated from the previous frame.

HRESULT get_Roll(VARIANT * pVal);

Parameters

pVal

[out, retval] Pointer to VARIANT that receives the roll.

Return Values

Value	**Meaning**
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.3.1.4 INPVector::get_X

Returns the X position calculated from the previous frame.

HRESULT get_Yaw(VARIANT * pVal);

Parameters

pVal

[out, retval] Pointer to VARIANT that receives the X position.

Return Values

Value	**Meaning**
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.3.1.5 INPVector::get_Y

Returns the Y position calculated from the previous frame.

HRESULT get_Yaw(VARIANT * pVal);

Parameters

pVal

[out, retval] Pointer to VARIANT that receives the y position.

Return Values

Value	Meaning
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.3.1.6 INPVector::get_Z

Returns the Z position calculated from the previous frame.

HRESULT get_Z(VARIANT * pVal);

Parameters

pVal

[out, retval] Pointer to VARIANT that receives the Z position.

Return Values

Value	Meaning
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.3.2 Methods

4.1.3.2.1 INPVector::Update

Call the update method to calculate the current vector positions based on the given camera frame. If the call to Update is successful, the properties of the INPVector interface will contain the new values.

HRESULT Update(INPCamera * pCamera, INPCameraFrame * pFrame);

Parameters

pCamera

[in] Pointer to the camera that the frame originated.

pFrame

[in] Pointer to the frame used to calculate the values.

Return Values

Value	Meaning
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.3.2.2 INPVector::Reset

The reset method resets the Vector calculations to their initial state.

HRESULT Reset();

Parameters

None

Return Values

Value	**Meaning**
S_OK	Method succeeded.

4.1.4 INPVector2

(Vector will remain in the SDK, but support will no longer be provided for it)

The INPVector2 interface builds upon the functionality provided in INPVector. INPVector2 belongs to INPVector, all of the methods from INPVector are supported in addition to the new functionality.

INPVector2 provides the ability to change the configuration of the Vector clip. The custom vector clip must be a triangle with two points lower than the third point. The upper point should also be offset from the two lower points. This interface allows for larger or smaller triangles to be created, not custom shapes. There are 4 values of interest when changing the configuration. They are dist01, dist02, dist12 and distol. The following figure shows the layout of the Vector clip.

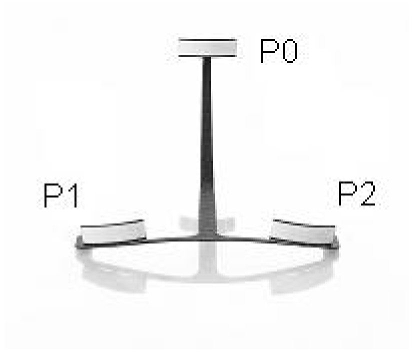

Figure 5 - Vector Layout

dist01 – The distance in mm from the center of P0 to P1

dist02 – The distance in mm from the center of P0 to P1

dist12 – The distance in mm from the center of P1 to P2

distol – The distance of the perpendicular of P1 to P2 to P0 in mm measured when looking down at the clip

For optimal performance, dist01 and dist02 should be equal.

The following figure shows another view of the measurements.

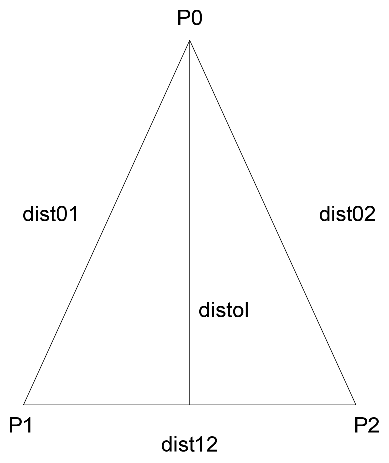

Figure 6 - Top down view of Vector clip

4.1.4.1 Properties

4.1.4.1.1 INPVector2::get_dist01

Returns the distance in mm between P0 and P1 of the vector clip.

HRESULT get_dist01(VARIANT * pVal);

Parameters

pVal

[out, retval] Pointer to VARIANT that receives the value.

Return Values

Value	**Meaning**
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.4.1.2 INPVector2::put_dist01

Sets distance in mm between P0 and P1 of the vector clip.

HRESULT put_dist01(VARIANT Val);

Parameters

Val

[in] VARIANT that specifies the distance. Variant should be of type VT_R8.

Return Values

Value	**Meaning**
S_OK	Method succeeded.

E_INVALIDARG	Invalid value.

4.1.4.1.3 INPVector2::get_dist02

Returns the distance in mm between P0 and P2 of the vector clip.

HRESULT get_dist02(VARIANT * pVal);

Parameters

pVal

[out, retval] Pointer to VARIANT that receives the value.

Return Values

Value	**Meaning**
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.4.1.4 INPVector2::put_dist02

Sets distance in mm between P0 and P2 of the vector clip.

HRESULT put_dist02(VARIANT Val);

Parameters

Val

[in] VARIANT that specifies the distance. Variant should be of type VT_R8.

Return Values

Value	**Meaning**
S_OK	Method succeeded.
E_INVALIDARG	Invalid value.

4.1.4.1.5 INPVector2::get_dist12

Returns the distance in mm between P1 and P2 of the vector clip.

HRESULT get_dist12(VARIANT * pVal);

Parameters

pVal

[out, retval] Pointer to VARIANT that receives the value.

Return Values

Value	**Meaning**
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.4.1.6 INPVector2::put_dist12

Sets distance in mm between P1 and P2 of the vector clip.

HRESULT put_dist12(VARIANT Val);

Parameters

Val

[in] VARIANT that specifies the distance. Variant should be of type VT_R8.

Return Values

Value	**Meaning**
S_OK	Method succeeded.
E_INVALIDARG	Invalid value.

4.1.4.1.7 INPVector2::get_distol

Returns the distance in mm of the perpendicular to P1 and P2 from P0 of the vector clip.

HRESULT get_distol(VARIANT * pVal);

Parameters

pVal

[out, retval] Pointer to VARIANT that receives the value.

Return Values

Value	**Meaning**
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

~~4.1.4.1.8 INPVector2::put_distol~~

<u>Note : This call is has been deprecated. distol gets calculated automatically using the other distance parameters.</u>

Sets the distance in mm of the perpendicular to P1 and P2 from P0 of the vector clip.

HRESULT put_distol(VARIANT Val);

Parameters

Val

[in] VARIANT that specifies the distance. Variant should be of type VT_R8.

Return Values

Value	**Meaning**
S_OK	Method succeeded.
E_INVALIDARG	Invalid value.

4.1.4.1.9 INPVector2::get_Tracking

Returns the status of the last update. If true, the last frame given to the vector engine contained three valid points.

HRESULT get_Tracking(VARIANT * pVal);

Parameters

pVal

[out, retval] Pointer to VARIANT that receives the value.

Return Values

Value	**Meaning**
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.4.2 Methods

4.1.4.2.1 INPVector2::GetPoint

GetPoint returns the 3D location of one of the points of the vector clip. Call this method after calling the Update method with a camera frame. Check the tracking status before calling this method to ensure the data is valid.

HRESULT GetPoint(int nPoint, INPPoint ** ppPoint);

Parameters

nPoint

[in] The index of the point to query. Valid values are from 0 to 2.

ppPoint

[out, retval] Pointer to the INPPoint interface that receives the positional information.

Return Values

Value	**Meaning**
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.5 INPVector3

(Vector will remain in the SDK, but support will no longer be provided for it)

The INPVector3 provides additional functionality beyond the INPVector2 interface. INPVector3 belongs to INPVector, all of the methods from INPVector and INPVector2 are supported in addition to the new functionality.

INPVector3 provides the ability to change characteristics of the camera used in the Vector calculation, most importantly the focal length which is useful with non-stock lenses.

4.1.5.1.1 INPVector3:: get_imagerPixelWidth

Returns the width of the imager in pixels.

HRESULT get_imagerPixelWidth (VARIANT * pVal);

Parameters

pVal

[out, retval] Pointer to VARIANT that receives the value of type VT_R8.

Return Values

Value	**Meaning**
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.5.1.2 INPVector3:: put_imagerPixelWidth

Sets the width of the imager in pixels.

HRESULT put_imagerPixelWidth (VARIANT Val);

Parameters

Val

[in] VARIANT that specifies the distance. Variant should be of type VT_R8.

Return Values

Value	**Meaning**
S_OK	Method succeeded.
E_INVALIDARG	Invalid value.

4.1.5.1.3 INPVector3:: get_imagerPixelHeight

Returns the height of the imager in pixels.

HRESULT get_imagerPixelHeight (VARIANT * pVal);

Parameters

pVal

[out, retval] Pointer to VARIANT that receives the value of type VT_R8.

Return Values

Value	**Meaning**
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.5.1.4 INPVector3:: put_imagerPixelHeight

Sets the height of the imager in pixels.

HRESULT put_imagerPixelHeight (VARIANT Val);

Parameters

Val

[in] VARIANT that specifies the distance. Variant should be of type VT_R8.

Return Values

Value	**Meaning**
S_OK	Method succeeded.
E_INVALIDARG	Invalid value.

4.1.5.1.5 INPVector3:: get_imagerMMWidth

Returns the width of the imager in millimeters.

HRESULT get_imagerMMWidth (VARIANT * pVal);

Parameters

pVal

[out, retval] Pointer to VARIANT that receives the value of type VT_R8.

Return Values

Value	**Meaning**
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.5.1.6 INPVector3:: put_imagerMMWidth

Sets the width of the imager in millimeters.

HRESULT put_imagerMMWidth (VARIANT Val);

Parameters

Val

[in] VARIANT that specifies the distance. Variant should be of type VT_R8.

Return Values

Value	**Meaning**
S_OK	Method succeeded.
E_INVALIDARG	Invalid value.

4.1.5.1.7 INPVector3:: get_imagerMMHeight

Returns the height of the imager in millimeters.

HRESULT get_imagerMMHeight (VARIANT * pVal);

Parameters

pVal

[out, retval] Pointer to VARIANT that receives the value of type VT_R8.

Return Values

Value	**Meaning**
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.5.1.8 INPVector3:: put_imagerMMHeight

Sets the height of the imager in millimeters.

HRESULT put_imagerMMHeight (VARIANT Val);

Parameters

Val

[in] VARIANT that specifies the distance. Variant should be of type VT_R8.

Return Values

Value	**Meaning**
S_OK	Method succeeded.
E_INVALIDARG	Invalid value.

4.1.5.1.9 INPVector3:: get_imagerMMFocalLength

Returns the focal length of the lens used with the imager in millimeters.

TrackIR3/SmartNAV3/OptiTrack FLEX:3 stock lens = 3.7mm

TrackIR4 stock lens = 2.45mm

OptiTrack FLEX:C120 stock lens= 2.6mm

HRESULT get_imagerMMFocalLength (VARIANT * pVal);

Parameters

pVal

[out, retval] Pointer to VARIANT that receives the value of type VT_R8.

Return Values

Value	**Meaning**
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.5.1.10 INPVector3:: put_imagerMMFocalLength

Sets the focal length of the lens used with the imager in millimeters.

HRESULT put_imagerMMFocalLength (VARIANT Val);

Parameters

Val

[in] VARIANT that specifies the distance. Variant should be of type VT_R8.

Return Values

Value	**Meaning**
S_OK	Method succeeded.
E_INVALIDARG	Invalid value.

4.1.6 INPPoint

The INPPoint interface is a wrapper for a 3D location of a point. It is used by any interface that needs to return the position of a 3D point.

4.1.6.1 Properties

4.1.6.1.1 INPPoint::get_X

Returns the X axis position of the point.

HRESULT get_X(VARIANT * pVal);

Parameters

pVal

[out, retval] Pointer to VARIANT that receives the position. Returned VARIANT will be of type VT_R8.

Return Values

Value	**Meaning**
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.6.1.2 INPPoint::get_Y

Returns the Y axis position of the point.

HRESULT get_Y(VARIANT * pVal);

Parameters

pVal

[out, retval] Pointer to VARIANT that receives the position. Returned VARIANT will be of type VT_R8.

Return Values

Value	**Meaning**
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.6.1.3 INPPoint::get_Z

Returns the Z axis position of the point.

HRESULT get_Z(VARIANT * pVal);

Parameters

pVal

[out, retval] Pointer to VARIANT that receives the position. Returned VARIANT will be of type VT_R8.

Return Values

Value	**Meaning**
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.7 INPSmoothing

4.1.7.1 Properties

4.1.7.1.1 INPSmoothing::get_Amount

Returns the amount of smoothing applied to the data.

HRESULT get_Smoothing(VARIANT * pVal);

Parameters

pVal

[out, retval] Pointer to VARIANT that receives the smoothing amount. Returned VARIANT will be of type VT_R8.

Return Values

Value	**Meaning**
S_OK	Method succeeded.

E_POINTER	Pointer is invalid.

4.1.7.1.2 INPSmoothing::put_Amount

Sets the amount of smoothing applied to the data. Valid values are from NP_SMOOTHING_MIN to NP_SMOOTHING_MAX.

HRESULT put_Smoothing(VARIANT Val);

Parameters

Val

[in] VARIANT that specifies the smoothing amount. Variant should be of type VT_R8.

Return Values

Value	**Meaning**
S_OK	Method succeeded.
E_INVALIDARG	Invalid amount.

4.1.7.1.3 INPSmoothing::get_X

Returns the smoothed X value.

HRESULT get_X(VARIANT * pVal);

Parameters

pVal

[out, retval] Pointer to VARIANT that receives the smoothed X value. Returned VARIANT will be of type VT_R8.

Return Values

Value	**Meaning**
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.7.1.4 INPSmoothing::get_Y

Returns the smoothed Y value.

HRESULT get_Y(VARIANT * pVal);

Parameters

pVal

[out, retval] Pointer to VARIANT that receives the smoothed Y value. Returned VARIANT will be of type VT_R8.

Return Values

Value	Meaning
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.7.2 Methods

4.1.7.2.1 INPSmoothing::Update

Call the update method to calculate smoothing based on the new values. If the call to Update is successful, the properties of the INPSmoothing interface will contain the new values.

HRESULT Update(VARIANT ValX, VARIANT ValY);

Parameters

ValX

[in] VARIANT that contains the new X data. VARIANT should be of type VT_R8.

ValY

[in] VARIANT that contains the new Y data. VARIANT should be of type VT_R8.

Return Values

Value	Meaning
S_OK	Method succeeded.

4.1.7.2.2 INPSmoothing::Reset

The reset method resets the smoothing calculations to their initial state.

HRESULT Reset();

Parameters

None

Return Values

Value	Meaning
S_OK	Method succeeded.

4.1.8 INPCameraFrame

This object contains information about the current camera frame. This interface is a standard COM enumeration interface containing a list of all objects in the frame.

4.1.8.1 Properties

4.1.8.1.1 INPCameraFrame::get__NewEnum

This property returns a copy of the enumerator. The returned object will be an INPObject interface with the index pointing to the first object in the collection.

HRESULT get__NewEnum(LPUNKNOWN * ppunk);

Parameters

ppunk

[out, retval] Pointer to an IUnknown interface.

Return Values

Value	**Meaning**
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.8.1.2 INPCameraFrame::get_Count

Returns the number of objects in the collection.

HRESULT get_Count(LONG * pVal);

Parameters

pVal

[out, retval] Pointer to LONG that receives the count.

Return Values

Value	**Meaning**
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.8.1.3 INPCameraFrame::get_Id

Returns the ID number associated with this frame. This method is only supported for OptiTrack FLEX:C120 and V100 cameras.

HRESULT get_Id(LONG * pVal);

Parameters

pVal

[out, retval] Pointer to LONG that receives the ID.

Return Values

Value	**Meaning**
S_OK	Method succeeded.

4.1.8.1.4 INPCameraFrame::get_SwitchState

Returns the current state of the switches at the time the frame was captured.

HRESULT get_SwitchState(LONG * pVal);

Parameters

pVal

[out, retval] Pointer to LONG that receives the switch state. Switch states are defined by the NP_SWITCH_STATE enumeration.

Return Values

Value	**Meaning**
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.8.1.5 INPCameraFrame::get_TimeStamp

Arrival time of the camera frame into the PC. Data returned is of type DOUBLE.

HRESULT get_TimeStamp(VARIANT * pVal);

Parameters

pVal

[out, retval] Pointer to Variant that receives the timestamp.

Return Values

Value	**Meaning**
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.8.1.6 INPCameraFrame::get_TimeStampFrequency

Indicates the number of ticks per second used for the value returned by get_TimeStamp. Data returned is of type DOUBLE.

HRESULT get_TimeStampFrequency(VARIANT * pVal);

Parameters

pVal

[out, retval] Pointer to Variant that receives the timestamp.

Return Values

Value	**Meaning**
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.8.1.7 INPCameraFrame::get_IsCorrupt

This method indicates whether the specified frame arrived with corrupted data.

Also see NP_OPTION_SEND_FRAME_MASK which controls whether or not corrupted frames will trigger callbacks.

HRESULT get_IsCorrupt(VARIANT_BOOL * pVal);

Parameters

pVal

[out, retval] Pointer to LONG that receives the result. Value will be TRUE if the frame contains corrupted data.

Return Values

Value	**Meaning**
S_OK	Method succeeded.

4.1.8.1.8 INPCameraFrame::get_IsGreyscale

This method indicates whether the specified frame contains greyscale image data.

HRESULT get_IsGreyscale(VARIANT_BOOL * pVal);

Parameters

pVal

[out, retval] Pointer to LONG that receives the result. Value will be TRUE if the frame contains greyscale image data.

Return Values

Value	**Meaning**
S_OK	Method succeeded.

4.1.8.1.9 INPCameraFrame::get_IsEmpty

This method indicates whether the specified frame contains any objects.

HRESULT get_IsEmpty(VARIANT_BOOL * pVal);

Parameters

pVal

[out, retval] Pointer to LONG that receives the result. Value will be TRUE if no objects are present in the frame.

Return Values

Value	**Meaning**
S_OK	Method succeeded.

4.1.8.2 Methods

4.1.8.2.1 INPCameraFrame::Item

The Item method returns the corresponding item in the collection. In this case Item will return an INPObject interface.

HRESULT Item(LONG a_vlIndex, INPObject ****** ppObject**);**

Parameters

A_vlIndex

[in] Index of the item to retrieve.

ppObject

[out, retval] Pointer that receives an INPObject interface.

Return Values

Value	**Meaning**
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.8.2.2 INPCameraFrame::GetObjectData

GetObjectData() provides access to multiple objects in a frame with a single COM call, this can improve performance for frames with large numbers of objects. Once frame->GetObjectData() is called, frame->Item() will no longer return valid NPObjects. Making the call a single time will automatically switch OptiTrack COM into a bulk transfer mode until OptiTrack COM is shut down.

```
struct sCameraObject
{
   float X;
   float Y;
   int Width;
   int Height;
   int Area;
   int Rank;
   int Score;
};
```

```
// Traditional/old object access method
// Pull individual object 2D locations from the INPCameraFrame
for(int i=0; i<m_ObjectCount; i++)
{
   CComPtr<INPObject> object;
   frame->Item(i, &object);
   VARIANT X,Y;
   object->get_X(&X);
   object->get_Y(&Y);
   object.Release();
}


// New Bulk object method
frame->GetObjectData((byte*)m_ObjectTrans, kMax2DObjects*sizeof(sCameraObject),&objectCount);
for(int i=0; i<objectCount; i++)
{
   sCameraObject &object = m_ObjectTrans[i];
}
```

HRESULT GetObjectData((byte*)Buffer, (long) BufferSize , (long*) ReturnedObjectCount);

Parameters

Buffer

[in] destination buffer which the object data will be copied into

BufferSize

[out] size of Buffer

ReturnedObjectCount

[in] the number of objects copied into Buffer for the frame

Return Values

Value	**Meaning**
S_OK	Method succeeded.

4.1.8.2.3 INPCameraFrame::Free

The Free method must be called when all processing is complete for this frame. Camera frames are a limited resource in the system and must be released as soon as possible. All other methods that depend on a camera frame will fail if the frame is accessed once this method has been called.

HRESULT Free();

Parameters

None

Return Values

Value	Meaning
S_OK	Method succeeded.

4.1.9 INPObject

4.1.9.1 Properties

4.1.9.1.1 INPObject::get_Area

Read-only. Area in pixels of the object. Data returned is of type DOUBLE.

HRESULT get_Area(VARIANT * pVal);

Parameters

pVal

[out, retval] Pointer to a VARIANT that receives the area of the object.

Return Values

Value	Meaning
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.9.1.2 INPObject::get_X

Read-only. X position of the object in pixels. Data returned is of type DOUBLE.

HRESULT get_X(VARIANT * pVal);

Parameters

pVal

[out, retval] Pointer to a VARIANT that receives the x position of the object.

Return Values

Value	Meaning

S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.9.1.3 INPObject::get_Y

Read-only. Y position of the object in pixels. Data returned is of type DOUBLE.

HRESULT get_Y(VARIANT * pVal);

Parameters

pVal

[out, retval] Pointer to a VARIANT that receives the y position of the object.

Return Values

Value	Meaning
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.9.1.4 INPObject::get_Score

Read-only. Overall score of the object. Data returned is of type DOUBLE.

HRESULT get_Score(VARIANT * pVal);

Parameters

pVal

[out, retval] Pointer to a VARIANT that receives the score of the object.

Return Values

Value	Meaning
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.9.1.5 INPObject::get_Rank

Read-only. The rank specifies the ranking of the object in the camera frame. The object with a rank of 1 is the tracked object.

HRESULT get_Rank(LONG * pVal);

Parameters

pVal

[out, retval] Pointer to a LONG that receives the rank of the object.

Return Values

Value	Meaning

S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.9.1.6 INPObject::get_Width

Read-only. Specifies the width of the bounding rectangle of the object.

HRESULT get_Rank(LONG * pVal);

Parameters

pVal

[out, retval] Pointer to a LONG that receives the width.

Return Values

Value	Meaning
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.9.1.7 INPObject::get_Height

Read-only. Specifies the height of the bounding rectangle of the object.

HRESULT get_Rank(LONG * pVal);

Parameters

pVal

[out, retval] Pointer to a LONG that receives the height.

Return Values

Value	Meaning
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.9.2 Methods

4.1.9.2.1 INPObject::Transform

Applies any transformations specified by the options set for the given camera. Transformations include camera rotation and mirroring of X and Y axis. If this method is called, the X and Y positions of the object will change slightly. Normally the origin for object positions is the upper left corner of the device. X values increase moving right, y values increase moving down.

After transforming the value, the x and y coordinates will be relative to an origin in the middle of the camera.

HRESULT Transform(INPCamera * pCamera);

Parameters

pCamera

[in] Pointer to an INPCamera interface that the frame originated.

Return Values

Value	**Meaning**
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.10 INPAvi

(INPAvi will remain in the SDK, but support will no longer be provided for it)

The INPAvi interface allows the bitmap images of tracking information to be written to an AVI file. The compression used for the AVI file is the Microsoft MPEG 4 v2 codec.

The image saved to the AVI file is the same image that appears when calling INPCamera::DrawFrame. The color of each object in the view can be changed by the standard NP_OPTION enumeration values.

NOTE: By default the system does not pass empty frames up to client applications. Use the NP_OPTION_SEND_EMPTY_FRAMES option to receive these frames. If this option is not set, only frames with visible objects will be written to the AVI file.

4.1.10.1 Properties

4.1.10.1.1 INPAvi::get_FileName

File name to be used when saving the AVI file. The default file name is optitrack.avi. If a full path is not specified, the file will be located in the current working directory of the process.

HRESULT get_FileName(BSTR * pVal);

Parameters

pVal

[out, retval] Pointer to a BSTR that receives the file name.

Return Values

Value	**Meaning**
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.10.1.2 INPAvi::put_FileName

Sets the file name of the output AVI.

HRESULT put_FileName(BSTR Val);

Parameters

Val

[in] VARIANT that contains a BSTR for the file name.

Return Values

Value	Meaning
S_OK	Method succeeded.

4.1.10.1.3 INPAvi::get_FrameRate

Number of frames per second that will be written to the AVI file. The default is 120.

HRESULT get_FileName(LONG * pVal);

Parameters

pVal

[out, retval] Pointer to a LONG that receives the frame rate.

Return Values

Value	Meaning
S_OK	Method succeeded.
E_POINTER	Pointer is invalid.

4.1.10.1.4 INPAvi::put_FrameRate

Sets the frame rate of the AVI file.

HRESULT put_FileName(LONG Val);

Parameters

Val

[in] LONG that contains the frame rate.

Return Values

Value	Meaning
S_OK	Method succeeded.

4.1.10.2 Methods

4.1.10.2.1 INPAvi::Start

Initializes the AVI file for writing. This function must be called before calling AddFrame.

HRESULT Start();

Parameters

none

Return Values

Value	Meaning

S_OK	Method succeeded.

4.1.10.2.2 INPAvi::Stop

Closes the AVI file when complete.

HRESULT Stop();

Parameters

none

Return Values

Value	**Meaning**
S_OK	Method succeeded.

4.1.10.2.3 INPAvi::AddFrame

Writes a frame to the AVI file.

HRESULT AddFrame(INPCamera * pCamera, INPCameraFrame * pFrame)

Parameters

pCamera

[in] Pointer to an INPCamera object that the frame originated.

pFrame

[in] Pointer to an INPCameraFrame object. This is the frame that will be written to the AVI file.

Return Values

Value	**Meaning**
S_OK	Method succeeded.

4.1.11 _INPCameraCollectionEvents

4.1.11.1 Methods

4.1.11.1.1 _INPCameraCollectionEvents::DeviceRemoval

This method is called when a device is removed from the system.

void DeviceRemoval(INPCamera * pCamera);

Parameters

pCamera

[in] Pointer to an INPCamera object. The camera object is the object that has been removed from the system.

Return Values

None.

4.1.11.1.2 _INPCameraCollectionEvents::DeviceArrival

This method is called when a device is added to the system.

void DeviceArrival(INPCamera * pCamera);

Parameters

pCamera

[in] Pointer to an INPCamera object. The camera object is the object that has been added to the system.

Return Values

None.

4.1.12 _INPCameraEvents

4.1.12.1 Methods

4.1.12.1.1 _INPCameraEvents::FrameAvailable

This method is called when the camera has captured a frame.

void FrameAvailable(INPCamera * pCamera);

Parameters

pCamera

[in] Pointer to an INPCamera object. This is the camera object that captured the frame. In order to get the most recent frame, call the GetFrame method of INPCamera.

Return Values

None.

4.1.12.1.2 _INPCameraEvents::SwitchChange

This method is called when the switch state has changed.

void SwitchChange(INPCamera * pCamera, LONG lNewSwitchState);

Parameters

pCamera

[in] Pointer to an INPCamera object. This is the camera object that captured the frame. In order to get the most recent frame, call the GetFrame method of INPCamera.

lNewSwitchState

[in] New state of the switch. Switch states are defined by the NP_SWITCH_STATE enumeration.

Return Values

None.

5 Sample Code

5.1 VBScript

Several VBScript sample files are provided in the SDK. These scripts show how to enumerate cameras, get information from them and query the data. The following scripts are provided:

camInfoLED.vbs

This script is an example of how to enumerate cameras connected to the system and display their information. It also shows how to hook up connection points to receive information about camera removal and arrival. Finally, it shows how to control the LEDs on the device.

camData.vbs

This script is an example of how to get camera frame data from the device. It builds upon the camInfoLED script to get notifications when a frame is available.

camSwitch.vbs

This script is shows how to get information about the state of the switches. The script uses connection points on the camera object to receive notifications.

camVector.vbs

This script is an example of how to calculate the Vector positions. All 6 degrees of freedom are computed.

camRecord.vbs

This script is an example of how to save the tracking view to an AVI file.

5.2 VB.NET

A sample application written in Visual Basic .NET is provided in the SDK. The application can control one camera at a time, but is able to switch between any cameras attached to the system. All options available through the API are accessible through the options dialog off the main dialog. Connection point callbacks are used for device notification and data availability.

5.3 VC

The C/C++ sample application is available in a Visual C/C++ project. The VC project was chosen for maximum compatibility as many developers use VC.

This application exercises all components of the OptiTrack API. Cameras can be enumerated, data gathered, smoothed and Vector positions calculated. The sample will also draw the camera image. The sample also includes code to enable connection points for callbacks.