Connect SDK

of

Optris PI Connect Software

Version 2

(PI Connect Rel. 2.17.2229.0 and ImagerIPC2.dll 2.6.2030.0)

The communication to the process imager device is handled by the optris PI Connect software (Imager.exe) only. Optris supplies a dynamic link library (ImagerIPC2.dll) that serves the **Inter Process Communication (IPC)** for other attached processes. The DLL can be dynamically linked into the secondary application (client application). Or it can be done static by a lib file too.
Both Imager.exe and ImagerIPC2.dll are designed for MS Windows 7/8/10 only.
There is also a ImagerIPC2x64.dll to support Intel 64 bit platforms.

The ImagerIPC2.dll will export a bunch of functions that are responsible for initiating the communication, retrieving data and setting some control parameters.

You will find a short roadmap to start with a project at the end of this document. In addition there are some sample projects on the PI installation CD-ROM: Small samples in C++, C++/CLI, C# and Visual Basic (VBA embedded in a MS-Word document) and an extensive sample for C++/CLI (.NET).

The main difference to the former Version 1 (ImagerIPC.dll) is the support of more than one optris PI via multiple instances of optris PI Connect. In the appendix you will find some hints for switching from ImagerIPC.dll to ImagerIPC2.dll.

On the optris PI Connect CD-ROM you will find some samples to check out how the IPC is working.

Sample name	Progr.Platform	Description
Start IPC2	C++/CLI	A simple program to demonstrate basic concepts of IPC2 with C++/.NET
Start IPC2 Extensive	C++/CLI	A program to demonstrate advanced concepts of IPC2
Start IPC2 VisCam	C++/CLI	A program to demonstrate communication on both channels IR and Visible (if available)
Start MultiIPC2	C++/CLI	A program to demonstrate the multiple instance capabilities of IPC2
Start IPC2 Win32	C++	A simple Win32 program as a classical Windows-application
Start IPC Console Polling	C	A simple console program using polling
Start IPC Console CB	C	A simple console program using callbacks
Start IPC CSharp	C#	A simple C# program using callbacks or polling
PI_in_Word	VBA / Word	Simple macro's embedded in a MS Word Document

Content

1 Init procedure

1.1 Init

The application that uses the ImagerIPC2.dll must first call the `SetImagerIPCCount`-function to setup the count of imagers (respectively the number of instances of optris PI Connect):

Function	Description
`SetImagerIPCCount`	Sets the number of imagers

Parameters: Number of imagers (unsigned short)

Return value: Error/Success code (HRESULT)

This function is new in Version 2.

After that the `InitImagerIPC-` or the `InitNamedImagerIPC`-function must be called for any imager:

Function	Description
`InitImagerIPC`	Initiziates the IPC
`InitNamedImagerIPC`	Initiziates the IPC to an instance of imager.exe that was called with an instance name

Parameters:

`InitImagerIPC`:
- Imager index (unsigned short)

`InitNamedImagerIPC`:
- Imager index (unsigned short)
- Instance name (wchar_t*)

Return value: Error / Success code (HRESULT).

The `InitNamedImagerIPC`-function is needed if the PI Connect software was called with an instance name (command prompt parameter "/name=xxx").
If the return value signals no success (less than zero) you can call the function again and again to wait for answer of the imager.exe.

After this "Init" command is the point to decide to operate with callbacks or in polling mode. The callback mode is enabled with setting at least one callback function. If no callback function is set between the call of "Init" and "Run" function the IPC will operate in polling mode and expect repeatedly calls of GetFrame() and GetIPCState().

In most samples you will find a compiler switch "POLLING". Uncommenting or not defines whether the application will be compiled for polling mode or not.

1.2 Setting callbacks

If the “Init” function was successful and the user wants to operate in callback mode the “SetCallback_xxxx” functions have to be executed to load pointers to the callback functions (or delegates in .net):

Function	Description
`SetCallback_OnServerStopped`	Sets the `OnServerStopped` event, it occurs if PI connect stopps the IPC
`SetCallback_OnFrameInit`	Sets the `OnFrameInit` event, it occurs before the first frame will be sent
`SetCallback_OnNewFrame`, `SetCallback_OnNewFrameEx`	Sets the `OnNewFrame` or `OnNewFrameEx` event, it occurs for any new video frame, only one of both callbacks should be set
`SetCallback_OnVisibleFrameInit`	Sets the `OnVisibleFrameInit` event (for the channel of the visible cam)
`SetCallback_OnNewVisibleFrame`, `SetCallback_OnNewVisibleFrameEx`	Sets the `OnNewVisibleFrame` or `OnNewVisibleFrameEx` event (for the channel of the visible cam)
`SetCallback_OnInitCompleted`	Sets the `OnInitCompleted` event, it occurs if the IPC initialization is complete
`SetCallback_OnConfigChanged`	Sets the `OnConfigChanged` event. It occurs if the configuration of PI Connect has changed
`SetCallback_OnFileCommandReady`	Sets the `OnFileCommandReady` event. It occurs if the resulting files have been written (e.g. Snapshot, Record,..)
`SetCallback_OnNewNMEAString`	Sets the `OnNewNMEAString` event. It occurs if a new NMEA string is available.

Parameters:
- Imager index (unsigned short)
- a function pointer (or delegate in .NET) that matches the callback function

Return value: Error / Success code (HRESULT).

When working with multiple instances, any instance must have its own set of callback functions. And the appropriate pointers must be set with the SetCallback-functions.

After the `OnInitCompleted` event is occurred all other functions to retrieve or set values are ready to be called. Don’t try to use get or set function before this callback was called.

The DLL will give an `OnFrameInit` event after a short time and provide information about width, height and depth of the frames. The application can now allocate memory for its own purposes.

1.3 Run

Now the application should call the `RunImagerIPC`-function. In callback mode it is important to set all callbacks before you call this function. `RunImagerIPC` must be called for any imager:

Function	Description
`RunImagerIPC`	Runs the IPC

Parameters:
- Imager index (unsigned short)

Return value:
- Error/Success code (HRESULT).

The `RunImagerIPC`-function notifies the optris PI Connect software the initialization is ready and the data transmission can begin.
If the return value signals no success (less than zero) you should start over with the initialization procedure.

1.4 Clean up

Function	Description
`ReleaseImagerIPC`	Stopps the IPC

Parameters:
- Imager index (unsigned short)

Return value:
- Error/Success code (HRESULT)

The Release-function should be called to stop the communication with the Imager.exe in the following situations:

- On closing the application
- After an `OnServerStopped` event
- If the application don't use the IPC any longer.
- Before connecting to another instance (`InitNamedImagerIPC`)

2 Callbacks (Events)

When operating in callback mode for any function pointer you handle with a set-callback-function an appropriate function must be defined. If you are working with more than one imager you need multiple definitions of the functions too. These functions that are called by the IPC are listed in the following:

2.1 OnServerStopped

Parameters: Reason (int)
- 0 → imager.exe has closed the IPC regulary
- 1 → imager.exe does not communicate any longer

Return value: Success (HRESULT), 0 to signal success

Note: Some actions of the imager.exe need some drastic re-allocating of objects within the imager.exe. Therefore the imager.exe will relaunch the IPC and call the `OnServerStopped` event also after this actions. For that reason you application should be able to respond to that by reinitializing the IPC.

2.2 OnFrameInit

Parameters:
1. Width (int): width of the video frame
2. Height (int): height of the video frame
3. Depth (int): bytes for any pixel (allways 2)

Return value: Success (HRESULT), 0 to signal success

This function will be called prior to the first call of `OnNewFrame` and gives you the opportunity to allocate memory or create objects that depends from the size of the video frames. The size of the buffer that will contain the video data is Width*Height*Depth.

2.3 OnNewFrame

Parameters:
1. Buffer (char*): pointer to the buffer that contains the video frame
2. FrameCounter (int): the internal frame counter of imager.exe

Return value: Acknowledge (HRESULT)
0: force the imager.exe to send the next frame as soon as it is possible.
< 0: the imager.exe will wait for calling the `AcknowledgeFrame` function prior sending the next frame

2.4 OnNewFrameEx (supported as of PI Connect 2.0)

Parameters:
1. Buffer (char*): pointer to the buffer that contains the video frame
2. pMetadata (FrameMetadata*): pointer to the metadata structure that contains additional informations to the frame

Return value: See the `OnNewFrame`-event

2.5 OnVisibleFrameInit, OnNewVisibleFrame, OnNewVisibleFrameEx

Same as `OnFrameInit`, `OnNewFrame`, `OnNewFrameEx` but for the channel of the visible camera. The buffers of the `OnNewVisibleFrame` and `OnNewVisibleFrameEx` events always contain YUV values (16 bit).

2.6 OnInitCompleted

Parameters: -
Return value: - Success (HRESULT), 0 to signal success

The DLL calls this function to notify that the initialization is completed. You must wait for this event before you can use any other function.

2.7 OnConfigChanged

Parameters: - Reserved (long)
Return value: - Success (HRESULT), 0 to signal success

The DLL calls this function to notify that the configuration has changed. If this event has occurred you should re-read all information from the imager that is important for you.

2.8 OnFileCommandReady

Parameters: - path (wchar_t*) : The complete path to the file name of the stored file (Ravi or snapshot)
Return value: - Success (HRESULT), 0 to signal success

This event notifies that a file was successfully stored.

2.9 OnNewNMEAString

Parameters: - NMEA string (wchar_t*) : **R**ecommended **M**inimum Sentence **C** (RMC), containing the GPS information acquired during frame acquisition.
Return value: - Success (HRESULT), 0 to signal success

This event notifies about a availablity of a new NMEA-String.

3 Polling

If your application cannot deal with callbacks or if you have other reasons for it you can also use polling to acquire all data from the IPC. In this case you must not set any of the `SetCallback_xxxx` functions but must repeatedly call the following functions. You can do this controlled by a timer ore if the application is in idle state.

Function	Description
`GetIPCState`	Get information about events that happened in the DLL

Parameters:
- Imager index (unsigned short)
- Reset (bool)

Return value:
- Returns the events that are occurred in the DLL

If the DLL shall reset existing events, the Reset parameter must be set to true (normal operation). The appropriate bit will not be set on the next call of the function until the event occurs again. The return value can be interpreted as a bitfield with the following bits:

Bit	Name of bit	Corresponding event	Function that should be called after event:
0	IPC_EVENT_INIT_COMPLETED	OnInitCompleted	
1	IPC_EVENT_SERVER_STOPPED	OnServerStopped	
2	IPC_EVENT_CONFIG_CHANGED	OnConfigChanged	
3	IPC_EVENT_FILE_CMD_READY	OnFileCommandReady	GetPathOfStoredFile
4	IPC_EVENT_FRAME_INIT	OnFrameInit	GetFrameConfig
5	IPC_EVENT_VIS_FRAME_INIT	OnVisibleFrameInit	GetVisibleFrameConfig
6	IPC_EVENT_NEW_NMEA_STRING	OnNewNMEAString	

After the `IPC_EVENT_FRAME_INIT` or `IPC_EVENT_VIS_FRAME_INIT` bit was set, you should call the following function to retrieve information about the size of the video frames. According to the `OnFrameInit` or `OnVisibleFrameInit` event you can use this information to create resources that depend on the size of the video frames.

Function	Description
GetFrameConfig, GetVisibleFrameConfig	Return Width, Height, Depth

Parameters:
- Imager index (unsigned short)
- Width (int*): width of the video frame
- Height (int*): height of the video frame
- Depth (int*): bytes for any pixel (always)

Return value:
- Error/Success code (HRESULT)

After the `IPC_EVENT_FILE_CMD_READY` bit was set, you should call the following function to retrieve the complete path of the file that was stored.

Function	Description
GetPathOfStoredFile	Returns the path of the stored file (Ravi or snapshot)

Parameters:
- Imager index (unsigned short)
- Pointer to the string buffer that return the filename (wchar_t*)
- Max length of the string buffer (int)

Return value:
- Error / Success code (HRESULT)

Appropriate to the events `OnNewFrameEx` and `OnNewVisibleFrameEx` there are functions to retrieve frames from the IPC:

Function	Description
GetFrame and GetVisibleFrame	Return frames with data and metadata

Parameters:
- Imager index (unsigned short)
- Timeout (unsigned short) in msec.
- Pointer to the buffer that return data (void*)
- Buffer size (unsigned int)
- Pointer to metadata (FrameMetadata*)

Return value:
- Error / Success code (HRESULT)

To prevent calling `OnNewFrameEx` and `OnNewVisibleFrameEx` more than necessary or to see how far behind you are from the most recent frame it is possible to ask how many entries are in the frame queue:

Function	Description
GetFrameQueue, GetVisibleFrameQueue	Return number of entries in the frame queue

Parameters:
- Imager index (unsigned short)

Return value:
- Number of queue entries (unsigned short)

The interprocess communication offers logging capability for multiple applications. One option for enable logging is to specify a log file the following function:

Function	Description
SetLogFile	Set initial parameters for logging process.

Parameters:
- Path/Name of the log file(wchar_t*)
- Logging levels: init(0), command(1), frame(2), frameplus(3) (integer)
- Append to log file (bool)

Return value:
- Error / Success code (HRESULT)

A second possibility to enable the logging functionality can be achieved by calling:

Function	Description
SetLogging	Set initial parameters for logging process.

Parameters:

- Logging group (32-bit mask)
  LOGGRP_INIT(bit0)
  LOGGRP_ALIVE(bit1)
  LOGGRP_FRAME(bit 2)
  LOGGRP_FRAMEINIT(bit 3)
  LOGGRP_STRING(bit 4)
  LOGGRP_STATE(bit 5)
  LOGGRP_COMMAND(bit 6)
  LOGGRP_DEVINFO(bit 7)
  LOGGRP_DEVSETUP(bit 8)
  LOGGRP_DEVDATA(bit 9)
  LOGGRP_SYSINFO(bit 10)

Return value:

- Error / Success code (HRESULT)

The logging command for custom log messages is defined as follows. Messages can be assigned to a distinct logging level. Levels can be used to group different events.

Function	Description
Log	Logs an event

Parameters:

- Imager index (int)
- String to log (char *)
- Logging levels (int):
  init(0)
  command(1)
  frame(2)
  frameplus(3)

Return value:

- Error / Success code (HRESULT)

4 Video frames

After the initialization the IPC will call the callback function `OnFrameInit`. Or the user gets this event after a call to `GetIPCState`. Here the user can allocate memory or create objects that needs the size of the video frame.
Thereafter the first `OnNewFrame` callback function will be called or the user calls the `GetFrame` function. The application can read out the data of the buffer according to width/height/depth and can do with the data what it wants to do.
In case of callbacks: If the `OnFrameInit` function returns with zero the next frame will be send if it is available. If the return value is less than zero the application has to call the `AcknowledgeFrame` function later.

With both methods the secondary application can control the frame rate it wants to process frames.

Function	Description
`AcknowledgeFrame`	Notify the imager.exe to send the next frame after you left the `OnNewFrame` callback function with a return value less than zero

Parameters: - Imager index (unsigned short)

Return value: - Error/Success code (HRESULT)

In theory the full frame rate (device frame rate) could be transferred if the main application is setup with the option “Highspeed temperature calculation” (otherwise the frame rate of the IPC is the same as the display frame rate). Practically it depends from the computer hardware and from the effort the application has to process the data.

Because the depth/data size is always 2 bytes, any pixel represents a 16 bit value.
In the Imager.exe the content of the frame information can be chosen:

1. AD-Values (energy): Pixels contain pre-corrected energy values. For most applications this will be meaningless.
2. Temperature values: Pixels contain temperatures in °C. Depending from the return value of the `GetTempRangeDecimal`, the temperature can be calculated by the formulas:
   - 1 decimal place: T[°C] = (value - 1000) / 10
   - 2 decimal places: T[°C] = value / 100
3. YUV-colour values: Any macro-pixel (two neighbouring pixels in a row) contains the YUV colour information of two pixels.

To get what information this is you can use the following function:

Function	Description
`GetIPCMode`	Sets the IPC mode.

Parameters: - Imager index (unsigned short)

Return value: - Mode (unsigned short):
0 = ADU, 1 = Temp, 2 = Color

To set which information you want to have, use the following function:

Function	Description
`SetIPCMode`	Sets the IPC mode.

Parameters:
- Imager index (unsigned short)
- Mode (unsigned short):
  0 = ADU, 1 = Temp, 2 = Color

Return value:
- Mode (unsigned short)

Number of decimal places for temperature values:

Function	Description
`GetTempRangeDecimal`	Returns the number of decimal places for the temperature values of the matrix.

Parameters:
- Imager index (unsigned short)
- EffectiveValue (bool)

Return value:
- Number of decimal places (unsigned short)

The number of decimal places can be 1 or 2. This depends from the calibration of the temperature range.
If the parameter EffectiveValue is set to true the function returns the effective value for the number of decimal places that is depending from the calibrated data and from the checkbox “High resolution temperatures” on tab “Device” in the configuration dialog. This information is needed to determine in which format the temperature values of the matrix will be send:

Decimal places	Data format for temperature values	Formula to calculated the floating point value:	Example:
1	Unsigned short	T[°C] = (value - 1000) / 10	value = 1235 → T = 23.5 °C
2	Signed short	T[°C] = value / 100	value = 2357 → T = 23.57 °C

The 16 bit values are arranged in lines. That means the first word in the buffer is the first (most left) pixel in the first line. It follows the second pixel of the first line. ... up to the last (most right) pixel of the first line. It follows the second line and so on. The following scheme demonstrates the arrangement of the pixels for a 160x120 matrix:

rows	0	1	2	columns	157	158	159
0	0	1	2	...	157	158	159
1	160	161	162	...	317	318	319
2	320	321	322	...	477	478	479
...	...	...	...	...	...	...	...
119	19080	19081	19082	...	19197	19198	19199

5 Other data

All other data can be retrieved or manipulated by a couple of “Get” and “Set” functions.
The following functions will be available for the user:

5.1 Single temperature information

These functions retrieve temperatures:

Function	Description
GetTempTec	Temperature of the thermo-electric cooler
GetTempChip	Get the temperature of the bolometer chip
GetTempFlag	Get the temperature of the flag
GetTempProc	Get the temperature of the processor
GetTempBox	Get the temperature of the box
GetTempHousing	Get the temperature of the housing
GetTempTarget	Get the temperature of the main measuring area

Parameters:
- Imager index (unsigned short)

Return value:
- temperature in °C (float)

The only temperature you can set is that of the thermo-electric cooler:

Function	Description
SetTempTec	Temperature of the thermo-electric cooler

Parameters:
- Imager index (unsigned short)
- Set temperature in °C (float)

Return value:
- temperature in °C (float).

5.2 IR Image Arranging

IR Arranging is both rotation of the image and zoom (changing the size). With the following function you can set and get both in one structure (see definition in the appendix).

Function	Description
GetIRArranging	Get the complete IRAranging struct
SetIRArranging	Set the complete IRAranging struct

Parameters:
- imager index (unsigned short)
- IR Arranging struct (IRArranging*)

Return value:
- Error / Success code (HRESULT)

5.3 Measure area

These functions are needed to retrieve temperatures of any measure area:

Function	Description
`GetMeasureAreaCount`	Get the number of defined measure areas
`GetTempMeasureArea`	Get the tempearture of a measure area

First call `GetMeasureAreaCount` to retrieve the count of measure areas:

Parameters:
- imager index (unsigned short)

Return value:
- count of measure areas (unsigned short)

Then call `GetTempMeasureArea` to get the temperature for any area:

Parameters:
- imager index (unsigned short)
- index of the measure area (unsigned long)

Return value:
- temperature in °C (float)

Use these functions to get and set the location of any measure area instantaneously:

Function	Description
`GetLocMeasureArea`	Get the location of a measure area
`SetLocMeasureArea`	Get the location of a measure area

First call `GetMeasureAreaCount` to retrieve the count of measure areas:

Parameters:
- imager index (unsigned short)
- index of the measure area (unsigned long)

`GetLocMeasureArea`:
- Location of measure area (POINT)

`SetLocMeasureArea`:
- Location of measure area (POINT*)

Return value:
- Error/Success code (HRESULT)

An Extended functionality for creating, manipulating and deleting measure areas are provided by the following functions:

Function	Description
`RemoveMeasureArea`	Deletes a measure area
`GetMeasureArea`	Gets a struct which represents a measure area
`SetMeasureArea`	Sets a measure area using a struct

Parameters:
- imager index (unsigned short)
- index of the measure area (unsigned long)

`SetMeasureArea/GetMeasureArea`:
- measure area attributes (MeasureArea*)

`SetMeasureArea`:
- The value indicates, if an exisiting area is changed or a new area is added. If true, area is appended, index of area is neglected. (bool)

Return value:
- Error / Success code (HRESULT)

The name of a measure area can be defined/ retrieved by using the functions:

Function	Description
`SetMeasureAreaName`	Set the measure area's name.
`GetMeasureAreaName`	Get the measure area's name.

Parameters:
- imager index (unsigned short)
- index of the measure area (unsigned long)
- name of the measure area (wchar_t*)

`GetMeasureAreaName`:
- length of the retrieved area name (int*)
- maximal length of the name (int)

Return value:
- Error / Success code (HRESULT)

5.4 Flag

Functions to set and get the flag state:

Function	Description
`GetFlag`	Get the state of the flag
`SetFlag`	Set the state of the flag

Parameters:

`GetFlag`:
- imager index (unsigned short)

`SetFlag`:
- imager index (unsigned short)
- Set flag state (bool):
  - true = close flag
  - false = open flag

Return value:
- Flag state (bool):
  - true = flag is closed
  - false = flag is open

Renew the flag (complete flag cycle):

Function	Description
`RenewFlag`	Renew the flag

Parameters:
- imager index (unsigned short)

Return value:
- Success (bool):
  - true = flag is renewed
  - false = flag could not be renewed (possible reason: flag is controlled by PIF)

5.5 Optics

Functions to handle the optics:

Function	Description
`GetOpticsCount`	Get the number of existing optics
`GetOpticsIndex`	Get the index of the used optics (starts with 0)
`SetOpticsIndex`	Set the index of the used optics
`GetOpticsFOV`	Get the used field of view of the optics

First call `GetOpticsCount` to retrieve the count of existing optics:

Parameters: - imager index (unsigned short)
Return value: - count of existing optics (unsigned short)

Call `GetOpticsIndex` to get the optics that is in use:

Parameters: - imager index (unsigned short)
Return value: - optics index (unsigned short)

Call `SetOpticsIndex` to set the optics you want to use:

Parameters: - imager index (unsigned short)
- optics index (unsigned short)
Return value: - optics index (unsigned short)

Call `GetOpticsFOV` to get the field of view of the optics with the given index:

Parameters: - imager index (unsigned short)
- optics index (unsigned long)
Return value: - angle in ° (unsigned short)

5.6 Temperature Ranges

Functions to handle the temperature ranges:

Function	Description
`GetTempRangeCount`	Get the number of existing temp. ranges
`GetTempRangeIndex`	Get the index of the used temp. range (starts with 0)
`SetTempRangeIndex`	Set the index of the used temp. range
`GetTempMinRange`	Get the min temperature of the used temp. range
`GetTempMaxRange`	Get the max temperature of the used temp. range

First call `GetTempRangeCount` to retrieve the count of existing temperature ranges:

Parameters: - **imager index (unsigned short)**
Return value: - count of existing temperature range (unsigned short)

Call `GetTempRangeIndex` to get the temperature range that is in use:

Parameters: - **imager index (unsigned short)**
Return value: - temperature range index (unsigned short)

Call `SetTempRangeIndex` to set the temperature range you want to use:

Parameters: - **imager index (unsigned short)**
- **temperature range index (unsigned short)**
Return value: - temperature range index (unsigned short)

Call `GetTempMinRange` and `GetTempMaxRange` to get the lower and upper limit of the temperature range with the given index:

Parameters: - **imager index (unsigned short)**
- **temperature range index (unsigned long)**
Return value: - temperature in °C (float)

5.7 VideoFormats

Functions to handle the video formats:

Function	Description
GetVideoFormatCount	Get the number of existing video formats
GetVideoFormatIndex	Get the index of the used video format (starts with 0)
SetVideoFormatIndex	Set the index of the used video format
GetVideoFormat	Get the video format with the given index
GetClippedFormatPos	Get the position of the imager's sub frame mode
SetClippedFormatPos	Set the position of the imager's sub frame mode

First call `GetVideoFormatCount` to retrieve the count of existing video formats depending from the connected camera:

Parameters: - imager index (unsigned short)
Return value: - count of existing video formats (unsigned short)

Call `GetVideoFormatIndex` to get the video format that is enabled:

Parameters: - imager index (unsigned short)
Return value: - video fromat index (unsigned short)

Call `SetVideoFormatIndex` to set the video fromat that should be enabled:

Parameters: - imager index (unsigned short)
- video fromat index (unsigned short)
Return value: - video fromat index (unsigned short)

Call `GetVideoFormat` to get the video fromat with the given index:

Parameters: - imager index (unsigned short)
- video fromat index (unsigned long)
- Pointer to a video fromat structure (VideoFormat*) that returns the video format
Return value: - Error/Success code (HRESULT)

Call `GetClippedFormatPos` to get the imager's clipped format position:

Parameters: - imager index (unsigned short)
- Location of the clip (POINT*)
Return value: - Error/Success code (HRESULT)

Call `SetClippedFormatPos` to set the imager's clipped format position:

Parameters: - imager index (unsigned short)
- Location of the clip (POINT)
Return value: - Error/Success code (HRESULT)

5.8 Measuring settings

Functions to get ans set measuring settings:

Function	Description
`GetFixedEmissivity`	Gets the fixed value for emissivity
`SetFixedEmissivity`	Sets the fixed value for emissivity
`GetFixedTransmissivity`	Gets the fixed value for transmissivity
`SetFixedTransmissivity`	Sets the fixed value for transmissivity
`GetFixedTempAmbient`	Gets the fixed value for ambient temperature
`SetFixedTempAmbient`	Sets the fixed value for ambient temperature

Parameters:

`Get`-Functions:

- imager index (unsigned short)

`Set`-Functions:

- imager index (unsigned short)
- Set value (float)
  - for emissivity and transmissivity 0.1 … 1.1
  - for TAmbient temperature in °C

Return value:

- Value (float)
- for emissivity and transmissivity 0.1 ... 1.1
- for TAmbient temperature in °C

5.9 Setting Output if the Process Interface (PIF)

These function set the three output pins of the PIF:

Function	Description
`SetPifOut`	SetPifOut

Parameters:

- imager index (unsigned short)
- output channel (unsigned short): AO1 = 0, AO2 = 1, AO3 = 2
- output value (float), allowed values 0.0 ... 10.0

Return value:

- output value (float)

5.10 Getting version information

These functions retrieve the versions of the corresponding executable files:

Function	Description
`GetVersionApplication`	Version of PI Connect
`GetVersionHID_DLL`	Version of the SteuerHID.dll
`GetVersionCD_DLL`	Version of the ControlDevice.dll
`GetVersionIPC_DLL`	Version of the ImagerIPC2.dll

Parameters:

- imager index (unsigned short)

Return value:

- version (__int64): four 16 bit values (words): Major-Version, Minor-Version, Build, Revison

5.11 Application appearance

With the following functions the connected application can control the appearance of PI connect. By switching it to the “Embedded State” and moving and resizing it in an intelligent way the PI Connect video window can look like as it belongs to the connected application.
If the “Embedded state” is set the video window can be embedded into any application. The user sees just the video window, no tool windows, no tool bars, and no frames. The window is always on top.

Functions to set and get the embedded state.

Function	Description
`GetMainWindowEmbedded`	Get the main window embedded state
`SetMainWindowEmbedded`	Set the main window embedded state

Parameters:

`GetMainWindowEmbedded`:
- imager index (unsigned short)

`SetMainWindowEmbedded`:
- imager index (unsigned short)
- set state (bool)

Return value:
- embedded state (bool)

Functions to set and get the application location:

Function	Description
`GetMainWindowLocX`	Get the main window left coordinate
`SetMainWindowLocX`	Set the main window left coordinate
`GetMainWindowLocY`	Get the main window top coordinate
`SetMainWindowLocY`	Set the main window top coordinate
`GetMainWindowWidth`	Get the main window width
`SetMainWindowWidth`	Set the main window width
`GetMainWindowHeight`	Get the main window height
`SetMainWindowHeight`	Set the main window height

First call `GetOpticsCount` to retrieve the count of existing optics:

Parameters:

`Get`-Functions:
- imager index (unsigned short)

`Set`-Functions:
- imager index (unsigned short)
- coordinate (unsigned short)

Return value:
- Coordinate (unsigned short)

5.12 File management

There are a number of commands to start and stop recording and playing, and to take and reopen snapshots:

`FileOpen`-function:

Function	Description
`FileOpen`	Opens a file with radiometric data. This can be ravi-, tiff- or a jpeg-file.

Parameters:
- imager index (unsigned short)
- file name (wchar_t*)

Return value:
- Error / Success code (HRESULT).

Note: Opening files with an included layout that use no inter-process communication will cause the current connection to be interrupted. It is recommended to answer the question to load the layout with “No” and mark the checkbox “Don’t ask me again”.

Other file management functions:

Function	Description
`FileSnapshot`	Takes a snapshot. The file name is automatically set according to the template of the triggered recording. If the snapshot is stored the application will be notified by a `OnFileCommandReady`-event.
`FileRecord`	Starts recording. The file name is automatically set according to the template of the triggered recording.
`FileStop`	Stops recording, playing and viewing a snapshot. If an imager is connected it will return to live view. After stop recording: If the ravi file is stored the application will be notified by a `OnFileCommandReady`-event.
`FilePause`	Pauses a ravi file that is currently playing.
`FilePlay`	Plays a ravi file that is currently paused.

Parameters:
- imager index (unsigned short)

Return value:
- Error / Success code (HRESULT).

5.13 Layout management

With the following command a layout can be activated by the IPC:

`LoadLayout`-function:

Function	Description
`LoadLayout`	Loads the layout with the name.

Parameters:
- imager index (unsigned short)
- layout name (wchar_t*)

Return value:
- Error / Success code (HRESULT).

The layout name must not contain the extension xml and the directory of the layout. The layout will be searched in the MeasuringLayout directory of the configuration path.

Note: Opening a layout that uses no inter-process communication will cause the current connection to be interrupted.

5.14 Other commands

Function	Description
GetSerialNumber	Get the serial number of the PI
GetPIFSerialNumber	Get serial number of process interface

Parameters:
- imager index (unsigned short)

Return value:
- serial number (unsigned long)

Function	Description
CloseApplication	Closing PI Connect

Parameters:
- imager index (unsigned short)

Return value:
- (void)

Function	Description
ReinitDevice	Reinitialze the communication between device and PI connect

Parameters:
- imager index (unsigned short)

Return value:
- (void)

Function	Description
GetInitCounter	Get the “Initialization Counter”

Parameters:
- imager index (unsigned short)

Return value:
- initialization counter (unsigned short): Estimated time in msec. until the application will be initialized. Can be used to control a progress bar.

Function	Description
GetAvgTimePerFrame	Get the average time for a frame
GetVisisbleAvgTimePerFrame	Get the average time for a visible frame

Parameters:
- imager index (unsigned short)

Return value:
- average time per frame in [100 ns]

Function	Description
GetHardwareRev	Get the hardware revision
GetFirmwareRev	Get the firmware revision

Parameters:
- imager index (unsigned short)

Return value:
- revision (unsigned short)

Function	Description
`GetPID`	Get the USB product ID
`GetVID`	Get the USB vendor ID

Parameters:
- imager index (unsigned short)

Return value:
- ID (unsigned short)

Function	Description
`GetPIFVersion`	Get version of process interface

Parameters:
- imager index (unsigned short)

Return value:
- version (unsigned short)

6 Appendix

6.1 A. Roadmap for an application that uses the ImagerIPC2.dll

This is a short step-by-step operational list for implementing an application that uses the ImagerIPC2.dll in callback or polling operation:

Callback operation:

- Set the number of imagers (respectively instances of optris PI Connect) by calling the `SetImagerIPCCount`-function. When using more than one imager the following steps must be done for any imager.
- Call (repeatedly) the `InitImagerIPC-` or the `InitNamedImagerIPC-` function and check if the return value signals success.
- Call all SetCallback functions with the appropriate function pointers.
- Call the the `RunImagerIPC`-function and check if the return value signals success.
- Wait for the `OnInitCompleted` event. After the event occurred you are allowed to call all get- and set-functions.
- Call get functions to retrieve configuration data you need from the imager.exe.
- Wait for the `OnFrameInit` event. If the event was occurred create all objects you need to process the data from the video frames.
- Process all `OnNewFrame` events. Enable sending further frames either by returning zero in the OnNewFrame callback function or by returning a value less than zero and using the `AcknowledgeFrame` function.
- Call all get functions that retrieves temperature values you need repeatedly.
- ...
- If an `OnConfigChanged` event occurs read all configuration data again.
- ...
- If an `OnServerStopped` event occurs, release all your objects and call the `ReleaseImagerIPC` function. Then reinitialize the IPC.
- ...
- ...
- Call the `ReleaseImagerIPC` function before closing your application.

Polling operation:

- Set the number of imagers (respectively instances of optris PI Connect) by calling the `SetImagerIPCCount`-function. When using more than one imager the following steps must be done for any imager.
- Call (repeatedly) the `InitImagerIPC-` or the `InitNamedImagerIPC-` function and check if the return value signals success.
- Call the the `RunImagerIPC`-function and check if the return value signals success.
- Wait until the `IPC_EVENT_INIT_COMPLETED` bit is set in the return value of the `GetIPCState` call. After the event occurred you are allowed to call all get- and set-functions.
- Call get functions to retrieve configuration data you need from the imager.exe.
- Wait until the `IPC_EVENT_FRAME_INIT` bit is set in the return value of the `GetIPCState` call. If the event was occurred create all objects you need to process the data from the video frames.
- Call repeatedly the `GetFrame` function to retrieve new frames.
- Call all get functions that retrieves temperature values you need repeatedly.
- ...
- If an `OnConfigChanged` event occurs read all configuration data again.
- ...
- If an `OnServerStopped` event occurs, release all your objects and call the `ReleaseImagerIPC` function. Then reinitialize the IPC.
- ...
- ...
- Call the `ReleaseImagerIPC` function before closing your application.

6.2 B. Data type definition for this document

Data type	Description
char	8 bit signed integer
short	16 bit signed integer
int	32 bit signed integer
long	32 bit signed integer
__int64	64 bit signed integer
unsigned short	16 bit unsigned integer
unsigned long	32 bit unsigned integer
wchar_t	16 bit (wide character)
HRESULT	32 bit signed integer
Float	32 bit single precision floating point, IEEE 754
POINT	struct {long x; long y;}

FrameMetadata structure:

Data type	Description
unsigned short Size	Size of this structure
unsigned int Counter	Frame counter
unsigned int CounterHW	Hardware frame counter
long long Timestamp	Time stamp in 1/10000000-sec. units
long long TimestampMedia	reserved
TFlagState FlagState	Flag state , TFlagState is an enum type with the values: fsFlagOpen, fsFlagClose, fsFlagOpening, fsFlagClosing, fsError
float TempChip	Chip temperature
float TempFlag	Flag temperature
float TempBox	Box temperature
WORD PIFin[2]	Value of analogue and digital PIFin - DI = Bit15 of PIFIn[0] - AI1 = Bit9 ... Bit0 of PIFin[0] (0 = 0V, 1000 = 10V) - AI2 = Bit9 ... Bit0 of PIFin[1] (0 = 0V, 1000 = 10V)

VideoFormat structure:

Data type	Description
int WidthIR	Frame width of IR channel
int HeightIR	Frame height of IR channel
int FramerateIR	Frame rate of IR channel
int WidthVisible	Frame width of visible channel
int HeightVisible	Frame height of visible channel
int FramerateVisible	Frame rate of visible channel

IR arranging structure:

Data type	Description
TRotationMode Rotation	Rotation mode, TRotationMode is an enum type with the values: rmOff = image rotation off rmCW90 = clockwise 90 degrees rmACW90 = anti-clockwise 90 degrees rmCW180 = clockwise 180 degrees rmCWH = clockwise horizontal diagonal rmCWV = clockwise vertical diagonal rmACWH = anti-clockwise horizontal diagonal rmACWV = anti-clockwise vertical diagonal rmUser = user defined
float RotationAngle	Angle of rotation if Rotation is rmUser
BOOL Zoom	True if zoom is enabled
RECT ZoomRect	The zoom rectangle

Measure area structure:

Data type	Description
MeasureAreaShape Shape	Area shape, MeasureAreaShape is an enum type with the values: masOff= shape is off masMP1x1 = area of 1x1 pixel masMP3x3 = area of 3x3 pixel masMP5x5 = area of 5x5 pixel masUserDefRect = user defined rectangle masEllipse = user defined ellipse masPolygon = user defined polygon masCurve = user defined spline polygon
MeasureAreaMode Mode	Area mode, MeasureAreaMode is an enum type with the values: mamMin = minimum value of area mamMax = maximum value of area mamAvg = average value of area mamDist = percentage of pixels within temp range (min, max)
BOOL BindToTemperatureProfile	Bind measure area to temperature profile.
BOOL UseEmissivity	Use alternative emissivity for the measure area
float Emissivity	Alternative emissivity [0.0 … 1.0]
BOOL ShowInDigDispGroup	Show in digital display group
float distMin, distMax	Minimum and maximum temperature value for MeasureAreaMode::mamDist
POINT Location	Central location of measure area in pixel
SIZE Size	Size of the measure area. Only used for masUserDefRect and masEllipse
BOOL IsHotSpot,IsColdSpot	Indicates if measure area is coldspot/hotspot

6.3 C. Upgrading a project from Version 1 to Version 2

There are a view changes for the programmer:

- Undecorated function names when importing dll functions:
  `SetFlag` instead of `_SetFlag@4`
- Setting the count of used imagers before initialization:
  e.g.: `SetImagerIPCCount(2)`
- Using the imager index as first parameter in any function:
  `SetFlag(WORD index)` instead of `SetFlag(void)`

Berlin, 02/22/2017