Documentation - A Python 2/3 library for ifm efector O2x5xx 2D sensors

Features

PCIC V3 client for result data transfer

Prerequisites

Usage with Python 2 requires the 'future' package

Installation

Install the package with following command:

$ python setup.py install

Examples

--------

For a quick start, go to the examples folder and run

$ python image\_viewer.py 192.168.0.69

with your device's IP address to create an example application and

$ python image\_viewer.py 192.168.0.69

to view the image data coming from the camera (requires matplotlib)

Usage

-----

### RPC client

The library provides all XMLRPC objects as mentioned in the camera's XMLRPC

manual. The entry point is the RPC main object which can be created with e.g.

camera = o3d3xx.Device("192.168.0.69")

RPC calls can be performed e.g. like this

version\_info = camera.getSWVersion()

These are the different RPC objects and the possibilities how they can be

acquired:

\* Session object

- `session = camera.requestSession()`

This is different from the regular RPC call which only returns the

session ID. The session ID can be retrieved from `camera.sessionID`

- `session = camera.session` (only valid after a `requestSession()` call)

\* EditMode object

- `edit = session.startEdit()` which is equivalent to

- `edit = session.setOperatingMode(1)` (again, this is different from the

pure RPC call)

- `edit = session.edit` (only valid after a `startEdit()` call)

\* DeviceConfig object

- `device = edit.device`

\* NetworkConfig object

- `network = edit.network`

\* ApplicationConfig object

- `application = edit.editApplication(1)` (different from the pure RPC)

- `application = edit.application` (only valid after a `editApplication()`

call)

\* ImagerConfig object

- `imagerConfig = application.imagerConfig`

\* Spatial filter configuration object

- `spatialFilter = application.spatialFilter`

\* Temporal filter configuration object

- `temporalFilter = application.temporalFilter`

### PCIC client

The library currently provides three basic clients:

\* A simple PCIC V3 client

- Create it with `pcic = o3d3xx.PCICV3Client("192.168.0.69", 50010)`

providing the device's address and PCIC port.

- Send PCIC commands with e.g. `answer = pcic.sendCommand("G?")`. All

asnychronous PCIC messages are discarded while waiting for the answer

to the command.

- Read back the next PCIC for a particular ticket number. This can be used

to read asynchronously sent results (ticket number "0000"):

`answer = pcic.readAnswer("0000")`

- Read back any answer coming from the device:

`ticket, answer = pcic.readNextAnswer()`

\* A PCIC client supporting arbitrary formatting

- Define an instance of `o3d3xx.PCICFormat` to describe which data you

want to retrieve

- Most common use case: retrieval of binary data. For example, if you want

to receive amplitude and radial distance images, use the format

`o3d3xx.PCICFormat.blobs("amplitude\_image", "distance\_image")`.

- Create the client with

`pcic = o3d3xx.FormatClient("192.168.0.69", format)` with `format` being

the `PCICFormat` instance described above.

- When `result = pcic.readNextFrame()` returns for the given example,

`result` is a dictionary containing image arrays for the keys

`amplitude\_image` and `distance\_image`.

\* A PCIC client for asynchronous image retrieval

- Deprecated, please use the `o3d3xx.FormatClient` instead, it uses much

less network bandwidth if you are not interested in all available image

types.

- Create it with `pcic = o3d3xx.ImageClient("192.168.0.69", 50010)`.

- It configures a PCIC connection to receive all image types.

- Read back the next result (a dictionary containing all the images)

with `result = pcic.readNextFrame()`

### PCIC interface (ifmVisionAssistant)

Following structure in the process interface is required for multiple and

nested records:

\* supported interface structure

- records require a special delimiter at the end like seen in below

example. default delimiters `|` and `\*` can be parsed without any

changes in library.

- multiple or nested records between two blob images

- multiple models and ROIs in application which yield cascaded model

records in result.

- blob image(s) at the start or between the result

star ; X Image ; models Normalized amplitude image ; stop

| |

ID ; ROIs; Value of SP1 ; Value of SP2 | <-- default delimiter for models

| |

ID ; Process value ; Status \* <-- default delimiter for ROIs

\* restrictions

- single blob image at the end of multiple or nested records

- positional swap of delimiters `|` and `\*` in process interface

Contributing

------------

o2x5xx-python is also available at

[Github](https://github.com/ifm/o3d3xx-python)

Smoke-Tests

------------

1.22.9009

O2D5xx | O2I5xx | tested function | comment

-------- | -------- | -------- | --------

x | x | activate\_application | -

x | x | application\_list | -

x | x | upload\_pcic\_output\_configuration | -

x | x | retrieve\_current\_process\_interface\_configuration | -

x | x | request\_current\_error\_state | -

x | x | request\_current\_error\_state\_decoded | -

x | x | gated\_software\_trigger\_on\_or\_off | -

x | x | request\_device\_information | -

x | x | return\_a\_list\_of\_available\_commands | -

o | o | request\_last\_image\_taken | uncompressed images possible? answer is always !

x | x | request\_last\_image\_taken\_decoded | -

x | x | overwrite\_data\_of\_a\_string | -

x | x | read\_string\_from\_defined\_container | -

x | x | return\_the\_current\_session\_id | -

x | x | set\_logic\_state\_of\_a\_id | -

o | o | request\_state\_of\_a\_id | Reading io states without linked element is not working (O3D3xx is working)

x | x | turn\_process\_interface\_output\_on\_or\_off | -

x | x | request\_current\_decoding\_statistics | -

x | x | execute\_asynchronous\_trigger | -

x | x | execute\_synchronous\_trigger | -

x | x | set\_current\_protocol\_version | -

x | x | request\_current\_protocol\_version | -

x | x | turn\_state\_of\_view\_indicator\_on\_or\_off | -