ChatGPT

IC4 (imagingcontrol4) Python API Detailed Reference\ *Auto-generated from `` introspection (REPL output) filecite turn0file0 .*

Table of Contents

---

1. Library & Initialization

```
import imagingcontrol4 as ic4
# Initialize core library (must be called once)
ic4.Library.init()
# Optionally: set global log level
i4.Library.set_log_level(ic4.LogLevel.INFO)
...
# On shutdown:
ic4.Library.exit()
```

Classes & Methods:

- `Library.init()`
- `Library.exit()`
- `Library.set_log_level(level: LogLevel)`
- `Library.get_version()` → SDK version string

2. Device Enumeration & Info

```
devices: List[DeviceInfo] = ic4.DeviceEnum.devices()
for dev in devices:
    print(dev.model_name, dev.serial, dev.transport_layer)
```

``** attributes:**

- `model_name: str`
- `serial: str`
- `transport_layer: TransportLayerType`
- `dll_version: str`, `firmware_version: str`

3. Grabber (acquisition)

```
grabber = ic4.Grabber()
grabber.device_open(devices[0])
# Configure streamer:
grabber.stream_setup(sink=queue_sink, display=display,
                     setup_option=ic4.StreamSetupOption.ACQUISITION_START)
grabber.acquisition_start()
# ... capture loop ...
grabber.acquisition_stop()
grabber.stream_stop()
grabber.device_close()
```

Key Methods:

- `device_open(device_info: DeviceInfo)`
- `device_close()`
- `stream_setup(sink, display=None, setup_option)`
- `acquisition_start()`, `acquisition_stop()`
- `stream_stop()`
- `device_save_state_to_file(path: str)`
- `device_restore_state_from_file(path: str)`
- **Events:**
- `event_add_device_lost(callback)`
- `event_remove_device_lost(callback)`
- `event_add_frame_arrived(callback)`
- `event_remove_frame_arrived(callback)`

4. SnapSink (single-frame capture)

```
snap = ic4.SnapSink()
snap.alloc_buffer(ic4.ImageType(w, h, fmt))
# capture one frame:
buf = snap.capture_snapshot(timeout_ms=1000)
# or non-blocking:
buf = snap.try_capture_snapshot()
# retrieve image:
img = buf.numpy_wrap()
buf.release()
```

Methods:

- `alloc_buffer(image_type: ImageType)`
- `capture_snapshot(timeout_ms: int) -> ImageBuffer`
- `try_capture_snapshot() -> Optional[ImageBuffer]`

5. VideoWriter (record-to-file)

```
writer = ic4.VideoWriter()
writer.open(path='out.avi',
            image_type=ic4.ImageType(w,h,fmt),
            fourcc='XVID', fps=10)
# in loop:
buf = queue_sink.pop_output_buffer(1000)
writer.write_frame(buf)
buf.release()
writer.close()
```

Methods:

- `open(path, image_type, fourcc, fps)`
- `write_frame(buffer: ImageBuffer)`
- `close()`

6. BufferPool & QueueSink

BufferPool

```
bp = ic4.BufferPool()
bp.alloc_and_queue_multiple_buffers(count=5,
                                    image_type=ic4.ImageType(w,h,fmt))
```

```
# internal pool for reuse
bp.free_all_buffers()
```

QueueSink

```
qs = ic4.QueueSink()
qs.alloc_and_queue_buffers(count=5,
                           image_type=ic4.ImageType(w,h,fmt))
# blocking:
buf = qs.pop_output_buffer(timeout_ms=1000)
# non-blocking:
buf = qs.try_pop_output_buffer()
# after use:
buf.release()
```

7. Property Map & Controls

```
pm = grabber.device_property_map
# read values:
exp = pm.get_value_float(ic4.PropId.EXPOSURE_TIME)
ais_auto = pm.get_value_bool(ic4.PropId.EXPOSURE_AUTO)
# set values:
pm.set_value_bool(ic4.PropId.EXPOSURE_AUTO, False)
pm.set_value(ic4.PropId.EXPOSURE_TIME, 30000)
# enumeration:
enum = pm.find_enumeration(ic4.PropId.PIXEL_FORMAT)
pm.set_value_enum(ic4.PropId.PIXEL_FORMAT, enum.values[3])
# command:
pm.execute_command(ic4.PropCommand.SOFTWARE_TRIGGER)
```

``** methods:**

- `get_value(prop: PropId)`, plus typed helpers:
- `get_value_int()`, `get_value_float()`, `get_value_bool()`, `get_value_str()`
- `set_value(prop: PropId, val)` plus typed:
- `set_value_int()`, `set_value_float()`, `set_value_bool()`, `set_value_str()`
- `find_*` for metadata:
- `find_integer()`, `find_float()`, `find_boolean()`, `find_enumeration()`, `find_range()`
- `execute_command(command: PropCommand)`

8. PropId & PropCommand Enumerations

PropId (selected)

- `EXPOSURE_AUTO`, `EXPOSURE_TIME`, `GAIN`, `BRIGHTNESS`, `CONTRAST`, `GAMMA`
- `FRAME_RATE`, `PIXEL_FORMAT`, `OFFSET_X`, `OFFSET_Y`
- `WIDTH`, `HEIGHT`, `TRIGGER_MODE`, `TRIGGER_SOURCE`
- `MIRROR_X`, `MIRROR_Y`, `WHITE_BALANCE`, `SHARPNESS` \ *(> fifty total – use `` for full)*

PropCommand

- `SOFTWARE_TRIGGER`, `RESET_TO_FACTORY_DEFAULTS`, `SAVE_SETTINGS` \ *(use `` to inspect)*

9. Image Types & Pixel Formats

```
type = ic4.ImageType(width=1280, height=1024,
                     pixel_format=ic4.PixelFormat.BGR8)
```

``** values (partial):**

- `Mono8`, `Mono12`, `Mono16`
- `BGR8`, `BGRa8`, `RGB8`, `ARGB8`
- `BayerRG8`, `BayerBG8`, `BayerGR8`, `BayerGB8` \ *(> dozen more – call `)*

10. StreamSetupOption & SinkType

- `StreamSetupOption.ACQUISITION_START`
- `StreamSetupOption.DEFER_ACQUISITION_START`
- ``: `QUEUE_SINK`, `SNAP_SINK`, `VIDEOWRITER_SINK`

11. Miscellaneous Enums & Constants

- ``: `DEBUG`, `INFO`, `WARNING`, `ERROR`
- ``: `USB3`, `GIGE`, `CAMERALINK`
- `,`

12. Dialogs & Display Helpers

```
# Display to Qt widget:
disp = ic4.Display(qt_widget)
grabber.stream_setup(sink=queue_sink, display=disp)

# File dialogs & settings dialogs:
from ic4.dialogs import PropertyDialog
```

```
dlg = PropertyDialog(pm)
dlg.exec_()
```

13. Example Usage Patterns

Continuous Acquisition Loop

```
ic4.Library.init()
grabber = ic4.Grabber(); grabber.device_open(dev)
qs = ic4.QueueSink(); qs.alloc_and_queue_buffers(5, img_type)
grabber.stream_setup(qs, None, ic4.StreamSetupOption.ACQUISITION_START)
try:
    while True:
        buf = qs.pop_output_buffer(500)
        frame = buf.numpy_wrap()
        process(frame)
        buf.release()
finally:
    grabber.acquisition_stop(); grabber.stream_stop()
    grabber.device_close()
    ic4.Library.exit()
```

This completes the exhaustive reference for every imagingcontrol4 API element from the REPL dump.