

edelkrone SDK

User Guide

Contents

1 Getting Started

2 Connecting

2.1 edelkrone.connect()

2.2 Finding Units in Range: edelkrone.discover()

2.3 Layouts

2.4 Ending the Connection

3 The Device

3.1 Identity: device.info

3.2 State: device.status

4 Moving

4.1 Joystick Control: device.move.joystick()

4.2 Stopping

5 Poses

6 Focus

7 Settings

8 Tripod

9 Errors

10 Diagnostic Log

11 API Summary

1 Getting Started

The edelkrone SDK controls edelkrone motion control equipment from Python over Bluetooth Low Energy. It supports heads, sliders, jibs, dollies, focus motors and tripods, used on their own or combined in a layout. Release 0.1 covers connecting, reading the device's identity and state, joystick control, stored poses, focus, settings and the tripod.

```
pip install edelkrone
```

Python	3.11 or newer.
Platforms	macOS, Windows and Linux. Bluetooth must be enabled. On macOS, the terminal or application requires Bluetooth permission (System Settings → Privacy & Security → Bluetooth); without it, no units are found.
Dependencies	<code>bleak</code> , the Bluetooth package, is installed automatically.
Included in the package	This guide, in <code>edelkrone/doc/</code> inside the installed package, and two example programs: <code>python -m edelkrone.examples.read_device</code> and <code>python -m edelkrone.examples.store_and_return</code> .
Version	<code>edelkrone.__version__</code>

- Figure 1: Installation -

A first program:

```
import edelkrone

for unit in edelkrone.discover():
    print(unit.to_json())

with edelkrone.connect(name="HeadPLUS V3") as device:
    print(device.info.to_json())
    print(device.status.to_json())

    device.move.joystick(pan=0.3)
moving
    device.move.stop()

    device.poses.store(0)
    result = device.poses.move_to(0)
    if not result.is_success:
        print(result.reason)
```

The SDK follows three principles:

- **Calls are synchronous.** Each call returns when its operation has completed. No event loop is required. In asynchronous code, run a call in a worker thread, for example `await asyncio.to_thread(device.poses.move_to, 0)`.

- **The SDK polls the device.** After `connect()`, the SDK requests a new reading four times per second, and `device.status` always holds the latest one.
- **Refusals are returned; faults are raised.** A request the device declines, such as a move to an empty slot, returns a result that states the reason. A failure of the connection or of a command raises an exception (9).

Not included in release 0.1. Targets and point mode, driving to coordinates, creating a dolly path and reading the dolly position, and timelapse. These functions are planned for later releases.

2 Connecting

2.1 edelkrone.connect()

`connect()` finds the unit, connects to it, reads its identity and a first reading, joins its modules if required, starts polling and returns a `Device`. If any of these steps fails, it raises an exception that identifies the step (9).

```
device = edelkrone.connect(name="HeadPLUS V3", mac="A4:C1:38:5E:7B:2D", timeout=20.0)
```

Argument	Default	Description
<code>name</code>	None	The product name printed on the unit. Case and spaces are ignored: "HeadPLUS V3", "headplus v3" and "HEADPLUSV3" are equivalent.
<code>mac</code>	None	The unit's Bluetooth address, in any common notation. The address does not change when the unit is restarted.
<code>with_modules</code>	None	For units that have not been used together before: the addresses of the modules to join to this unit (2.3).
<code>timeout</code>	20 s	How long to search for the unit before raising <code>not_found</code> .
<code>transport</code>	The desktop Bluetooth radio	A <code>Transport</code> held by the program, so that several calls share one radio. A radio opened by <code>connect()</code> is closed by <code>disconnect()</code> .
<code>config</code>	Default values	A <code>SessionConfig</code> that sets timeouts and the polling interval.
<code>allow_narrowing</code>	False	Set to <code>True</code> to connect to a unit on its own while it is configured to drive modules. By default this is refused, because poses stored in the reduced layout would not include the modules' axes.

- Figure 2: `connect()` arguments -

Specify a name, an address, or both; when both are given, both must match. When neither is given, the SDK connects to the only unit in range. If more than one unit is in range, the call is refused and the message lists the units found.

2.2 Finding Units in Range: `edelkrone.discover()`

`edelkrone.discover(seconds=8.0)` scans for the given time and returns every unit found as a list of `Seen`, ordered by signal strength. It accepts the same `transport` argument as `connect()`.

```
for unit in edelkrone.discover():
    print(unit.to_json())

device = edelkrone.connect(mac=edelkrone.discover()[0].mac)
```

```
{
  "name": "HeadPLUS V3",
  "mac": "A4:C1:38:5E:7B:2D",
  "signal": -51,
  "layout": "pan_tilt_and_slide",
  "modules": ["SliderPLUS V6"],
  "recognised": true
}
```

Field	Description
<code>name</code> , <code>mac</code>	The product name and the Bluetooth address, as accepted by <code>connect()</code> .
<code>signal</code>	The received signal strength in dBm. Values closer to zero indicate a stronger signal. The list is sorted by this field.
<code>layout</code> , <code>modules</code>	The layout the unit advertises and the product names of the modules in it. Empty for a unit on its own.
<code>recognised</code>	Whether the SDK supports this combination of units. An unrecognised combination is listed but cannot be connected to.

- Figure 3: Seen -

Connected units are not listed. A unit does not advertise while it is connected to another program or to the edelkrone application on a phone. If a powered unit in range is not listed, close any other connection to it.

2.3 Layouts

Modules mounted on a head, such as a slider, a jib or a dolly, are controlled through the head. The program connects to the head, which relays commands to its modules.

`device.info.modules` lists the modules and `device.info.layout` names the layout, for example `PAN_TILT_AND_SLIDE` .

The two three-piece products are also layouts, and in both the program connects to the head:

- **Krone X:** a HeadPLUS V3 with a JibPLUS and a DollyPLUS (`PAN_TILT_DOLLY_AND_JIB_PLUS`).
- **Krone S:** a HeadPLUS V3 with a SliderPLUS and a DollyPLUS (`PAN_TILT_DOLLY_AND_SLIDER`).

Units that have been used together before advertise their layout, and `connect()` requires no further arguments. Units that have not been used together must be joined once. Pass the module addresses in `with_modules` ; the SDK joins them by cable or by radio, as the hardware requires:

```
device = edelkrone.connect(mac="A4:C1:38:5E:7B:2D", with_modules=["C8:F0:9E:11:22:33"])
```

A combination of units that the SDK does not support is refused with `NotSupported` , and the message names the units.

2.4 Ending the Connection

`device.disconnect()` stops polling and closes the connection. If `connect()` opened the radio, it closes the radio as well. Calling it more than once has no further effect. A `Device` is a context manager, so a `with` block closes the connection in all cases:

```
with edelkrone.connect(name="SliderPLUS V6") as device:
    device.move.joystick(slide=0.2)
    time.sleep(1.0)
    device.move.stop()
```

Stop the equipment before disconnecting. Disconnecting does not stop moving equipment. Call `device.move.stop()` first, and place both calls in a `finally` block so that the equipment stops if the program fails.

3 The Device

`connect()` returns a `Device`. Its `info` and `status` describe the unit, and four groups provide the operations: `move`, `poses`, `focus` and `settings`. The `Device` itself manages the connection:

Member	Description
<code>info</code>	The unit's identity, read when connecting (3.1).
<code>status</code>	The latest reading (3.2).
<code>status_updates</code>	A stream that delivers each new reading; see <code>subscribe()</code> in 3.2.
<code>refresh()</code>	Requests the identity, position and state from the device immediately.
<code>poll_position(every=...)</code>	Keeps <code>status.pose</code> up to date. The periodic reading reports the battery and the activity, but not the position; without position polling, <code>status.pose</code> holds the value from the last <code>refresh()</code> . <code>poll_position()</code> requests the position every 250 ms, <code>poll_position(every=1.0)</code> once per second, and <code>poll_position(every=None)</code> stops. Disabled by default, because it doubles the radio traffic.
<code>is_connected</code>	Whether the connection is currently open.
<code>connection_changes</code>	A stream of <code>True</code> and <code>False</code> values as the connection opens and closes.
<code>disconnect()</code>	Closes the connection. A <code>with</code> block calls it on exit.

- Figure 4: Device -

3.1 Identity: `device.info`

An `Info` object, read when connecting and unchanged until the connection is closed. `to_dict()` returns it as a dictionary and `to_json()` as a JSON object; the field names match the Python attributes. All readings and results in the SDK provide the same two methods.

```
{
  "name": "HeadPLUS V3",
  "mac": "A4:C1:38:5E:7B:2D",
  "firmware": "0.31",
  "radio_firmware": "0.34",
  "firmware_ok": true,
  "layout": "pan_tilt_and_slide",
  "modules": ["SliderPLUS V6"],
  "axes": ["pan", "slide", "tilt"],
  "capabilities": ["focus", "manual_drive", "position", "settings", "stored_pose"]
}
```

Field	Description
<code>name</code> , <code>mac</code>	The product name printed on the unit and its Bluetooth address.
<code>firmware</code> , <code>radio_firmware</code> , <code>firmware_ok</code>	The firmware versions of the device and of its radio module, and whether both meet the minimum versions this SDK supports. If <code>firmware_ok</code> is <code>false</code> , update the firmware with the edelkrone application before continuing.
<code>layout</code>	The modules the unit drives and their arrangement. <code>null</code> if the unit reports no layout.
<code>modules</code>	The product names of the other units in the layout.
<code>axes</code>	The axes the layout can drive or read, as <code>RigAxis</code> values: <code>PAN</code> , <code>TILT</code> , <code>SLIDE</code> , <code>JIB_PAN</code> and <code>JIB_TILT</code> , shown in lower case in JSON. A head on its own has two axes; a head with a slider has three.
<code>capabilities</code>	The functions the SDK supports on this model, as <code>Capability</code> values: <code>POSITION</code> , <code>STORED_POSE</code> , <code>MANUAL_DRIVE</code> , <code>FOCUS</code> , <code>SETTINGS</code> and <code>TRIPOD_LEVEL</code> , shown as <code>position</code> , <code>stored_pose</code> , <code>manual_drive</code> , <code>focus</code> , <code>settings</code> and <code>tripod_level</code> in JSON. A function not listed here is refused with <code>NotSupported</code> . To check before calling, use for example <code>Capability.FOCUS</code> in <code>device.info.capabilities</code> .

- Figure 5: Info -

3.2 State: `device.status`

A `DeviceStatus` object holding the latest reading, in degrees, centimetres, percent and seconds. It is held in memory, so reading it does not communicate with the device. Polling updates it four times per second, and `device.refresh()` requests a reading immediately. Any field may be `null` , which always means that the device does not report the value. For example, a slider has no pan angle and a tripod reports no battery level.

```
{
  "model": "HeadPLUS V3",
  "at": 1789822822.031,
  "activity": "idle",
  "pose": {"pan_deg": 89.96, "tilt_deg": 0.0, "slide_cm": 12.4, "focus_cm": null,
  "jib_pan_deg": null, "jib_tilt_deg": null},
  "slide_fraction": 0.31,
  "battery_percent": 82.0,
  "is_calibrated": true,
  "has_focus_motor": false,
  "layout": "pan_tilt_and_slide",
  "stored_pose_slots": [0, 3],
```

```

"modules_responding": true,
"move_progress": null,
"position_is_trusted": true,
"has_status_frame": true,
"tilt_deg": 1.2,
"is_level": true,
"orientation_warning": false
}

```

Field	Description
<code>model</code> , <code>at</code>	The product name of the unit that sent the reading, and the time of the reading in seconds.
<code>activity</code>	An <code>ActivityState</code> : <code>IDLE</code> , <code>MOVING</code> , <code>TRANSITIONING</code> or <code>TRACKING_TARGET</code> , shown as <code>idle</code> , <code>moving</code> , <code>transitioning</code> and <code>tracking_target</code> in JSON. The last two occur only while the edelkrone application is tracking a target with the head. <code>null</code> before the first reading. <code>is_moving</code> provides the same information as a Boolean.
<code>pose</code> , <code>position_is_trusted</code>	The position of the equipment as a <code>RigPose</code> : pan and tilt in degrees, the slider position in centimetres and the jib axes in degrees. Axes not present in the layout are <code>null</code> . <code>position_is_trusted</code> is <code>false</code> when the position was taken from a stored slot rather than from a live reading.
<code>slide_fraction</code>	The slider or jib position as a fraction of its travel, from 0 to 1. Unlike the value in centimetres, it is included in every reading. <code>null</code> until the slider has been referenced.
<code>battery_percent</code>	The battery level. A tripod is powered from the mains and reports none.
<code>is_calibrated</code>	For a slider, whether it has been referenced; for a head, whether its level reference is set. <code>null</code> for models that require neither. A slider that is not referenced refuses planned moves and sounds its buzzer; reference it with the edelkrone application.
<code>has_focus_motor</code>	Whether a focus motor is fitted, as reported by the device.
<code>layout</code>	The layout most recently reported by the device.
<code>stored_pose_slots</code>	The slots that contain a stored pose.
<code>modules_responding</code>	Whether the modules in the layout respond to the unit. A module that advertises is powered, but not necessarily responding.
<code>move_progress</code>	The progress of a planned move, from 0 to 1.
<code>has_status_frame</code>	Whether a complete reading has been received. Until then, the other fields are <code>null</code> . The property <code>is_known</code> returns the same value.

Field	Description
<code>tilt_deg</code> , <code>is_level</code> , <code>orientation_warning</code>	The unit's inclination from level in degrees, whether it is within 15°, and the device's orientation warning. Reported by PRO heads and by sliders; <code>null</code> on units without an inclination sensor.

- Figure 6: DeviceStatus -

To receive each reading as it arrives, subscribe to `status_updates` :

```
for status in device.status_updates.subscribe(max_pending=8):  
    print(status.pose.pan_deg, status.battery_percent)
```

A subscription is a queue that the program reads at its own pace. `max_pending` sets how many readings it holds (256 by default); when it is full, the oldest reading is discarded. Read the subscription on a separate thread and do not send commands from within it: the SDK refuses a command sent from a subscription callback, because the reply would have to be delivered on the thread that is waiting for it.

4 Moving

4.1 Joystick Control: `device.move.joystick()`

`joystick()` sets a speed for each axis, from -1 to 1 as a fraction of the maximum speed. It does not set a target position. **The command must be repeated** approximately every 100 ms for as long as the equipment is to move; when commands stop arriving, the device decelerates and stops after approximately one second. Call `stop()` to end the movement.

```
deadline = time.monotonic() + 2.0
try:
    while time.monotonic() < deadline:
        device.move.joystick(pan=0.3, tilt=-0.1)
        time.sleep(0.1)
finally:
    device.move.stop()
```

Axis	Movement
<code>pan</code> , <code>tilt</code>	The head.
<code>slide</code>	The slider carriage. In a jib layout, the height of the jib arm.
<code>jib_pan</code> , <code>jib_tilt</code>	The rotation and the tilt of the jib arm.
<code>linear</code> , <code>angular</code>	A dolly: forward and backward movement, and rotation in place. A dolly on its own moves freely; a dolly in a layout moves only along an existing path, created with the edelkrone application.

- Figure 7: Joystick axes -

Axes not present in the layout are ignored, so passing zero for them has no effect. Values outside ± 1 are limited to ± 1 . A value that is not a number is refused.

4.2 Stopping

Member	Description
<code>stop(settle=2.0)</code>	Stops all movement: joystick control and a move to a stored pose alike. The stop is sent at once, ahead of any other pending command; every axis is then set to zero, and the call waits up to <code>settle</code> seconds for the device to report that it is stationary. It does not raise exceptions, so it can be used in a <code>finally</code> block: if the connection was lost during the movement, no second exception conceals the first.
<code>is_moving</code>	Whether the latest reading reports the device as moving. <code>None</code> before the first reading, which differs from <code>False</code> .

Member	Description
<code>wait_until_idle(timeout=60.0)</code>	Waits until the device reports that it is idle; returns <code>False</code> on timeout. <code>poses.move_to()</code> already waits in this way.

- Figure 8: device.move -

Polling pauses during joystick control. While any joystick axis is non-zero, the SDK pauses polling to keep the joystick responsive. During this time `status_updates` delivers no readings and `device.status` keeps the last one. Polling resumes after `move.stop()` or a joystick command with all axes at zero.

5 Poses

A pose is the position of every axis in the layout, stored in a numbered slot on the device.

`move_to()` returns the equipment to a stored pose; the device plans the motion so that all axes arrive at the same time.

```
device.poses.store(0)
result = device.poses.move_to(0, style=MoveStyle(speed=0.15, acceleration=0.4))
print(result.to_json())          # {"accepted": true, "completed": true, "duration": 3.4,
"reason": null}
```

Member	Description
<code>store(slot)</code>	Stores the current pose in <code>slot</code> and confirms from the next reading that the device has stored it. Returns <code>False</code> if the device did not store it, for example for a slot the model does not have, or on a dolly, which stores its positions on its path instead. A HeadPLUS V3 provides slots 0 to 11.
<code>stored</code>	The slots that contain a pose, as last reported by the device. To test a slot: <code>slot in device.poses.stored</code> .
<code>clear(slot)</code> , <code>clear_all()</code>	Deletes the pose in one slot, or in all slots.
<code>move_to(slot, style=None, start=None, timeout=60.0, verify=True)</code>	Moves to the stored pose and waits until the equipment is stationary. With <code>verify</code> enabled, the SDK compares the start and end positions, and a move in which the equipment did not move is not reported as successful. <code>start</code> names a slot to plan the move from, instead of the current position. A refusal is returned as a result, not raised.
<code>start_move_to(slot, style=None)</code>	Sends the move and returns immediately, for programs that control several devices. <code>move.wait_until_idle()</code> waits for the move to finish.

- Figure 9: device.poses -

The result of a move is a `MoveOutcome` :

Field	Description
<code>accepted</code>	The device accepted the command.
<code>completed</code>	The device reported the move as finished and the equipment as stationary.
<code>duration</code>	The duration of the move in seconds.
<code>reason</code>	If the move did not take place, the reason, for example an empty slot, a slider that is not referenced, or equipment that did not move. <code>None</code> on success.
<code>is_success</code>	<code>True</code> when both <code>accepted</code> and <code>completed</code> are true.

- Figure 10: MoveOutcome -

A `MoveStyle` defines how a move is performed. Its two values range from 0 to 1 as fractions of the device's maximum:

```
MoveStyle(speed=0.15, acceleration=0.4)           # slow, with long acceleration and
deceleration
MoveStyle(speed=1.0, acceleration=0.01)           # the fastest possible move
MoveStyle()                                       # speed 0.5, acceleration 0.25: the
default of move_to()
MoveStyle(speed=0.3, acceleration=0.25, loop=True) # moves back and forth between two poses
until move.stop()
```

`speed` sets the speed. `acceleration` sets the share of the move used for acceleration at the start and deceleration at the end. `loop` repeats the move back and forth.

Short moves. If the equipment is within approximately 3° of the stored pose, the device may not start the planned move. In this case `move_to()` returns a result with `is_success` set to `False` and a reason stating that the equipment did not move. This is a property of the device and not a fault.

Axes stored in a pose. A pose stores only the axes of the current layout. If a head is connected without its modules, its poses contain pan and tilt only. Check `device.info.layout` before storing poses.

6 Focus

A head with a focus motor, or a focus module on its own, sets focus as a **percentage of the lens travel**, from 0 at one end to 100 at the other. The motor must first learn the lens: it sets its reference, finds both ends of the travel and measures the distance between them.

`setup()` performs these three steps. It must be called once per connection and turns the focus ring from one end to the other; the duration depends on the lens. Keep the focus ring clear while it runs.

```
device.focus.setup()           # once per connection; turns the focus ring
reached = device.focus.move_to_percent(50)
print(device.focus.percent())  # current position of the focus ring
```

Member	Description
<code>setup(timeout=120.0)</code>	Prepares the focus motor. It performs any missing steps in order: motor reference, end detection and travel measurement. A second call returns immediately. It must be called once per connection, and again after the unit has been restarted. Refused if no focus motor is fitted.
<code>move_to_percent(percent, tolerance=1.0)</code>	Moves to a percentage of the travel and returns the percentage reached. The device performs the movement and stops at the ends found by <code>setup()</code> .
<code>percent()</code>	The current position of the focus ring, as a percentage of the travel.

- Figure 11: device.focus -

`setup()` is required before other operations. On a unit that reports a focus motor, the following calls raise `CommandError · needs_focus_setup` until `setup()` has been called on the current connection: the joystick, all stored-pose operations, the settings and the tripod operations. The following calls are always available: all reads (`info`, `status`, `refresh()`, `poses.stored`), `move.stop()` and `move.wait_until_idle()`, `disconnect()` and `setup()`.

The reason is that a stored pose on such a unit includes the focus position, which is measured from the ends found by `setup()`. A pose stored or recalled before `setup()` would place the focus ring at an unintended position.

When `connect()` connects to a unit with a focus motor, it issues a `UserWarning` stating that `setup()` is required. To suppress it, use `warnings.filterwarnings("ignore", message=".*focus motor.*")`. Units without a focus motor are not affected.

Focus in centimetres, that is a distance to the subject, is not supported in this release.

7 Settings

Member	Description
<code>buzzer(muted=...)</code>	Mutes or unmutes the buzzer.

- Figure 12: device.settings -

The device does not acknowledge settings. The call returns once the command has been sent.

8 Tripod

A motorized tripod levels itself and raises or lowers its column. The two operations are reached through `device.settings` and are refused on units other than tripods.

```
device.settings.level_tripod()          # starts the automatic levelling

end = time.monotonic() + 3.0            # raises the column for 3 seconds
while time.monotonic() < end:
    device.settings.drive_tripod(40)
    time.sleep(0.2)
device.settings.drive_tripod(0)        # stops the column
```

Member	Description
<code>level_tripod()</code>	Starts the automatic levelling of the tripod.
<code>drive_tripod(relative_speed)</code>	Moves the column at a speed from -100 to 100: positive values extend it, negative values retract it, and 0 stops it. Values outside the range are limited to -100 and 100. It must be repeated every 100 to 200 ms for as long as the column is to move, in the same way as the joystick; a single call does not move the column.

- Figure 13: Tripod operations of `device.settings` -

The tripod does not acknowledge these commands. The calls return once the command has been sent.

9 Errors

The SDK distinguishes two kinds of failure. A **refusal**, such as an empty slot, a slider that is not referenced or a move that did not start, is returned as a result with `is_success` and a `reason`. A **fault** raises one of three exceptions, all subclasses of `edelkrone.Error`. Each carries a `reason` code, a `message` and a `cause` with technical details for support requests.

```
try:
    device = edelkrone.connect(name="HeadPLUS V3")
except edelkrone.ConnectionError as e:
    print(f"{e.reason}: {e.message}")          # not_found: HeadPLUS V3 was not seen in 20.0
s. Seen: ...
```

Exception and reason	Cause	Action
<code>ConnectionError · not_found</code>	The unit was not found within the timeout. It is switched off, out of range, or connected to another program or phone.	Check the power and close other connections to the unit. <code>discover()</code> lists the units in range.
<code>ConnectionError · ambiguous</code>	No name or address was given, and more than one unit is in range.	Specify the unit. The message lists the units found and their addresses.
<code>ConnectionError · no_radio</code>	The computer has no Bluetooth adapter, or the Bluetooth package is not installed.	Check the adapter. Reinstall with <code>pip install edelkrone</code> .
<code>ConnectionError · no_permission</code>	The operating system has not granted Bluetooth access to the program.	macOS: System Settings → Privacy & Security → Bluetooth.
<code>ConnectionError · timed_out</code>	The unit was found but did not respond while connecting (three attempts at three-second intervals), or did not respond to the first identity request.	Restart the unit and reduce the distance.
<code>ConnectionError · dropped</code>	The connection was lost.	Reconnect. Place <code>move.stop()</code> and <code>disconnect()</code> in a <code>finally</code> block.
<code>ConnectionError · wrong_device</code>	The unit connected but does not provide the services expected for its model, usually because of outdated firmware.	Check <code>info.firmware_ok</code> and update the firmware with the edelkrone application.
<code>ConnectionError · layout_refused</code>	The unit did not accept the layout: joining its modules failed, or it is configured with modules and was asked to operate alone.	Check that the modules are switched on. To operate the unit alone, pass <code>allow_narrowing=True</code> .
<code>CommandError · not_connected</code>	A command was sent after the connection was lost.	Reconnect.
<code>CommandError · closed</code>	A command was sent after <code>disconnect()</code> .	Connect again. A <code>Device</code> cannot be reused after it is closed.
<code>CommandError · no_reply</code>	The device did not respond to a command in time.	Retry once. If the error persists, restart the unit.
<code>CommandError · bad_reply</code>	The device sent an unexpected response.	Report the error with <code>info.firmware</code> and <code>cause</code> .

Exception and reason	Cause	Action
<code>CommandError</code> · <code>bad_argument</code>	An argument is invalid, for example not a number, a slot out of range or a speed of <code>nan</code> .	Correct the argument.
<code>CommandError</code> · <code>needs_focus_setup</code>	The unit reports a focus motor and <code>setup()</code> has not been called on this connection. Only calls that change the equipment raise it.	Call <code>device.focus.setup()</code> once per connection (6).
<code>NotSupported</code> · <code>unsupported</code>	The model or layout does not provide the function, for example a pose on a tripod or focus on a slider, or the units named in <code>with_modules</code> do not form a supported layout.	Check <code>device.info.capabilities</code> .

- Figure 14: Errors raised by the SDK -

Signal strength. A unit at the edge of the radio range can be found by `discover()` but may fail after connecting, with `dropped`, `not_connected` or `no_reply`. Layouts with modules are affected first. If connections are unreliable, reduce the distance to the unit.

10 Diagnostic Log

The SDK writes diagnostic messages to `EdelkroneLog`. The log is disabled by default. To enable it, assign a function to `EdelkroneLog.sink` and, optionally, set `EdelkroneLog.minimum_level` to an `EdelkroneLogLevel`: `TRACE`, `DEBUG`, `INFO` (the default), `WARNING` or `ERROR`.

```
from edelkrone import EdelkroneLog, EdelkroneLogLevel

EdelkroneLog.sink = lambda level, message: print(f"[{level.name}] {message}")
EdelkroneLog.minimum_level = EdelkroneLogLevel.DEBUG
```

A sink is an `EdelkroneLogSink`: a function that takes the level and the message. It may be called from several threads, must not raise exceptions and must return quickly. To forward messages to Python's `logging` module, call a `logging` logger from the sink.

11 API Summary

The following table lists every name exported by `import edelkrone` and the section that describes it.

Name	Description	Section
<code>connect()</code> , <code>discover()</code>	Connect to a unit; list the units in range.	2
<code>Device</code>	A connected unit, with its identity, state and four operation groups.	3
<code>Info</code> , <code>DeviceStatus</code> , <code>Seen</code>	The identity, a reading, and a unit found in a scan. Each can be exported as JSON.	3.1, 3.2, 2.2
<code>MoveControl</code> , <code>PoseController</code> , <code>FocusControl</code> , <code>SettingsControl</code>	The types of <code>device.move</code> , <code>device.poses</code> , <code>device.focus</code> and <code>device.settings</code> , for use in type hints. They are not constructed directly.	4, 5, 6, 7, 8
<code>MoveOutcome</code> , <code>MoveStyle</code>	The result of a pose move; the speed and acceleration of a move.	5
<code>ActivityState</code> , <code>Capability</code>	The activity of a device; a function a model supports.	3.2, 3.1
<code>RigPose</code> , <code>RigAxis</code> , <code>EdlSetup</code> , <code>DeviceFamily</code>	A position, an axis, a layout, and a device family (head, slider, jib, dolly, focus or tripod).	3.2, 3.1, 2.3
<code>SessionConfig</code> , <code>Transport</code>	The types accepted by <code>connect(config=...)</code> and <code>connect(transport=...)</code> .	2.1
<code>Error</code> , <code>ConnectionError</code> , <code>CommandError</code> , <code>NotSupported</code>	The base exception and the three fault types.	9
<code>EdelkroneLog</code> , <code>EdelkroneLogLevel</code> , <code>EdelkroneLogSink</code>	The diagnostic log, its levels and the sink type.	10
<code>__version__</code>	The installed version.	1

- Figure 15: Exported names -

This content is subject to change.

If you have any questions about this document, please contact edelkrone.

Copyright © 2026 edelkrone.