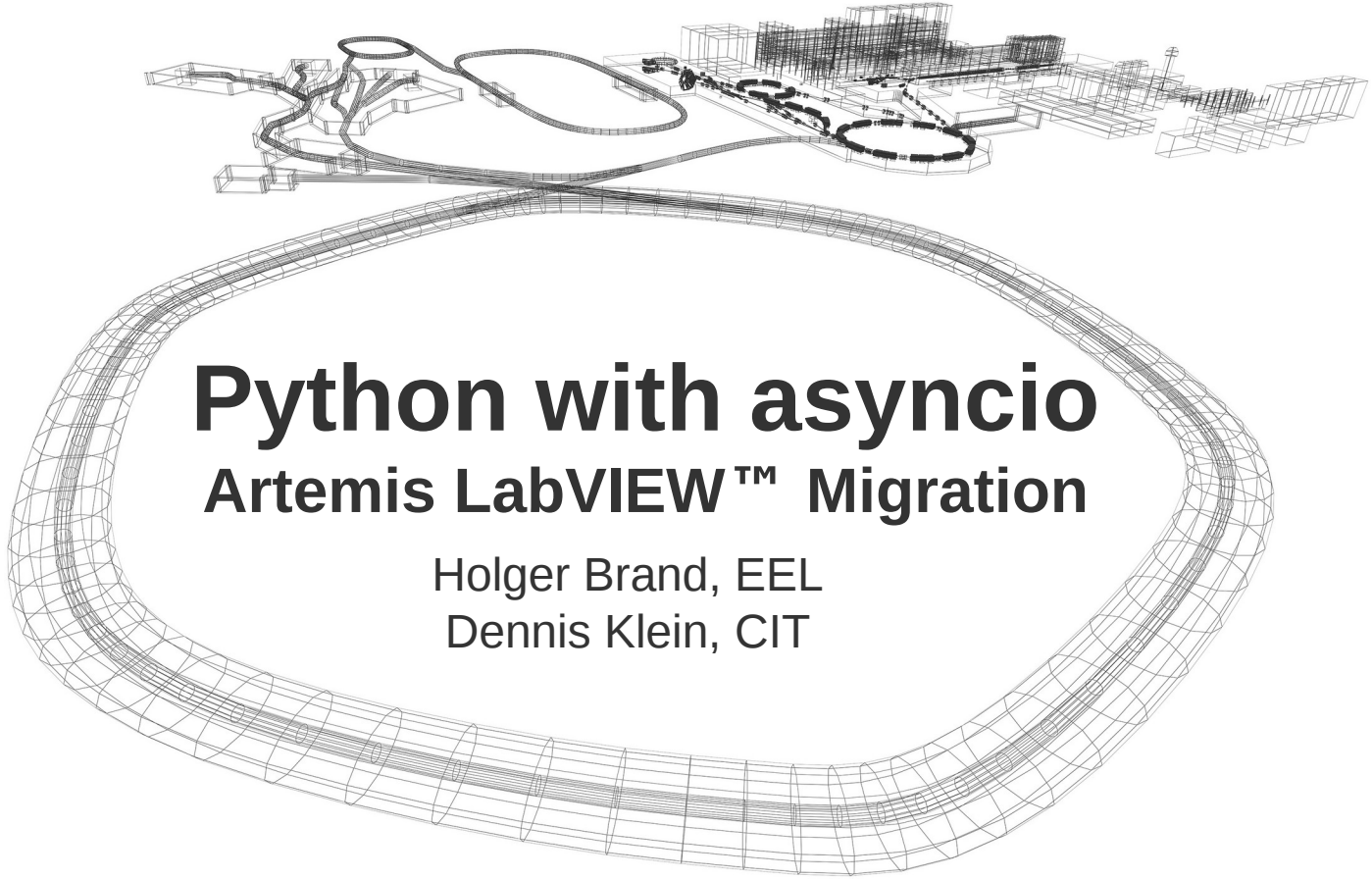
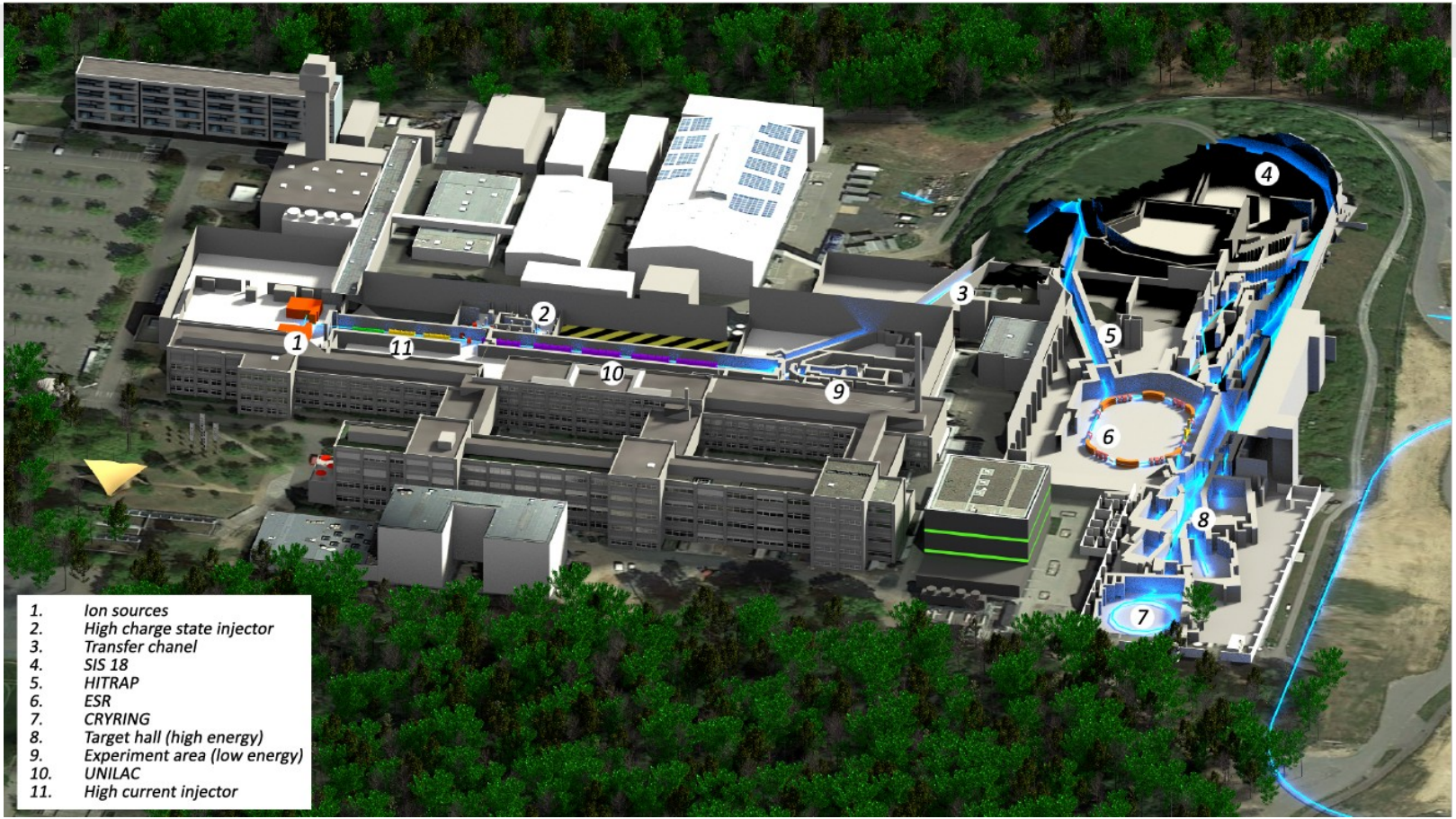


Python with asyncio

Artemis LabVIEW™ Migration

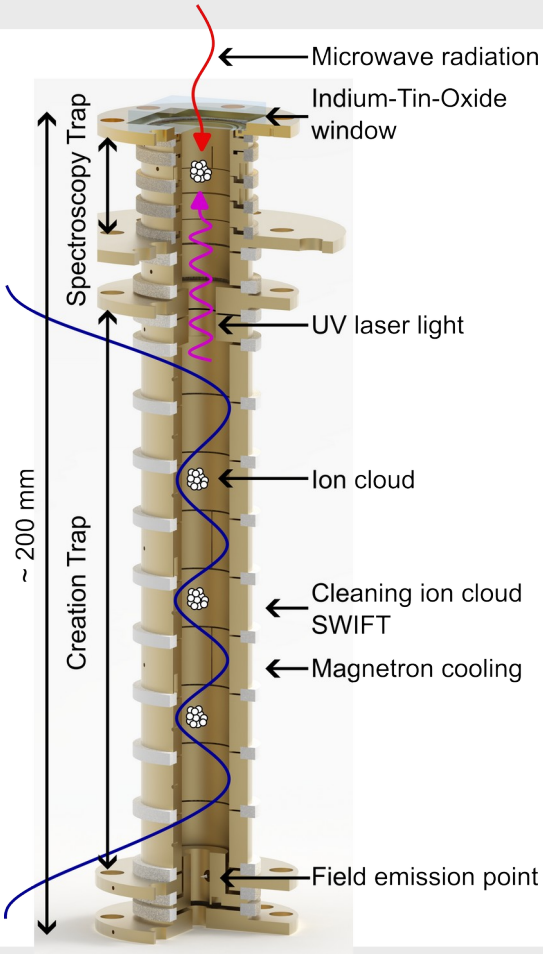
Holger Brand, EEL
Dennis Klein, CIT

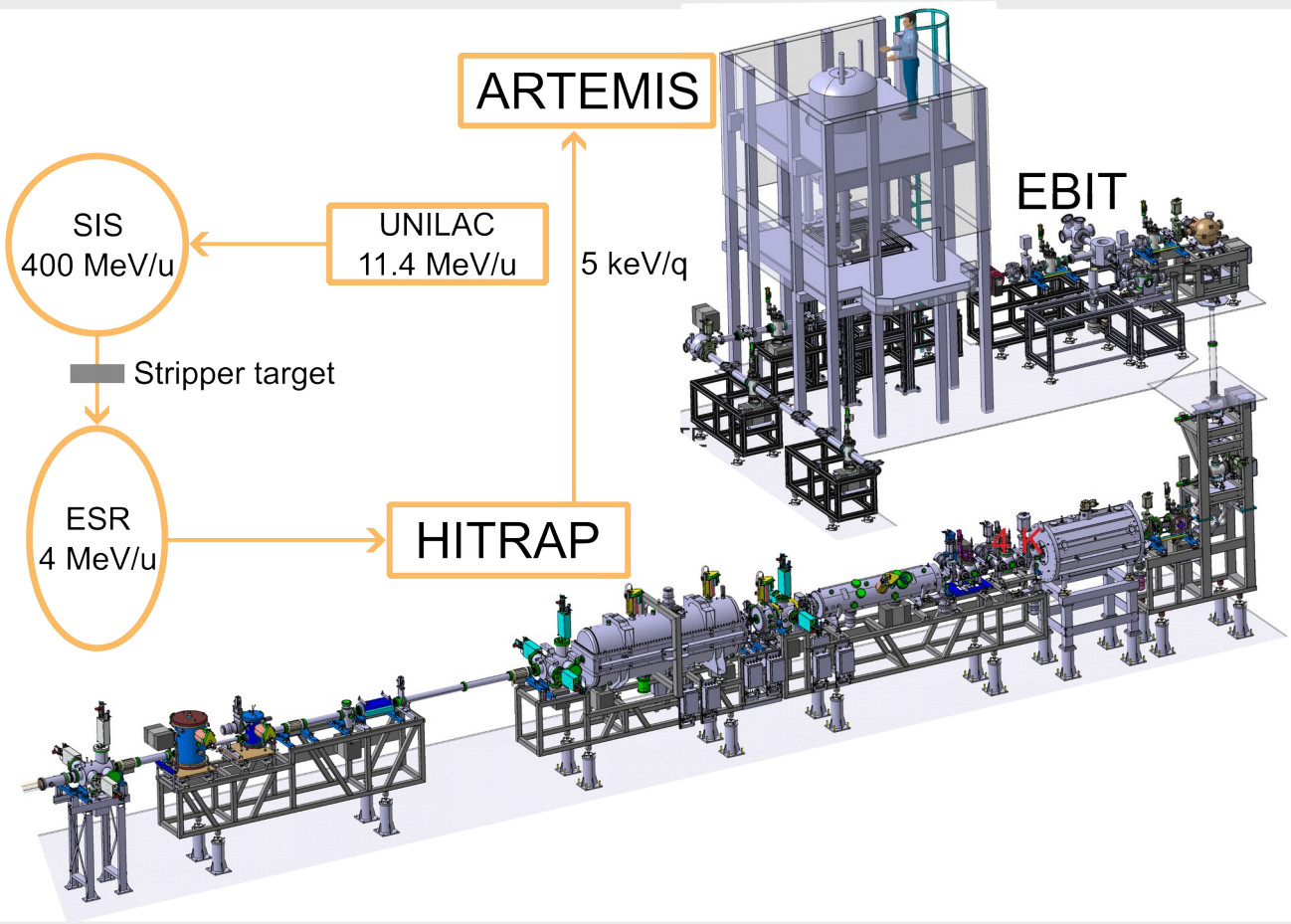
ARTEMIS: Precision Trap for g-Factor Measurements



https://www.gsi.de/en/work/research/appamml/atomic_physics/experimental_facilities/hitrap/experiments/artemis

ARTEMIS: Precision trap for g-factor measurements

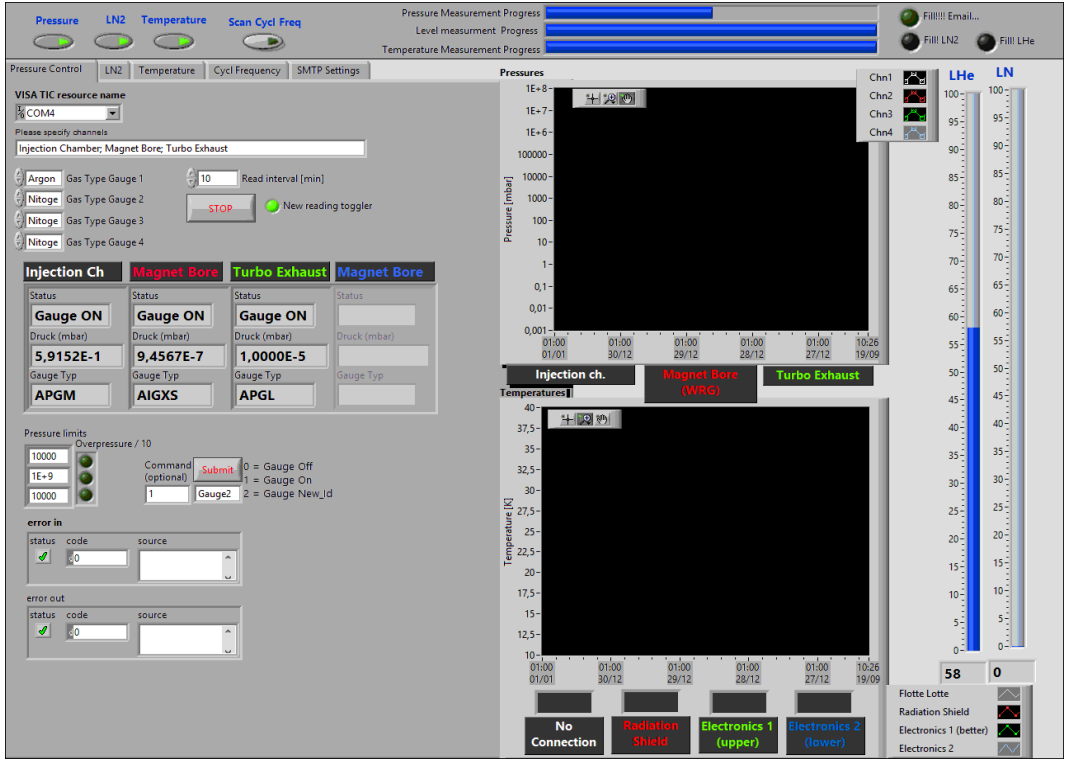




Artemis
LabVIEW DAQ & Control & Monitoring

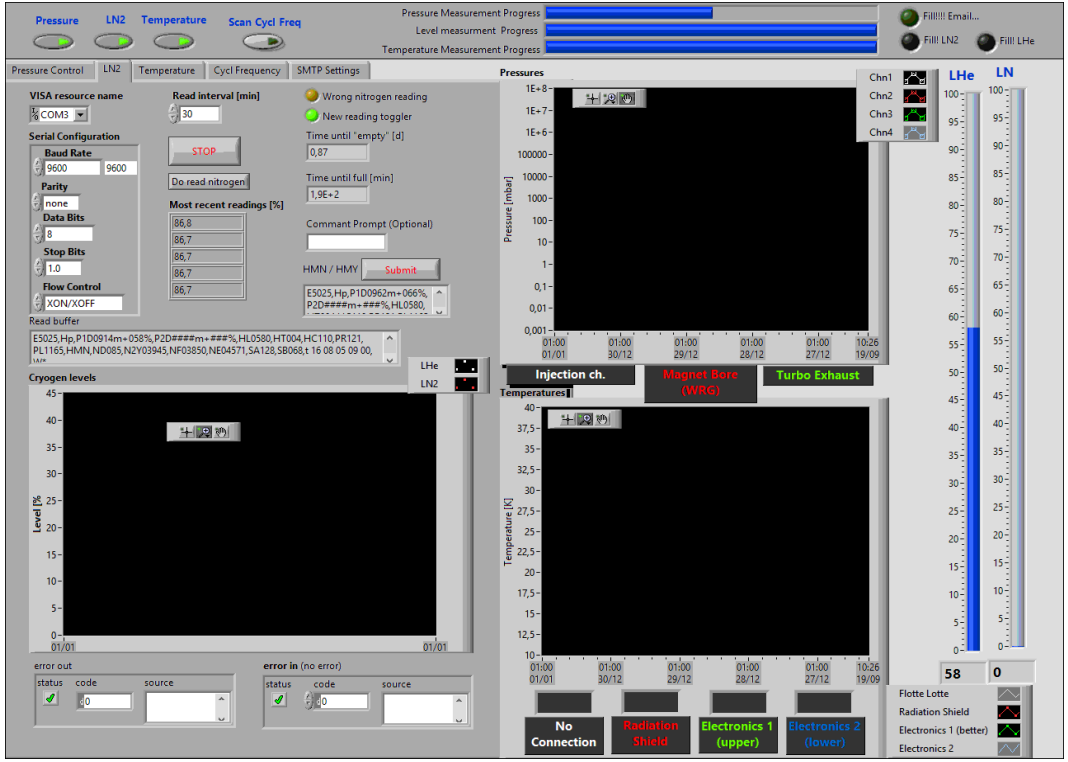
- Original LabVIEW developers are not available anymore.
- It’s a collection of quite some independent LabVIEW VI’s.
- A review of randomly selected VI’s show very simple programming approach.
- At least some VI’s
  - Missing VI’s
  - Not executable or not stable
  - Some while-loops communicating using local variables
  - Local data storage
  - Give unreliable results due
    - Not following dataflow
    - No usage of notifcation, queue etc.

AmbientAll_FoV.vi - Status Quo Pressure Control



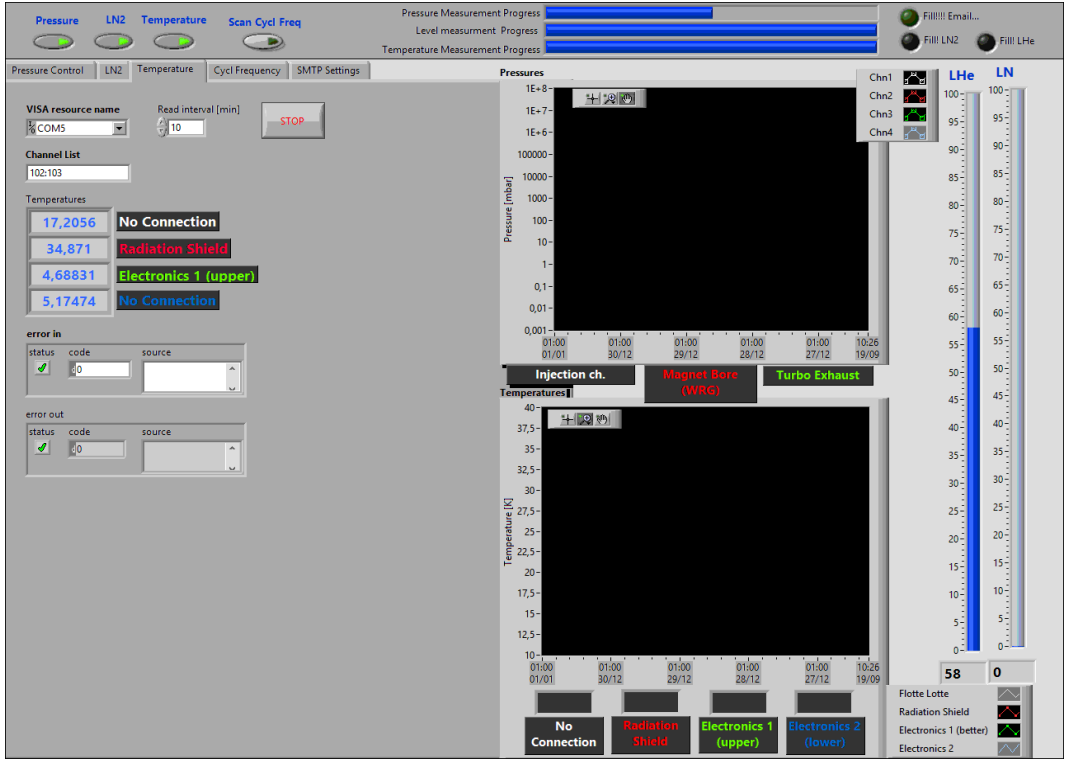
- Edwards TIC Instrument Controller 6-Gauge

AmbientAll_FoV.vi - Status Quo
He-N_2 Level Monitor



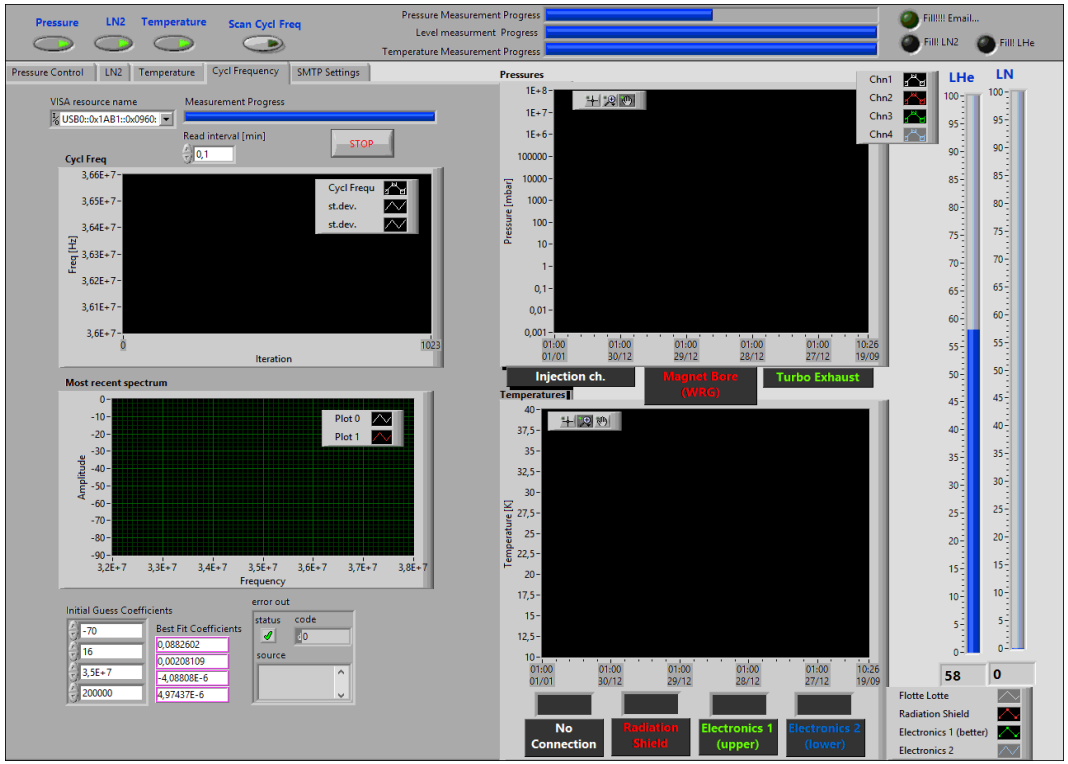
- Agilent E5025
  Digital Cryogen Monitor

AmbientAll_FoV.vi - Status Quo Temperatur Monitor



- Keithley Model 2700 Multimeter/Switch System

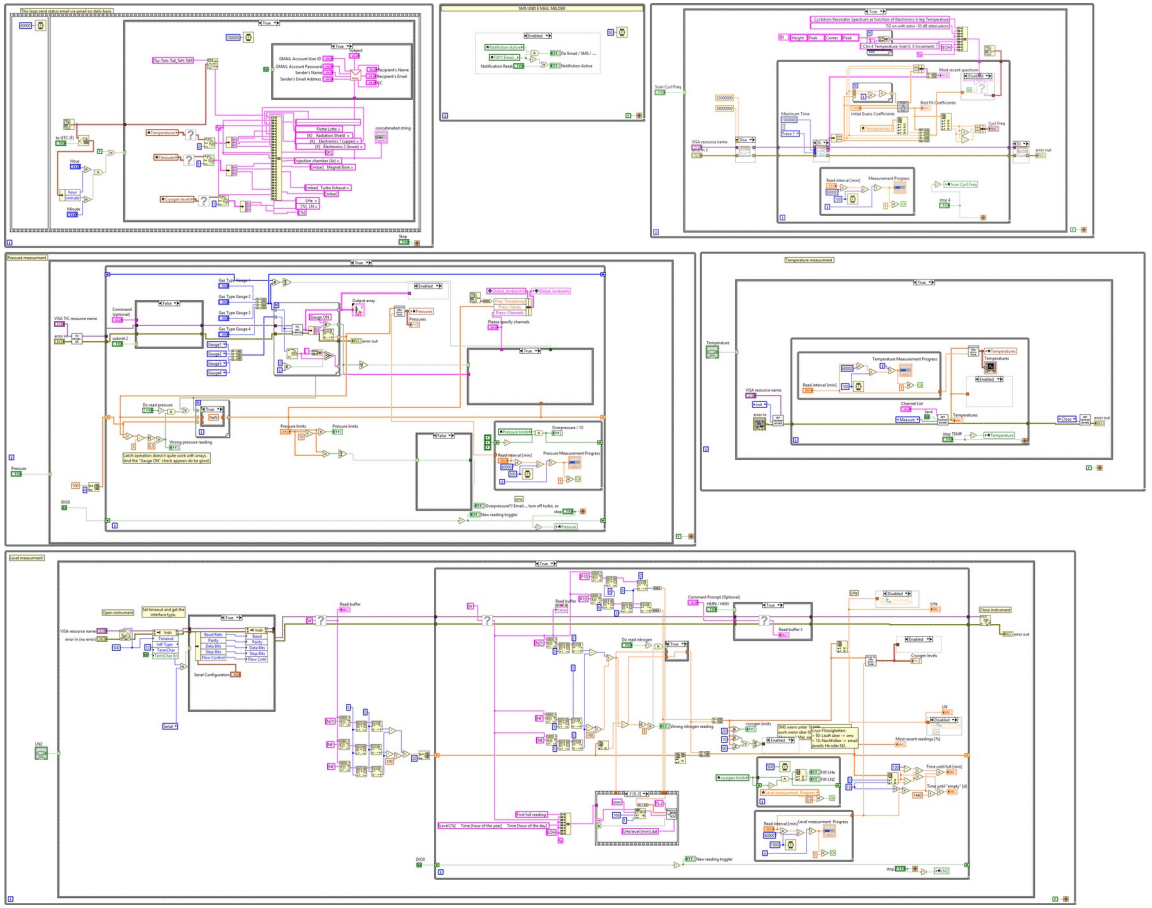
AmbientAll_FoV.vi - Status Quo
Cycle Frequency Monitor



- Rigol DSA 800
  - SubVI calls disabled

AmbientAll_FoV.vi - Status Quo Blockdiagram

- Missing VIs
- SIX Loops
  - Four instruments loops
    - One instrument deactivated
    - Serial line interfaces
      - Via RS232-Ethernet-Gateway
    - Simple implementation
      - Readout & Scaling
      - Status Display & Trendings
      - Local historical data strorage
  - Two loops dealing with eMail
    - Getting data via local variables from instrument loops
    - Send via SMTP



Migration Path

- Experimentators have no programming experience.
  - With respect to their own statements.
    - Neither LabVIEW nor Python nor other programming language.
    - Complex object-oriented frameworks are not recommended in this case.
- Instrument control/monitoring is IO driven
  - Simple approach seems feasable using Python's build-in asyncio
  - Each LabVIEW loop corresponds to an asyncio task.

- Added Value
  - Network Process Variables
    - Mosquitto MQTT Broker
    - Publish/subscribe using aiomqtt (based on paho-mqtt)
  - Historical data & Alarming
    - MQTT, Telegraf, InfluxDB2
- Visualization
  - MQTT-Explorer, Grafana, Streamlit, dedicated GUI
- Asyncio packages
  - Instrument communication
    - socket/streams, aioserial
  - User interaction: aioconsole
  - Email: aiosmtplib
  - Files: aiofile

Event-driven Programming with Python asyncio

- Software Controllers: Feedforward/-back loops
- Clock or Counter events
- (Graphical) User Interfaces: User interaction
- Drivers: Hardware I/O, Sensor readout
- Distributed Architecture: Messaging / Signaling between components
- (Micro-)Services: Databases, Monitoring

- Anatomy of event-driven programs:
  - **Event Loop**: typically provided as a framework or library, responsible for event dispatch and scheduling
  - **Event Handlers**: Contains business logic (User code)

- Primary concern: **Concurrent Multitasking**

(not distributed computing)

Concurrent Multitasking

- Preemptive methods:
  - Python threading and multiprocessing
  - Can provide parallelism as well, but **require expert programmer** (Synchronisation, Inter-process communication)
- Cooperative methods:
  - Python *„fibers“*
  - Python 3: `asyncio` + coroutines

Event handler types:

- Simple function (Callback)
  - Single entry and exit
  - Sometimes in disguise (e.g. as virtual member function overrides)
  - *„Callback hell“*
- Coroutine
  - Generalisation of a function
  - Multiple resume / suspend points
  - Avoids *„Inversion of Control“*

Code Example (1): EPICS readout task

```python
async def run_epics_readout(mqtt_send_queue):
    """Monitor the PV 'randomwalk:x' and record timeseries in memory"""
    async with caproto.asyncio.client.Context() as ca_client:
        x = (await ca_client.get_pvs("random_walk:x"))[0]
        async with x.subscribe(data_type="time") as sub:
            async for value in sub:
                mqtt_send_queue.put_nowait(
                    (value.metadata.timestamp, value.data[0])
                )
```

More complete example at: https://github.com/dennisklein/ecs_workshop_apr23_handson/blob/main/demo.py

Code Example (2): MQTT publish task

```python
async def run_mqtt_sender(send_queue, topic, min_wait=5):
    """Publish data via MQTT, but not more often than 'min_wait' seconds"""
    mqtt_client = amqtt.client.MQTTClient()
    try:
        await mqtt_client.connect("mqtt://127.0.0.1:1883/")

        while True:
            msg = []  # Batch all pending data points into one MQTT msg
            msg.append(await send_queue.get())
            while not send_queue.empty():
                msg.append(send_queue.get_nowait())

            await mqtt_client.publish(topic, json.dumps(msg).encode())

            await asyncio.sleep(min_wait)
    finally:
        await client.disconnect()
```

More complete example at: https://github.com/dennisklein/ecs_workshop_apr23_handson/blob/main/demo.py

Code Example (3): Schedule the tasks concurrently

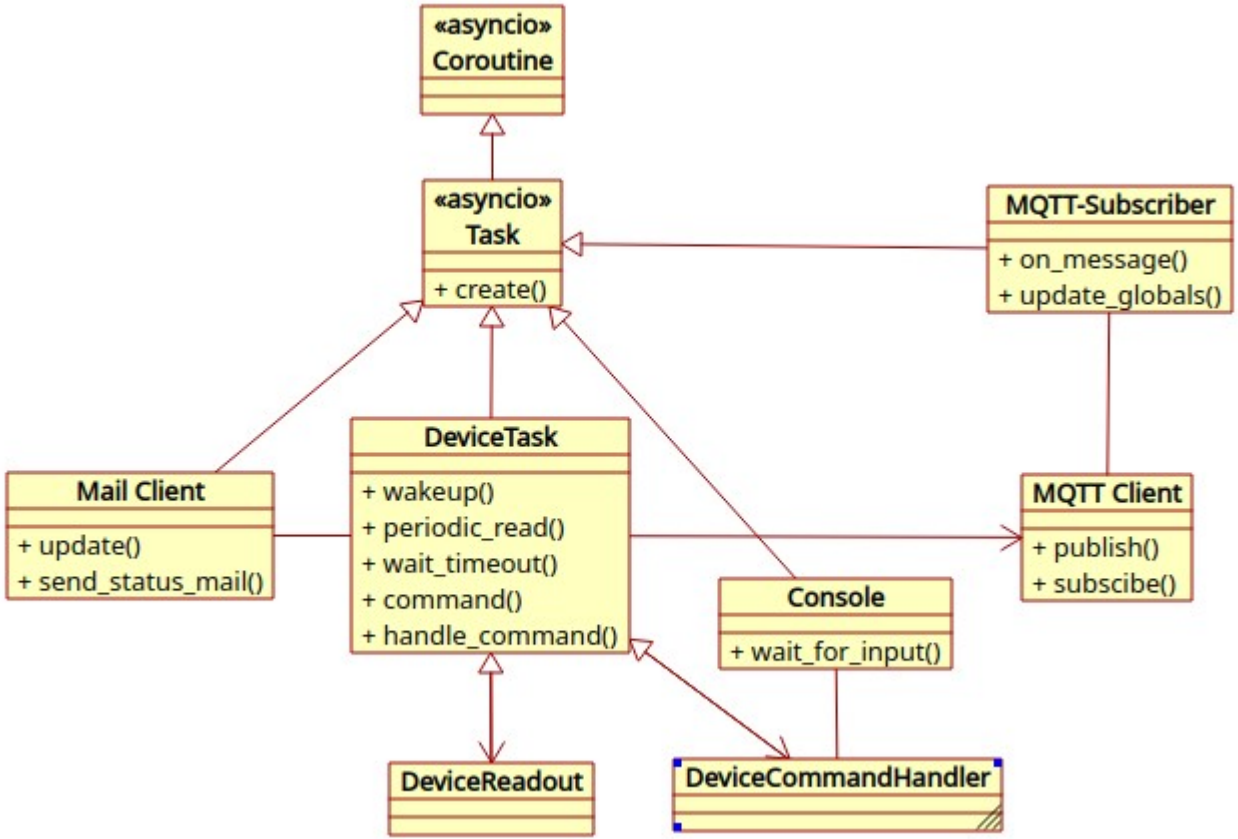
```python
async def main():
    """Main program"""
    mqtt_send_queue = asyncio.Queue()

    try:
        async with asyncio.TaskGroup() as tg:
            tg.create_task(run_epics_readout(mqtt_send_queue))
            tg.create_task(run_mqtt_sender(
                mqtt_send_queue, "/ecsws/random_walk:x"))
    except asyncio.exceptions.CancelledError:
        pass


if __name__ == "__main__":
    asyncio.run(main())
```

More complete example at: https://github.com/dennisklein/ecs_workshop_apr23_handson/blob/main/demo.py

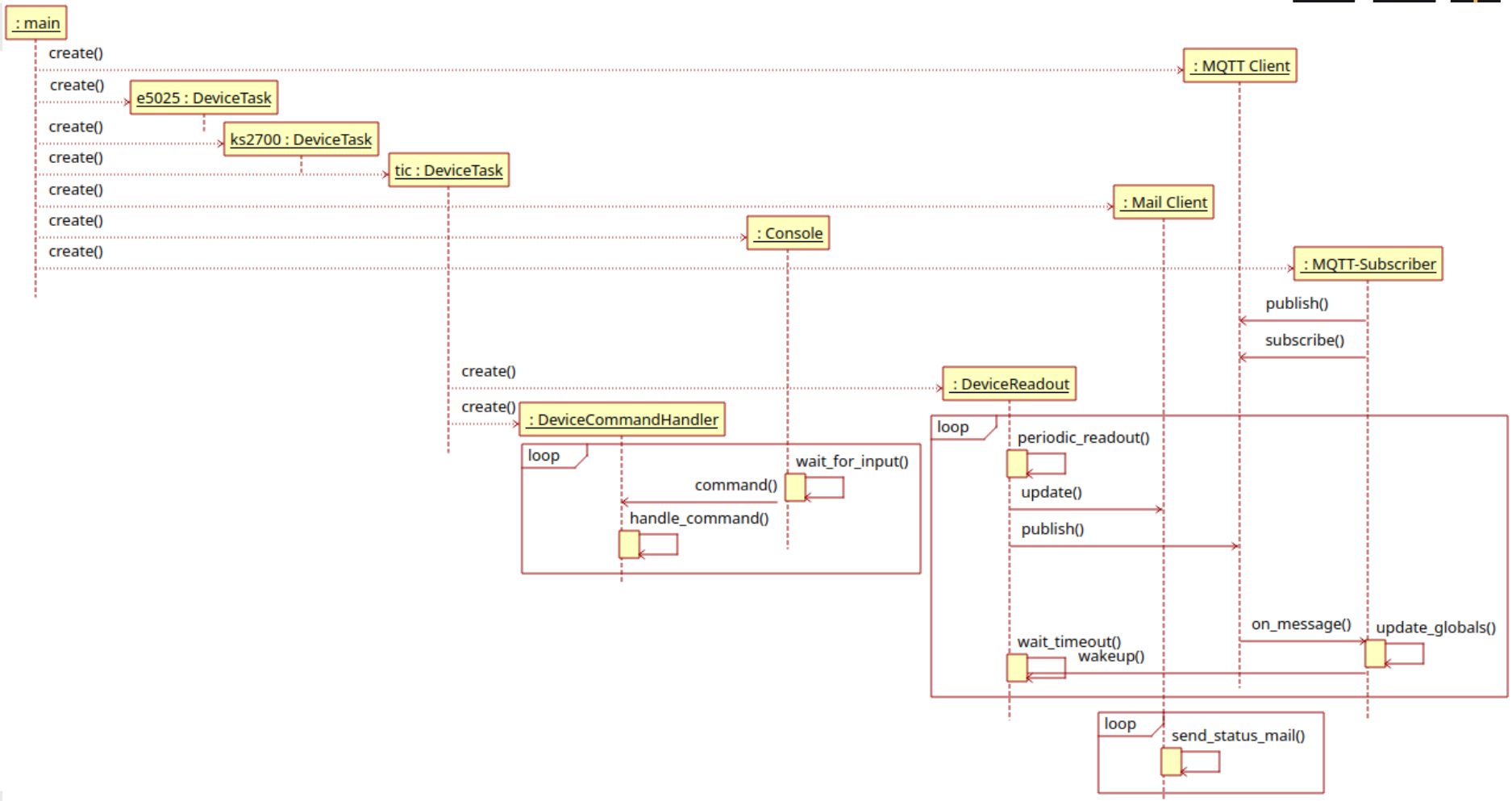
Code Example (4): Takeaways

- Reasonable and readable, but still asynchronous code
  - No „Callback Hell“, no „Inversion of Control“, -> Maintainability!
- Coroutines are a native language feature in Python 3
  - **Easy composability** across library boundaries
- asyncio is a built-in standard library module
  - Provides built-in standard event loop implementation
  - Provides many building blocks, e.g.
    - asyncio.Queue, asyncio.sleep(), asyncio.TaskGroup
    - asyncio.Future (adapter for callback-based libraries)
    - asyncio.to_thread() (adapter to blocking APIs)

Class Diagram (simplified view)



Sequence Diagram (simplified view)



main_cryo.py

```
if __name__ == '__main__':
    try:
    asyncio.run(
        main(
            mqtt={
               'broker': args.broker, 'port': args.port,
               'user': args.user, 'password': mqtt_password
               },
            email={'user': args.email_user, 'password': email_password},
            e5025_interface=args.e5025_interface,
            ks2700_interface=args.ks2700_interface,
            tic_interface=args.tic_interface
            )
        )
    except asyncio.exceptions.CancelledError as e:
        print(f'__main__: CancelledException from asyncio.run(main()): {e}')
        pass
```

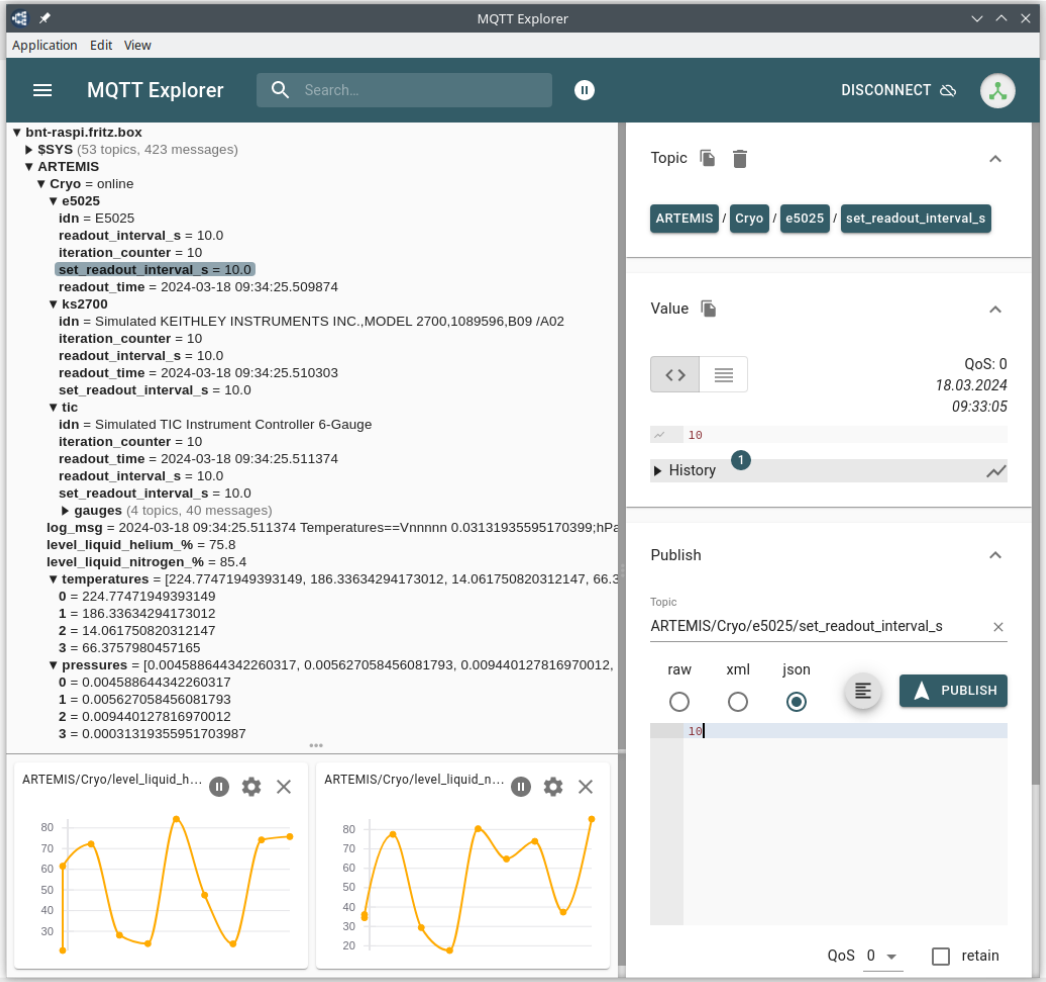
Instrument Communication Example
TIC Instrument Controller 6-Gauge

```
async def tic_transaction(mqtt_client, com_interface, command):
    """Perform low level write-read communication with device TIC6G."""
    termination = '\r'
    response = None
    if com_interface:
        await com_interface.write(command, termination)
        if re.search('\?', command):
            response = await com_interface.read_until(termination)
            cryo_tic_logger.debug(
               f'tic_transaction: TIC6G Response: {response}')
        await utilities.publish_log_msg(
               mqtt_client,
               cryo_globals.topic_log_msg,
               f'tic_transaction: {command} -> {response}')
    return response
```

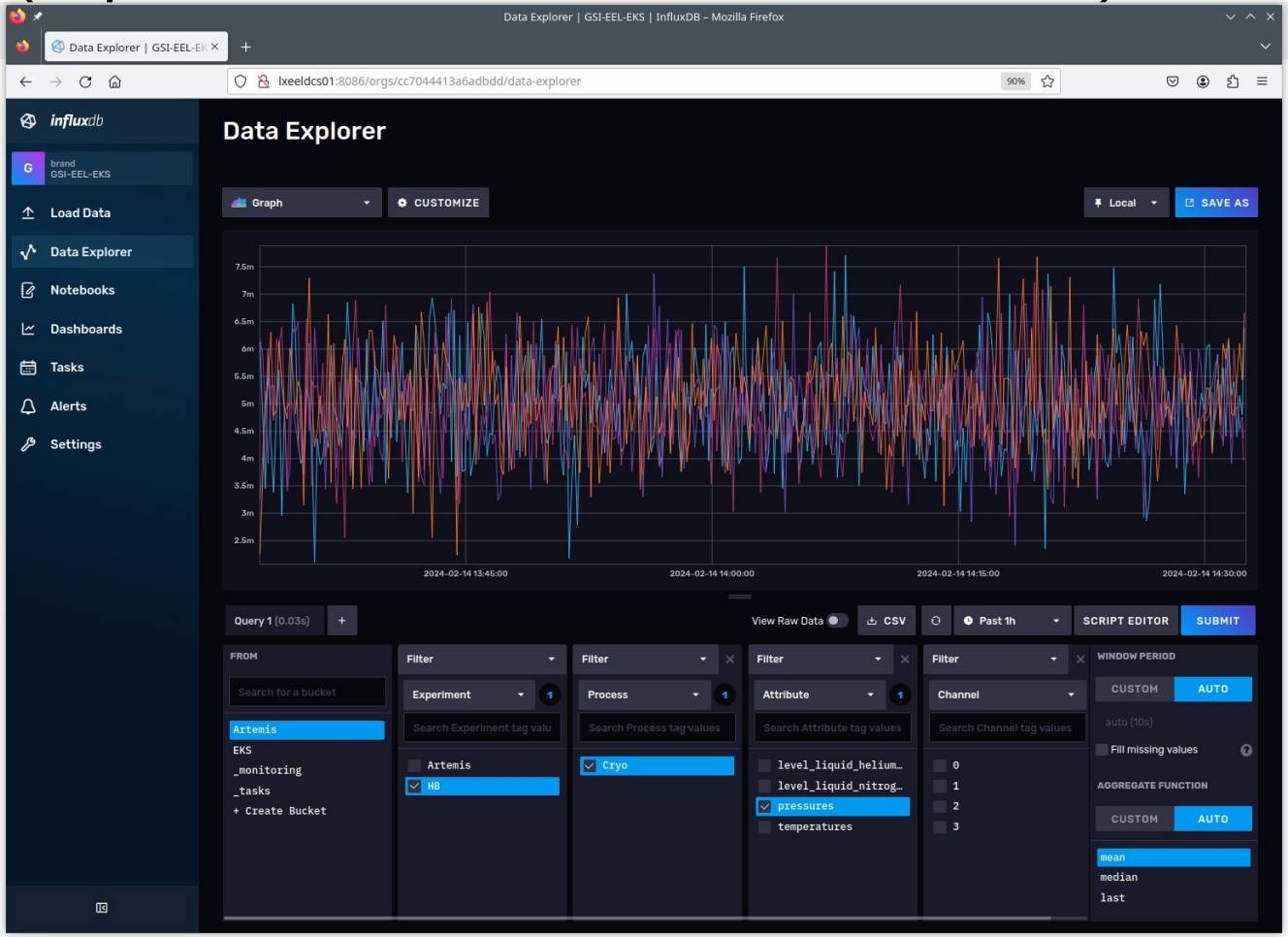
tic_readout_loop

```
async def tic_readout_loop(mqtt_client=None, tic_wakeup_event=None, com_interface=None):
    """Perform periodic readout of device TIC6G. Publish status to MQTT broker."""
    if com_interface:
        command = tic_build_command(tic_operation['QUERY SETUP'],
                                    tic_object['TIC Status'])
        response = await tic_transaction(mqtt_client, com_interface, command)
        result_list, errno = tic_parse(response)
        idn = ''
        for result in result_list:
            idn += result
    else:
        idn = 'Simulated TIC Instrument Controller 6-Gauge'
    if mqtt_client:
        try:
            await mqtt_client.publish(cryo_globals.topic_prefix + '/tic/idn', payload=idn)
        except Exception as e:
            print(f'tic_readout_loop: Exception cought. {e}')
    While True:
        # Perform readout and publish to MQTT
        await cryo.event_wait(tic_wakeup_event, cryo_globals.tic_readout_interval)
```

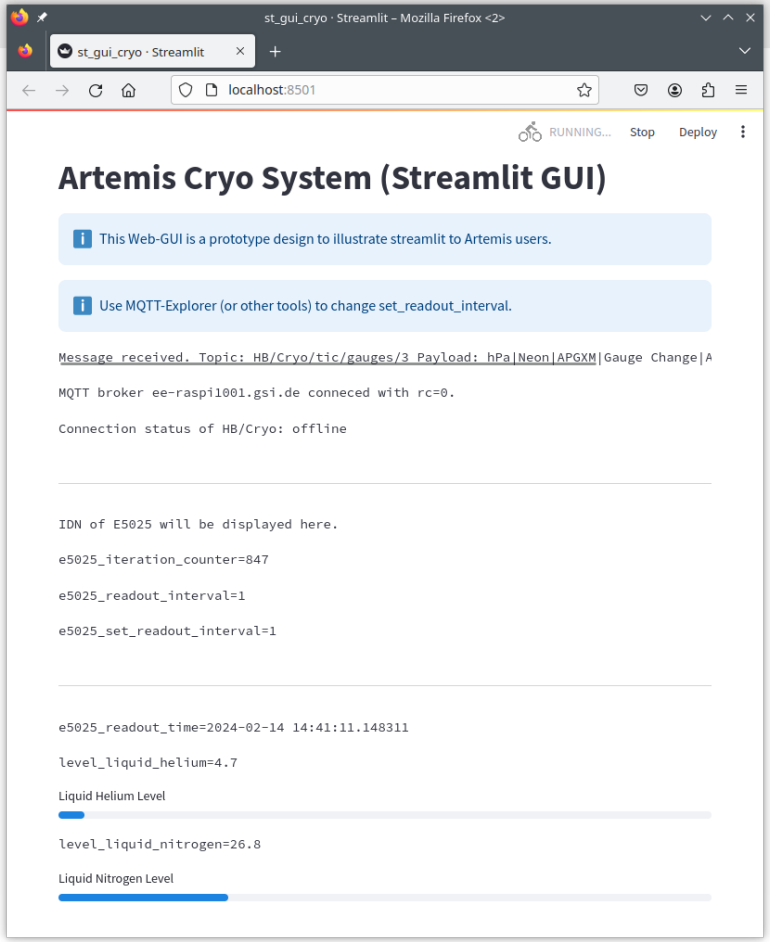
MQTT-Explorer (http://mqtt-explorer.com/)



InfluxDB Data Explorer (https://docs.influxdata.com/influxdb/v2/)



Python based Streamlit Web-GUI (https://streamlit.io/)



References

- ARTEMIS
  - Artemis images provided by Bi.Reich@gsi.de
  - https://www.gsi.de/en/work/research/appamml/atomic_physics/experimental_facilities/hitrap/experiments/artemis
  - Git-Repository: https://git.gsi.de/EKS/Python/Experiments/Artemis
- Python Docs asyncio
  - https://docs.python.org/3/library/asyncio.html
  - Experiment Control System Workshop: *Event-based Programming with Python 'asyncio'*, D. Klein
    - https://indico.gsi.de/event/17490/contributions/71045
  - SuperFastPython.com by Jason Brownlee
    - Python Asyncio Jump-Start: https://superfastpython.com/python-asyncio-jump-start/
    - Python Asyncio Mastery: https://superfastpython.com/python-asyncio-mastery/
- UML Diagram with Umbrello
  - https://uml.sourceforge.io/