

Constellation



**A flexible control and data acquisition framework
for dynamic small-scale experiments**

Stephan Lachnit, Simon Spannagel
Tutorial @ BTTB13

HELMHOLTZ



Content

1. Brief reminder of Constellation
2. Tutorials to get used to the software
3. Implementation of Temperature Readout

Introduction

- Reminder on Constellation:
 - Control and Data Acquisition Framework
 - Tailored towards dynamic small-scale environments (like test beam)
 - Synchronous operation of multiple devices in a local network
 - Programs operating those device are called satellites
- See also BTTB talk slides on [indico](#)
- If not done yet, install Constellation using instructions on [indico](#)

Tutorials

Controlling a Single Satellite

Using the Python controller (command line interface)

- Let's start by starting a controlling a single satellite using an interactive Python shell
- Follow this tutorial:
https://constellation.pages.desy.de/operator_guide/tutorials/single_satellite.html
- If you installed the Flatpak version on Linux, you need to replace
replace `SatelliteSpuntik`
with `Satellite -t Sputnik`
- **Important:** since everyone is in a common subnet, use a unique group name
via `-g group_name` to avoid controlling satellites from some else

Controlling Multiple Satellites

Using MissionControl (graphical user interface)

- Let's continue with a more advanced setup using multiple satellites
- Follow this tutorial:
https://constellation.pages.desy.de/operator_guide/tutorials/missioncontrol.html
- If you installed the Flatpak version on Linux, you can start MissionControl by searching for MissionControl in the applications (usually with the Windows key)
- **Important:** since the Pi has very little RAM, use `/home/pi/data` as `output_directory`
- **Important:** since everyone is in a common subnet, use a unique group name via `-g group_name` to avoid controlling satellites from some else

Logging of Multiple Satellites

Using Observatory (graphical user interface)

- Let's continue with a more advanced setup using multiple satellites
- Follow this tutorial:
https://constellation.pages.desy.de/operator_guide/tutorials/observatory.html
- If you installed the Flatpak version on Linux, you can start Observatory by searching for Observatory in the applications (usually with the Windows key)
- **Important:** since the Pi has very little RAM, use `/home/pi/data` as `output_directory`
- **Important:** since everyone is in a common subnet, use a unique group name via `-g group_name` to avoid controlling satellites from some else

Implementation

Implementing a Temperature Monitor

Your turn

- Goal: implement a satellite which provides the temperature as metrics
- Additional features:
 - Configurable measurement interval
 - Configurable metric name
 - Temperature available as custom command
 - Configurable critical threshold:
 - Log to critical when threshold is reached
 - Go to ERROR state

Implementing a Temperature Monitor

Setting up the environment

- Create a new folder for the project and cd into it
- Create a virtual environment: `python3 -m venv venv`
- Activate the virtual environment: `source venv/bin/activate`
- Install Constellation: `pip install ConstellationDAQ[cli]`
- Install pyserial for the sensor: `pip install pyserial`

Implementing a Temperature Monitor

Creating a dummy satellite

- Create a new Python file named `temp_mon_satellite.py`
- Create a new satellite class:

```
from constellation.core.configuration import Configuration
from constellation.core.satellite import Satellite
```

```
class TempMon(Satellite):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

    def do_initializing(self, config: Configuration) -> None:
        pass
```

- `__init__` is run when the satellite is created => should not initialize the hardware!
- `do_initializing` is run when the config is received from the controller => initialize hardware

Implementing a Temperature Monitor

Creating the starting script

- Extra code is required to setup the CLI parsing and logging as well as starting the satellite
- Create a new file called `main.py`:

```
from constellation.core.logging import setup_cli_logging
from constellation.core.satellite import SatelliteArgumentParser
from temp_mon_satellite import TempMon

def main(args=None):
    parser = SatelliteArgumentParser()
    args = vars(parser.parse_args(args))
    setup_cli_logging(args.pop("log_level"))
    s = TempMon(**args)
    s.run_satellite()

if __name__ == "__main__":
    main()
```

Implementing a Temperature Monitor

Starting the satellite

- Start the satellite:

```
python3 main.py -g group_name
```

- **Reminder:** since everyone is in a common subnet, use a unique group name via `-g group_name` to avoid controlling satellites from some else
- **Note:** after changes to the code, the satellite needs to be restarted for the changes to take effect
- Open MissionControl and and control the satellite

Constellation	Satellites	State	Run Identifier	Run Duration
edda	1	New	run_1 (next)	00:00:00

Type	Name	State	Last response	Last message	Heartbeat	Lives
PiTemp	pitemp	New	SUCCESS		1100ms	3

Implementing a Temperature Monitor

Reading the temperature

- The temperature sensors print the temperature every second in a serial port
- On Linux:
 - The default permissions do not allow to read from serial ports, you can get the required group for serial access by running `stat /dev/ttyACM0`
 - Add yourself to the group (e.g. dialout) using `sudo usermod -a -G dialout $USER` (and reboot)
 - Read the temperature using `cat /dev/ttyACM0`
- On MacOS:
 - The name is random, use `ls /dev/cu.usbmodem*` to find the sensor
 - Read the temperature using `cat /dev/cu.usbmodem<number>`

Implementing a Temperature Monitor

Reading the temperature

- Download this file: <https://gist.github.com/stephanlachnit/c6c1bebd5b110220c575f6e56455bdaf>
- It contains a class named `TempSensor` which takes the path to the serial port as argument
- Create an instance of the `TempSensor` class in the `do_initializing` function of the satellite using the path determined earlier (we will make it configurable later)
- Add a `read_temperature(self)` function to the satellite which returns the `read_temperature()` function of the `TempSensor` class:

```
def read_temperature(self):
    if self.temp_sensor:
        return self.temp_sensor.read_temperature()
    raise Exception("Temperature sensor not initialized yet")
```

Implementing a Temperature Monitor

Adding temperature reading code

- For now, let's log the temperature in the **RUN** state by adding a `do_run` function to the satellite:

```
import time

def do_run(self, payload) -> None:
    while not self._state_thread_evt.is_set():
        self.log.info(f'Current temperature: {self.read_temperature()}°C')
        time.sleep(0.1)
```

- Open MissionControl and Observatory and initialize and launch the satellite

Time	Sender	Level	Topic	Message
2025-05-02 16:29:23	PiTemp.slthin...	STATUS	FSM	State transition to steady state completed (stopping -> ORBIT).
2025-05-02 16:29:23	PiTemp.slthin...	STATUS	FSM	State transition stop initiated.
2025-05-02 16:29:22	PiTemp.slthin...	INFO	PITEMP	Current temperature: 20°C
2025-05-02 16:29:21	PiTemp.slthin...	INFO	PITEMP	Current temperature: 24°C
2025-05-02 16:29:20	PiTemp.slthin...	INFO	PITEMP	Current temperature: 25°C
2025-05-02 16:29:19	PiTemp.slthin...	INFO	PITEMP	Current temperature: 19°C
2025-05-02 16:29:18	PiTemp.slthin...	INFO	PITEMP	Current temperature: 20°C
2025-05-02 16:29:17	PiTemp.slthin...	INFO	PITEMP	Current temperature: 20°C
2025-05-02 16:29:16	PiTemp.slthin...	INFO	PITEMP	Current temperature: 22°C
2025-05-02 16:29:16	PiTemp.slthin...	STATUS	FSM	State transition start initiated.
2025-05-02 16:29:16	PiTemp.slthin...	STATUS	SATELLITE	Starting run 'run_1'

Note: change the global log level to INFO!

Implementing a Temperature Monitor

Implementing temperature as custom command

- Let's implement a custom command to read the temperature by adding a new function:

```
from constellation.core.commandmanager import cscp_requestable
from constellation.core.cscp import CSCPMMessage
```

```
@cscp_requestable
def get_temp(self, request: CSCPMMessage):
    """Get the current temperature"""
    temp = self.read_temperature() # Numeric response
    verb = f"{temp}°C" # Human-readable response
    return verb, temp, {}
```

- Restart and try the custom command with MissionControl

Implementing a Temperature Monitor

Implementing metrics

- Let's schedule a metric for the temperature in `do_initializing`:

```
from constellation.core.cmdp import MetricsType

def do_initializing(self, config: Configuration) -> None:
    sampling_interval = 1 # in seconds
    self.schedule_metric('NAME', 'UNIT', MetricsType.LAST_VALUE,
                        sampling_interval, self.read_temperature)
```

- Note: any existing metrics with the same name are overwritten
- If the metric name is configurable, the metrics need to be reset before:

```
self.reset_scheduled_metrics()
```

- Verify the satellite initializes (we will not see metrics in action yet)

Implementing a Temperature Monitor

Reading the configuration

- Let's read the configuration in `do_initializing`:

```
def do_initializing(self, config: Configuration) -> None:
    value1 = config['param1'] # Satellite goes to ERROR if not present
    value2 = config.setdefault('param2', 3.14) # Uses default if not present
    self.log.info(f'param1 = {value1} and param2 = {value2}')
```

- If the config is inconsistent or hardware initialization fails, simply raising an exception is sufficient to correctly go to the ERROR state (e.g. `raise Exception('reason')`)
- Goal: implement `port`, `channel_name` and `sampling_interval` configuration parameters to configure the scheduled metric
- Don't forget to reset the scheduled metrics before registering them again!

Implementing a Temperature Monitor

Monitoring with Grafana

- Let's take a look at the temperature data using Grafana (dashboard viewable in a browser)
- Start your satellite with the edda group
- Open MissionControl and Observatory to see what's going on
- We will initialize, launch and start the satellites
- We will start a new satellite called Influx, which writes the monitoring data to a database
- We will take a look at the monitoring data and create a dashboard in Grafana

- If you ever want to set up Grafana yourself, check [this how-to guide](#)

Implementing a Temperature Monitor

Reading the configuration

- Let's implement a (configurable) critical temperature threshold:
 - In `do_run`, check if the temperature exceeds the threshold
 - If the temperature threshold is reached, use `self.log.warning` to log a warning
 - Turn on the red LED when the threshold is exceeded
- Run and check with Observatory that the warning is emitted
- Let's make this more severe by going to `ERROR` if the threshold is exceeded:
 - Add `raise Exception('reason')` when the threshold is reached
 - Don't forget to reset the LED in `do_initialize`
- Now try again and start another satellite (e.g. `SatelliteSputnik`)
 - When the temperature threshold is reached, the other satellite will transition to `SAFE`
 - This can be used e.g. to shut off sensitive electronics

Implementing a Temperature Monitor

Possible further tasks

- Add a non-critical threshold which just logs a warning
- Setup Influx and Grafana:
 - Install podman, podman-docker and podman-compose
 - Follow [this how-to guide](#)
- Send data and store it in an HDF5 file
 - Use DataSender class from `constellation.core.datasender`
 - Create numpy array in `do_run`:

```
data = np.array([temperature])
self.data_queue.put((data.tobytes(), {"dtype": f"{data.dtype}"}))
```
 - Use [SatelliteH5DataWriter](#) to store data (requires HDF5 installation via apt and h5py)