

# Constellation Tutorial at BTTB

## Installation

### Linux

- Install Flatpak as described in <https://flathub.org/setup>
- Install Constellation via `flatpak install flathub de.desy.constellation`
- Ensure that Python 3.11 or newer is available

### MacOS

- Update your MacOS installation via `sudo softwareupdate -i -a -R` (important)
- Clone <https://gitlab.desy.de/constellation/constellation>
- Follow the build instructions on our website [https://constellation.pages.desy.de/application\\_development/intro/install\\_from\\_source.html#c-version](https://constellation.pages.desy.de/application_development/intro/install_from_source.html#c-version)

### Windows

- Install WSL as described in <https://learn.microsoft.com/windows/wsl/install>
- Set the WSL networking mode to *Mirrored* in the WSL Settings
- Install Constellation inside WSL following the Linux instructions
- Install USBIPD-WIN to allow connection USB devices to WSL as described in <https://learn.microsoft.com/windows/wsl/connect-usb>

### Firewall

- It might be needed to disable the firewall temporarily in case Constellation does not detect any satellites in the local network.

## Preparing the Template

- Clone <https://github.com/stephanlachnit/bttb-satellite>
- Create and activate a venv inside the folder
- Install the satellite via `pip install -e .`

# Tutorial

In this tutorial a simple Constellation satellite is implemented for a temperature sensor and online monitoring is used to visualize the temperature measurement.

## Triggered Temperature Board

A custom PCB called the Triggered Temperature Board<sup>1</sup> (TTB) is used for the temperature measurement. The board is equipped with a Raspberry Pi Pico 2 microcontroller<sup>2</sup> and a DS18B20 temperature sensor<sup>3</sup>.

The DS18B20 temperature sensor has an adjustable temperature resolution ranging from 0.5 °C to 0.0625 °C. However, finer resolution requires longer to convert to a digital value. In Table 1, the resolution and corresponding conversion times are shown.

Resolution Bits	Conversion Time [ms]	LSB [°C]
9	93.75	0.5000
10	187.50	0.2500
11	375.00	0.1250
12	750.00	0.0625

Table 1: DS18B20 resolution and conversion times

The microcontroller hosts the firmware and takes care of the serial communication via USB.

The microcontroller needs a few milliseconds of processing time in addition to the conversion times shown in Table 1 to read out the temperature from the sensor and send it over serial.

The firmware on the device implements a request-reply pattern. When connected, a request must be sent over serial to the device, which responds with a reply. Like many serial devices, data is transmitted as text encoded in ASCII<sup>4</sup>. A list of commands accepted by the firmware is shown in Table 2.

Command	Argument	Description	Return value
TEMP?	-	Get current temperature in °C	Float or FAIL
RES	Int	Set DS18B20 resolution in bits	OK or FAIL
RES?	-	Get DS18B20 resolution in bits	Int
TRIG	Bool	Enable or disable triggered mode	OK
TRIG?	-	Get current triggered mode	Bool

Table 2: Firmware commands

The firmware features a triggered mode. In this mode, the temperature is read out and sent via serial whenever a trigger arrives via the HDMI port.

## Establishing Serial Communication

The first task is to establish serial communication with the temperature board using a simple Python script with `pyserial`<sup>5</sup>.

<sup>1</sup><https://triggered-temperature-board-f7662e.pages.desy.de/>

<sup>2</sup><https://www.raspberrypi.com/products/raspberry-pi-pico-2/>

<sup>3</sup><https://www.analog.com/en/products/ds18b20.html>

<sup>4</sup><https://en.wikipedia.org/wiki/ASCII>

<sup>5</sup><https://pyserial.readthedocs.io/>

On Unix systems serial ports are terminals (`tty`) with standard output (TX) and input (RX). Depending on the interface their file names can be `ttyS*`, `ttyUSB*` or `ttyACM*`. Connect the temperature board and verify that the serial port is present under `/dev` as `ttyACM0`.

Create a new script called `serial_communication.py` and establish a serial connection:

```
import serial

serial_connection = serial.Serial('/dev/ttyACM0')
```

The next step is to send an ASCII-encoded command via serial. This can be done via:

```
serial_connection.write('TEMP?\n'.encode('ascii'))
```

Note that when using terminal-like serial communication, it is important to always end the command with a newline character (which in most cases is `\n`).

Finally, after sending the command, the response from the device can be read:

```
reply = serial_connection.readline().decode('ascii').strip()
print(reply)
```

The output is stripped since the device also ends its reply with a newline character.

Now run the Python script and read the temperature of the device. Extend the Python script to print the temperature periodically in a loop. You can heat the temperature sensor on the board with your fingers to increase the measured temperature. Change the resolution of the temperature sensor and observe the difference (qualitatively) in readout speed.

## Constellation Integration

In Constellation, each participant in the data acquisition is called a *satellite*. Satellites can correspond to real devices such as the temperature board, but also to programs which connect to databases to store monitoring data. Each satellite has a finite state machine (FSM) which controls the behaviour of the satellite and guarantees that it is always in a defined state. In Constellation, there are four non-error steady states: *NEW*, *INIT*, *ORBIT* and *RUN*. Satellites start in the *NEW* state without any configuration. During the *initialize* transition, which brings the satellite in the *INIT* state, the satellite receives a configuration and performs basic sanity checks. The *launch* transition brings the satellite to the *ORBIT* state in which the satellite is ready for data-taking. Finally, the *starting* transition brings the satellite to the *RUN* state in which the satellite is taking data. Figure 1 shows a state diagram of the FSM.

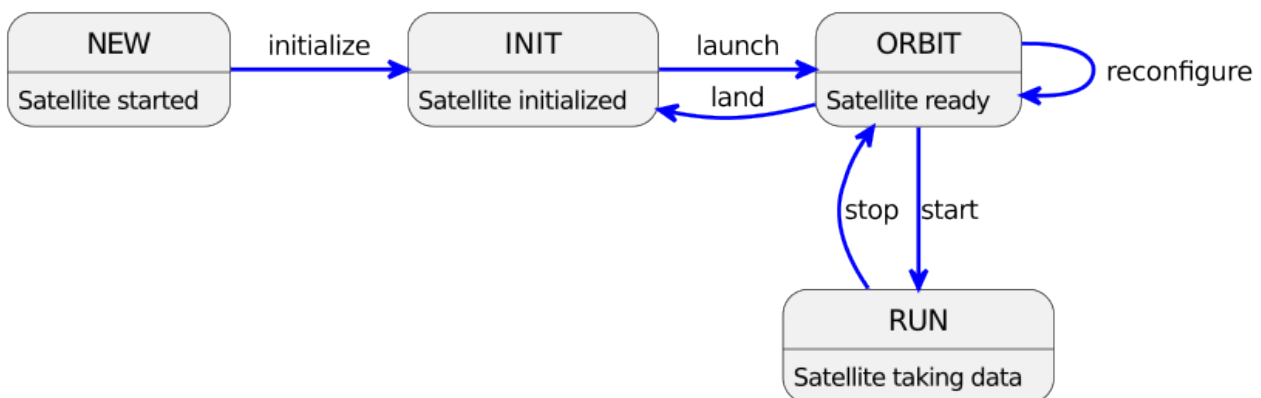


Figure 1: Finite State Machine of a Satellite

The goal is now to integrate the serial communication in a structured way into the finite state machine of Constellation.

Start the satellite within the `bttb` group:

```
SatelliteBTTB -g bttb
```

The satellite can be shut down by pressing Ctrl-C on the keyboard.

Now, instructions for the transitions of the finite state machine need to be added. The code of the satellite is located in `src/BTTB/BTTB.py` and contains a `TTB` class inheriting from the `Satellite` class with a few helper already defined.

The `query` method takes care of encoding, writing to serial port, reading from the serial port and decoding. The `query_ok` checks that the command returns OK instead of returning the reply.

In the *initializing* transition, the satellite should read the port and resolution from the configuration and open the serial port. The *initializing* transition can be added to the class using the following method:

```
def do_initializing(self, config: Configuration) -> None:
    """Initialize satellite"""
    # Read configuration
    serial_port = config.get_str("port")
    resolution = config.get_int("resolution", default_value=12)

    # Task: open serial port with 1s timeout
    self._serial_connection = ...
    # Task: set resolution
    self.query_ok(...)
```

After implementing the *initializing* transition, the next task is to implement a metric which periodically emits the current temperature while in the *ORBIT* and *RUN* state. This can be achieved by adding the following method to the satellite class:

```
@schedule_metric("°C", 1, [SatelliteState.ORBIT, SatelliteState.RUN])
def temperature(self) -> float:
    """Temperature of the sensor"""
    # Task: read temperature, convert to float and return
    ...
```

## Controlling the Satellite

MissionControl is a graphical user interface for controlling satellites in Constellation. Start MissionControl by searching for it in the application menu. Once MissionControl is started, a group name needs to be provided. Then start the satellite as before with the same group name as for MissionControl. In the main window, the satellite should appear in the list (a detailed guide with pictures can be found online<sup>6</sup> if needed).

A configuration file has to be provided before the satellite can be initialized. Create a file called `config.toml` in your group folder and add the following lines:

```
[BTTB._default]
port = "/dev/ttyACMO"
resolution = 12
```

---

<sup>6</sup>[https://constellation.pages.desy.de/operator\\_guide/tutorials/missioncontrol.html](https://constellation.pages.desy.de/operator_guide/tutorials/missioncontrol.html)

Select the configuration file in MissionControl, click the *Initialize* and finally the *Launch* button to bring the satellite in the *ORBIT* state.

After successfully launching the satellite, the next task is to ensure that the satellite works correctly.

Now we want to visualise the temperature metric. Start TelemetryConsole from the application menu. At the top of the window, select the satellite as sender and the temperature metric. Set a time range and click *Create* to create a plot (a detailed guide with pictures can be found online<sup>7</sup> if needed).

Finally start Observatory, which is a graphical logging interface, from the application menu. On the right side under *Individual Subscriptions*, select the **DEBUG** log level for the BTTB satellite (a detailed guide with pictures can be found online<sup>8</sup> if needed). Every second when the metric is sent two log messages appear containing what was written and read from the serial port.

## Optional: Online Monitoring with Grafana

As a last step, an online dashboard for monitoring using Grafana<sup>9</sup> can be created. Constellation can store all metrics in InfluxDB<sup>10</sup>, which is a time series database and possible data source for Grafana.

For this part, an installation of Podman<sup>11</sup> is required. The instructions for setting up InfluxDB and Grafana can be found in the operator guide<sup>12</sup>.

Once set up, create a visualization for the temperature. Have fun creating different kinds of visualizations!

---

<sup>7</sup>[https://constellation.pages.desy.de/operator\\_guide/tutorials/telemetryconsole.html](https://constellation.pages.desy.de/operator_guide/tutorials/telemetryconsole.html)

<sup>8</sup>[https://constellation.pages.desy.de/operator\\_guide/tutorials/observatory.html](https://constellation.pages.desy.de/operator_guide/tutorials/observatory.html)

<sup>9</sup><https://grafana.com/oss/grafana>

<sup>10</sup><https://www.influxdata.com/>

<sup>11</sup><https://podman.io/>

<sup>12</sup>[https://constellation.pages.desy.de/operator\\_guide/howtos/setup\\_influxdb\\_grafana.html](https://constellation.pages.desy.de/operator_guide/howtos/setup_influxdb_grafana.html)