

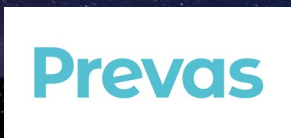




Towards Constellation 1.0: Autonomous Control Systems for Laboratory, Testbeams & Beyond

Simon Spannagel, DESY
for the EDDA Collaboration

4th DRD3 Week, CERN
2025-11-13



Okay, what's Constellation?



Constellation

Autonomous Control and Data Acquisition System

Constellation is a control and data acquisition system for small-scale experiments and experimental setup with volatile and dynamic constituents such as testbeam environments or laboratory test stands.



Yes, *thanks*, but what *is* it?

...it's all of this:

- A set of protocols
- Python and C++ implementations
- An API for integration
- Tools for controlling, monitoring and data acquisition
- A community!

```
cdtp.md 4.97 KIB Code Preview
```

Constellation Data Transmission Protocol

- Status: draft
- Editor: The Constellation authors

The Constellation Data Transmission Protocol (CDTP) defines how data is transmitted from a sending host to a receiving host.

Preamble

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHOULD", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

Goals

This specification is intended to formally document the names and the structure of the data transmission between two hosts of the Constellation framework. This document also defines the structure of the sending and receiving messages with cargo payload to and from other CDTDP hosts.

```
8 */
9
10 #pragma once
11
12 #include <cstdint>
13
14 namespace constellation::protocol::CDTP {
15
16     /** Possible conditions of a run */
17     enum RunCondition : std::uint8_t {
18         /** The run has concluded normally, no other information has been provided by the sender */
19         GOOD = 0x00,
20
21         /** The data has been marked as tainted by the sender */
22         TAINTED = 0x01,
23
24         /** The receiver has noticed missing messages in the sequence */
25         INCOMPLETE = 0x02,
```

```
48         super().__init__(
49             name=name,
50             group=group,
51             cmd_port=cmd_port,
52             hb_port=hb_port,
53             mon_port=mon_port,
54             interface=interface,
55         )
56
57         self.log_satellite = self.get_logger('satellite')
58
59         self.run_identifier: str = ""
60         self.run_degraded: bool = False
61         self.config = Configuration({})
62
63         # give monitoring a chance to start
64         time.sleep(0.1)
65
66
67
68
69
```

Constellation Observatory

List Filters

Level	TRACE	Sender	Level	Topic	Message
2024-09-13 13:37:01		Sputnik.test	DEBUG	SATELLITE	Internal configuration: 0 settings
2024-09-13 13:37:01		Sputnik.test	INFO	FSM	Calling initializing function of satellite...
2024-09-13 13:37:01		Sputnik.test	INFO	SATELLITE	Configuration: 0 settings
2024-09-13 13:37:01		Sputnik.test	STATUS	FSM	New state: initializing

Constellation: edda, Satellites: 0

Subscriptions: Global Level: DEBUG



Brought to you by:

DESY, Prevas AB, Lund University, Universität Bonn, Universität Hamburg, AGH University of Kraków



Introducing Constellation

- Project goals:
 - **Ease of use**
 - **Simplicity of integration**
 - **Flexibility of application**
- Participants are called **satellites**
 - Operation is governed by a **finite state machine**
 - Satellites can operate **autonomously** without active user interface
- Independent implementations in Python & C++





State of the Celestial Sphere

- Continuous development of Constellation over the past years
- Releases named after IAU-designated constellations
- Six preview releases with major features added, from 0.1 (Crux, 11/2024) *Protocols, satellite library, first satellites*; to ...
0.6 (Triangulum Australe, 09/2025) *Data transfer, Satellite templates*

Latest Release v0.6.1

License EUPL 1.2

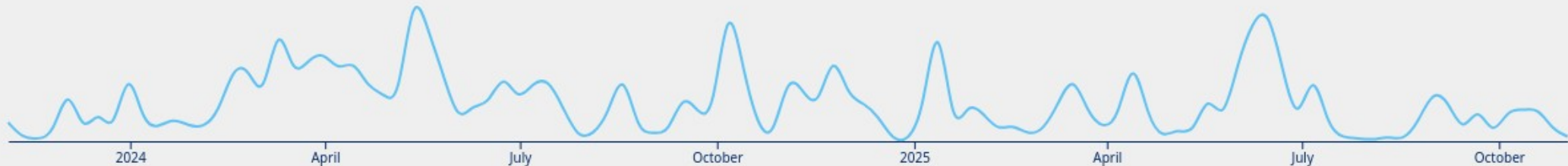
coverage 86.15%

pipeline passed

REUSE compliant

DOI 10.5281/zenodo.15688357

repo status Active



5265 commits | Last commit ≈ 1 week ago | 6 stars | 8 forks

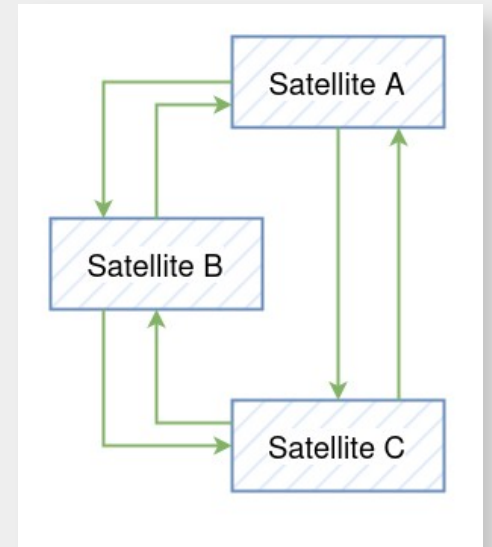


The Autonomous Control & DAQ System.

What does that even mean?

Autonomous & Collaborative Operation

- Constellations operate in decentralized manner, no central server is required
- Communication of state information between components happens through **heartbeats** containing:
 - The current state of the sender
 - The time interval until the next heartbeat
 - A set of flags (next slide...)
- Two different anomaly states:
 - **ERROR:** satellite failed; instrument doesn't react, ...
 - **SAFE:** reaction to detecting issue with *other* satellites
 - Main difference: SAFE is controlled shutdown, ERROR is unknown hardware state



Autonomy: Satellite Roles

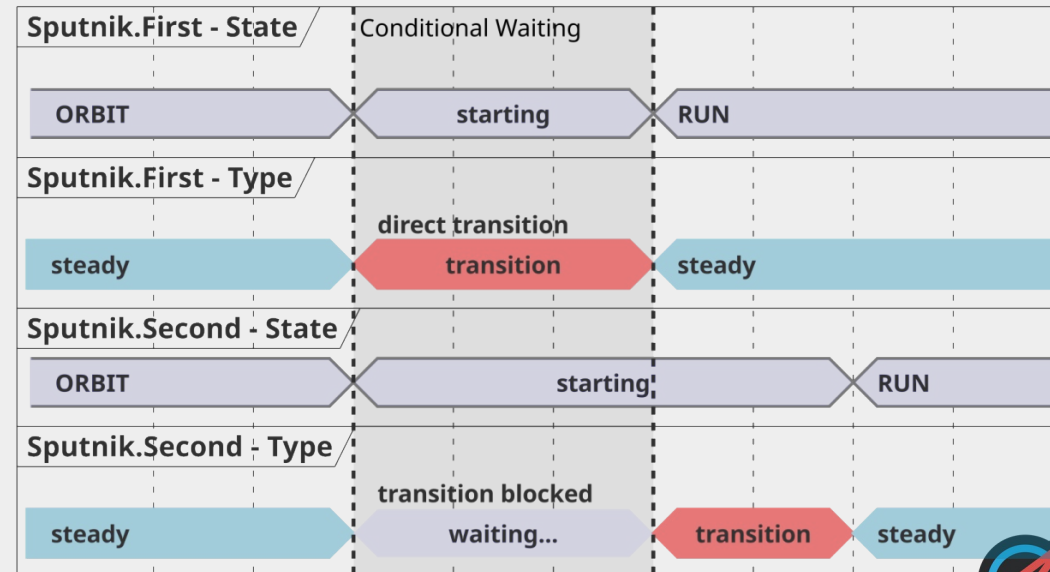
- Flags in heartbeats allow to control, how other satellites are supposed to react
That guy is gone, should I stop data taking or just ignore it and continue?
- Set through satellite **role in configuration**, provide different levels:
 - **ESSENTIAL:** This satellite is crucial, if anything happens to it, stop all data taking!
 - **DYNAMIC:** This satellite is *kind of* important, but if someone shut it down on purpose it's fine and we continue.
 - **TRANSIENT:** This satellite is dispensable to data taking, lost log problems and mark the data taking run as *degraded* if it disappears.
 - **NONE:** This satellite can appear & disappear without *making a fuzz*



Autonomy: Conditional Transitions

- Sometimes satellites require to be initialized in a specific order
(TLU sending clock only after configuring it, anyone?)
→ directed acyclic dependency graph
- Heartbeats contain current state of sending satellite – *we can use that!*
 - Satellite receives configuration which tells it what to wait for, e.g.

```
_require_starting_after = ["Sputnik.First"]
```
 - Satellite waits for all remotes to enter the given state
 - Then, satellite progresses through its own transition
- Asynchronously and autonomously progress from steady state to steady state
- No controller needed for supervising the order of action

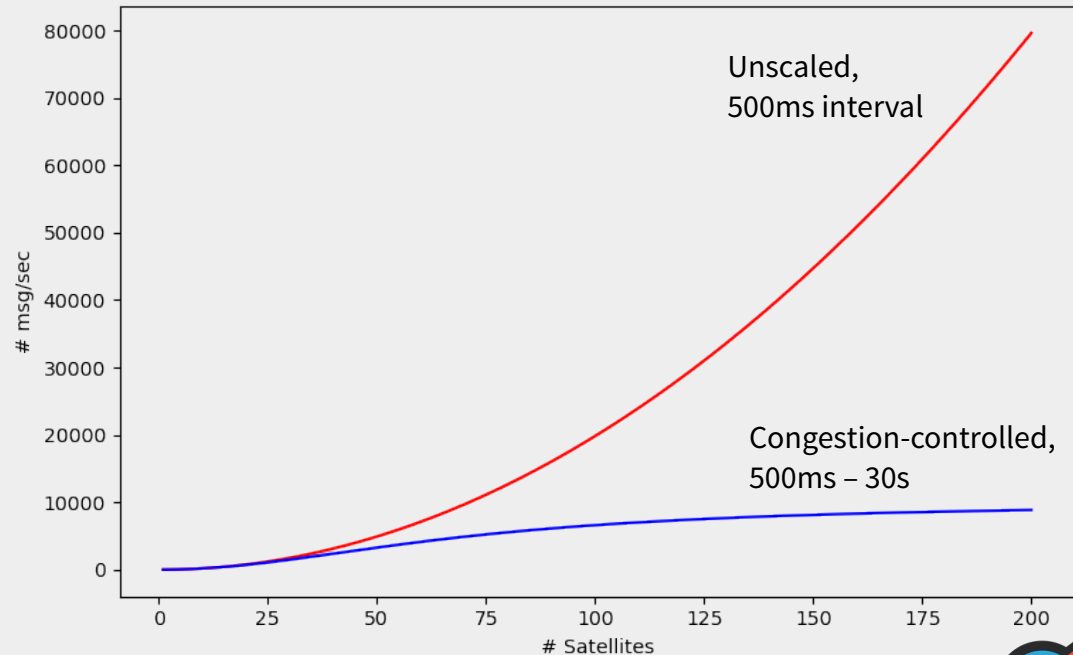


Autonomy: Congestion Control

- Constellation's autonomy is based on heartbeat messages between *all* nodes – this scales quadratically – problematic for many satellites
- Solution: heartbeat messages have variable **sender-defined intervals!**
Use this for congestion control:

$$\Delta t = \min \left(\Delta t_{\max}, \max \left(\Delta t_{\min}, \Delta t_{\min} \cdot \sqrt{N} \cdot \ell \right) \right)$$

- Vary interval (autonomously!) between minimum & maximum value
- So-called **extrasystoles** sent out immediately at state changes – no change in reactivity!



A Primary Goal is Ease of Use.

What's up with that one?



...not one but **three** manuals!

Constellation

Autonomous Control and Data Acquisition System

Constellation is a control and data acquisition system for small-scale experiments and experimental setup with volatile and dynamic constituents such as testbeam environments or laboratory test stands.

Get Started

Concepts

[See Application Developer Guide →](#)

[See Framework Reference →](#)

Autonomous

Constellation operates without a central server, satellites exchange heartbeats to keep in touch.

Flexible

Automatic network discovery of satellites make it easy to add and remove satellites on the fly.

Fast Integration

The finite state machine and satellite interface are designed for fast and easy integration of devices.

Robust

Constellation is based on widely adopted networking libraries such as [ZMQ](#) and [MsgPack](#).



Constellation



Constellation Operator Guide, structured in four parts:

- **Getting Started:** Installation and initial setup of Constellation
- **Tutorials:** Practical examples, starting & controlling a single satellite, more complex examples & setups coming
- **Concepts:** detailed explanation of the framework & its structure. This is not the technical documentation, just describes functionality and helps in developing an understanding of the system.
- **How-To Guides:** concise answers on how to achieve a specific goal: Logging to Mattermost; setting up a Grafana Dashboard; ...





Application Developer Guide

- **Introduction** on development tools
- **Tutorials** on implementing new satellites (C++ and Python)
- **Instructions** on extending satellites with more functionality
- **How-To Guides** e.g. for porting satellites from EUDAQ

```
ExampleSatellite(std::string_view type, std::string_view name) : Satellite(type, name)
{
    support_reconfigure();
}
```

and the corresponding transition function `reconfiguring(const config::Configuration& config)` needs to be implemented.

The payload of this method is a partial configuration which contains only the keys to be changed. The satellite implementation should check for the validity of all keys and report in case invalid keys are found.

Setting Status Messages

In addition to its state, each satellite also exposes a status message that provides additional human-readable information on how this state was reached.

See also

More information on the state and status message can be found in the [Operator Guide's section on satellites](#).

From C++, the status message can be configured from transitional states as follows:

```
void ExampleSatellite::launching() {
    submit_status("Successfully launched, 16 channels ramped up");
}
```

This status will be adopted by the satellite at the end of the transitional state. It should be noted that consequently, it is not possible to set multiple status messages during a single transitional state.

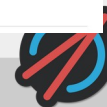




Framework Development Guide

- Technical documentation, contains technical details required for implementing Constellation libraries in different languages
- Contribution guidelines
- Documentation of release procedure
- Satellite library implementation guidelines
- Python & C++ API documentation
- The Protocols

Command	payload	verb reply	payload reply
<code>get_name</code>	-	Name of the Satellite	-
<code>get_version</code>	-	Constellation version identifier string	-
<code>get_commands</code>	-	Acknowledgement	List of commands as MsgPack map/dictionary with command names as keys and descriptions as values
<code>get_state</code>	-	Current state (as string)	-
<code>get_status</code>	-	Current status	-
<code>get_config</code>	-	Acknowledgement	Satellite configuration as flat MsgPack map/dictionary
<code>get_run_id</code>	-	Current or last run identifier (as string)	-
<code>initialize</code>	Satellite configuration as flat MsgPack map/dictionary	Acknowledgement	-
<code>launch</code>	-	Acknowledgement	-
<code>land</code>	-	Acknowledgement	-
<code>reconfigure</code>	Partial configuration as flat MsgPack map/dictionary	Acknowledgement	-
<code>start</code>	Run identifier as MsgPack string	Acknowledgement	-
<code>stop</code>	-	Acknowledgement	-
<code>shutdown</code>	-	Acknowledgement	-



Tutorials!

- First at [13th Beam Telescopes & Testbeams WS](#)
- “Implement your own Temperature Monitor”
 - Raspberry Pi with temperature sensors
 - Global Grafana dashboard on the projector
 - Everyone writes a satellite, connects it to a common Constellation
- Will repeat at upcoming BTTB14
- Let us know if you would like to run a tutorial at your workshop!



Flexibility of Application you say.

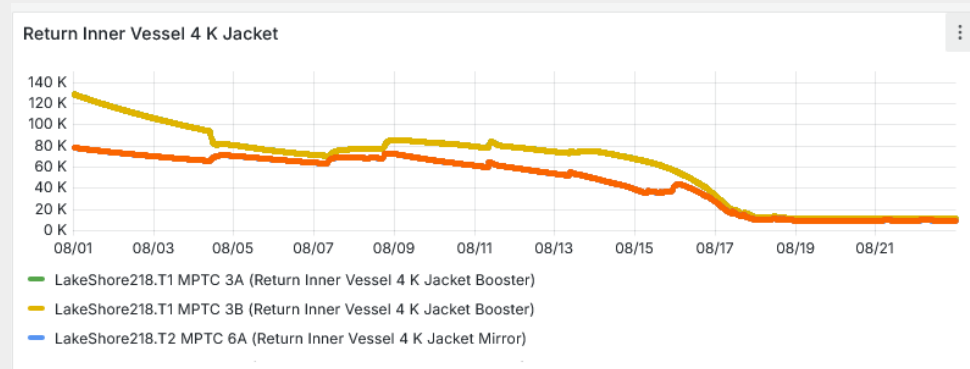
Who is actually using this?

*I mean, apart from
DESY II Testbeam Facility,
Fermilab Testbeam Facility,
CMS ETROC Testing,
Beamline4Schools and others...*



Monitoring the MADMAX Cryostat

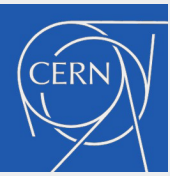
- Magnetized Disk and Mirror Axion eXperiment (MADMAX) is a dark matter experiment searching for axions in mass range $40 \mu\text{eV}$ to $400 \mu\text{eV}$
- Dielectric haloscope with cryostat to reach low noise temperature
- Constellation used to continuously monitor:
 - cryogenic temperatures
 - gas pressures
 - liquid helium level
- Mattermost integration used to trigger alarms
- Continuous operation over many weeks
- Plans to extend to booster & receiver systems



Section of MADMAX Grafana Dashboard (David Leppla-Weber)

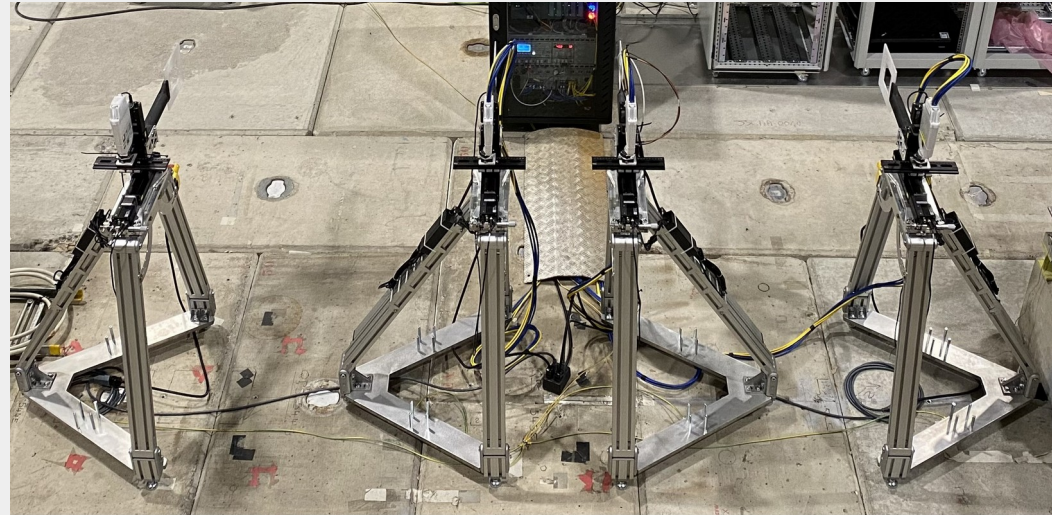


Characterizing Bent Crystals for Beam Manipulation



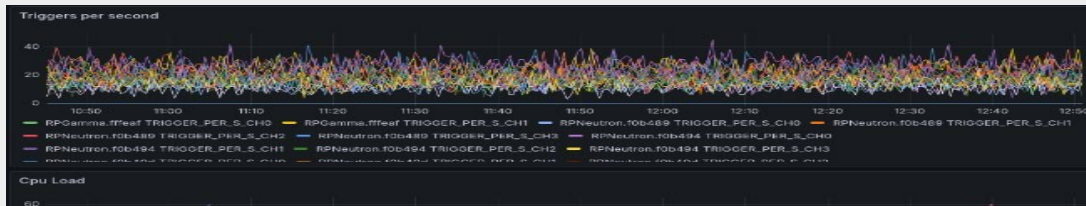
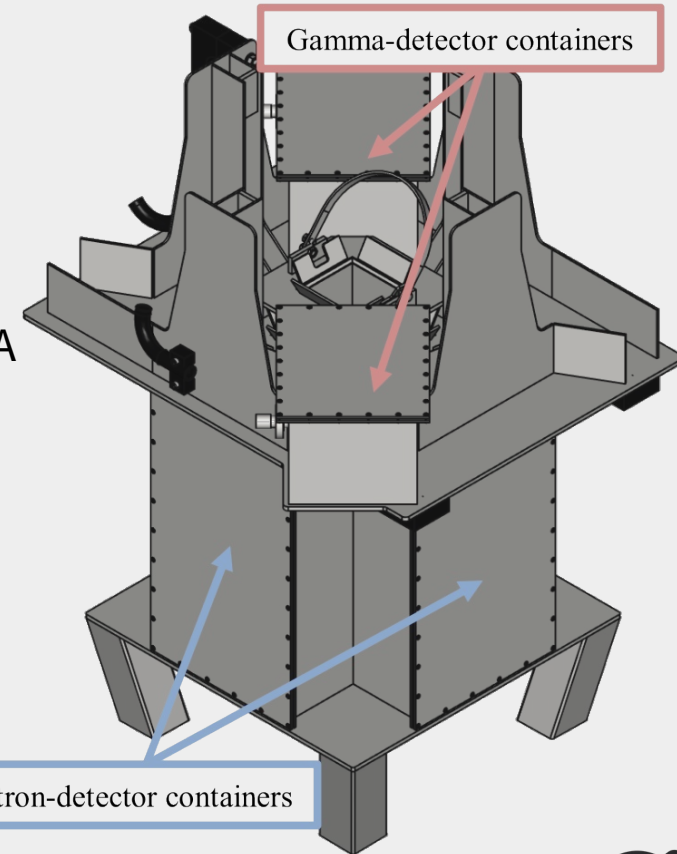
Justus Braach

- Characterization of bent crystals for beam manipulation
 - Crystal-assisted collimation for heavy-ion beams
 - Future fixed-target experiments in HL-LHC
- BIPXL: Dedicated long-lever-arm beam telescope at SPS North Area
 - 14 motion stages (crystal, telescope)
 - 6 Timepix3 detectors
 - Scintillator-PMT pairs
 - Crystal on four-axes goniometer
- Constellation development & setup within four weeks
- Stable operation during commissioning



Characterizing Spent Nuclear Fuel

- Prototype measurement station for spent nuclear fuel characterization at the Swedish Nuclear Fuel and Waste Management Company (SKB)
 - 24 Red Pitaya STEMLab 125-14
 - 1 Red Pitaya SIGNALab
 - 2 CAEN Power supply crates
- 98 analog detector channels processed in real-time on RP FPGA
- Triggered signals cached in RAM, amplitude & timestamp read by satellites & transferred via 10G fiber optical cable
- System handles continuous rates of 2 million events/sec/ch; corresponding to roughly 800 MB/s data written to disk
- Monitoring with Influx/Grafana dashboard



Towards Constellation 1.0

Now, what's the plan here?



Roadmap for Constellation 1.0

- At this point, Constellation is almost feature-complete
- Reference paper has been submitted to NIMA
- Roadmap to stable release:
 - 0.7 *Reticulum*
Configuration dictionaries, YAML support, Telemetry UI
 - 0.8 *Caelum*
CHIRP2, Python satellites in Docker+Flatpak
 - 1.0 *Corona Australis*
Stable API, foreseen **early 2026**

• Start adopting now!

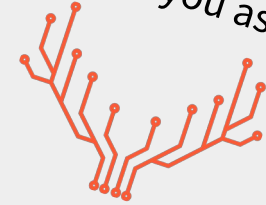
(your favorite beam telescopes, TLUs and power supplies likely are supported already!)



Summary & Outlook

- **Constellation** control & DAQ system **is maturing**
 - Several preview releases over the past year
 - Nearing release of version v1.0 with stable API
- Several, quite **diverse applications** already operating
- **Paper recently submitted** to NIMA
- Come to **BTTB 14 in Mainz** for a tutorial and a *fun challenge!*
- **Get involved** as developer, as adopter, as user – join us:
 - Mailing list: lists.desy.de/sympa/info/edda
 - Website / documentation: constellation.pages.desy.de
 - Mattermost: mattermost.web.cern.ch/constellation

Yes, of course there is a
Caribou Satellite
– why do you ask? 😇



DESY Training on Experimental Concepts & Techniques



- Yearly instrumentation school at DESY mainly targeting *early PhD students*
- Two weeks of lectures & **laboratory courses**
- Learn how to...
 - Work in a clean room
 - Characterize a sensor
 - Operate detectors at the testbeam
 - Build your own particle detector
 - Explore control systems & mechanical properties

First edition 8–19/6/2026!
Application opens 01/12/2025
Deadline 01/03/2026



