GSoC 2025 Project Proposal for

Constellation: Extending User Interface

(Duration: 350 hours)

ADITYA SAROHA

SJC INSTITUTE OF TECHNOLOGY,
KARNATAKA, INDIA

+91 8700857698
adityasaroha456@gmail.com
aditya-saroha-73b866268
Aditya-138-12

Table of Contents

Table of Contents

1. Basic Information 2

1.1. Personal Details 2

1.2. About Me 2

2. Project 3

2.1. Abstract 3

2.2. Motivation 3

2.3. Why HEP Software Foundation? 4

2.4. Proposed Deliverables 4

2.5. Plan of Action 5

2.5.1. Familiarization, Setup & Restructuring 6

2.5.2. Designing the User Interface 8

2.5.3. Dashboard Design and Diagram Integration 9

2.5.4. Configuration & Grafana Implementation 10

2.5.5. Qt5/Qt6 Compatibility & Optimization 10

2.5.6. Documentation & Testing 11

2.6. Timeline 11

3. Commitments 13

4. Contributions 13

4.1. Evaluation Tasks 13

4.2. Merged PR's 13

5. Post GSoc 14

1. Basic Information

1.1. Personal Details

Name: Aditya Saroha
Major: Computer Science & Engineering
Degree: Bachelor of Engineering (B.E.)
Year: 3rd
Institute: SJC Institute of Technology, Chikkaballapur, Karnataka, India
Resume: **ADITYA_SAROHA_CV_INFN.pdf**
Time Zone: Indian Standard Time (UTC + 5.30)

1.2. About Me

I am Aditya Saroha, a 3rd-year undergraduate at the SJC Institute of Technology, pursuing Computer Science & Engineering. I have experimented with various domains like Machine Learning, Computer Vision, IoT, Web Development, etc. I love writing reusable code and developing small products for others to use.

I have previously worked with the flask-api to create several API's, and developed a few websites. For the front end, I am familiar with ReactJS, NextJS, and ThreeJS. I am also experimenting with 3D game development using C++ and Vulkan as my graphics API.

- I have developed a Data Visualization platform named SJCIT-CONNECT for my college using ReactJS in the front end and Firebase along with Flask API in the backend. Students and Faculty can create accounts, upload their achievements, etc. and the HoD(Head of the Department) can generate reports, visualize the data in real time with many more features.
- I am voluntarily contributing to the CERN ROOT project, implementing features, fixing bugs, improving documentation, adding tests etc.

2. Project

2.1. Abstract

The constellation framework serves as a coordination tool for experimental physics laboratory setups, enabling the management of diverse instruments across a local network. While it excels in controlling instruments via the MissionControl interface and logging the messages through Observatory, it currently lacks a fast solution for real-time telemetry visualization on the go, GRAFANA influx is there, but setting it up is a complicated process for a newbie(like a undergraduate physics student). This Google Summer of Code project aims to develop a Qt-based desktop application–a telemetry dashboard–to address this gap which will be a faster alternative, with just a few clicks the researcher is equipped with the valuable insights!
The proposed user interface will automatically discover available satellites using the CHIRP, subscribe to their telemetry data (STAT topics) via CMDP protocol, and allow users to configure a customizable grid of real-time diagrams (eg. line/bar charts) displaying metrics such as CPU usage, temperature, or other desired metric. The dashboard configuration will persist across sessions using Qt's config system, ensuring a seamless user experience. Designed as a lightweight listener with no data storage or control functionality, the application will prioritize on-the-fly visualization, Qt5/6 compatibility, and a cohesive design aligned with existing Constellation GUIs. This project enhances Constellaltion's utility by providing physicists with an intuitive, real-time window into their experimental setup's performance.

2.2. Motivation

I was always curious what happens inside the LHC, as the protons collide with each other, with this in mind my love for Physics with CS increased and I was introduced to the world of Open source where you can contribute to these amazing projects, help researchers worldwide resolve their issues, help them. Likewise I landed on this amazing project of Constellation, helping researchers find something which is not yet discovered just feels great, so this is my main motivation to apply for this amazing project.

2.3. Why HEP Software Foundation?

I'm applying solely to HSF for GSoC 2025, captivated by LHC research since the Higgs Boson discovery. As a developer and physics enthusiast, I'm drawn to Constellation's role in experiment monitoring—vital for data quality when costs can hit 80k€/week or time is scarce (eg. one week/year for measurements). While Grafana/Influx excels, its setup is complex for beginners, like undergrads who might struggle for hours. A Qt dashboard offers a fast, simple alternative—start it, click a few buttons, and see key metrics like instrument temperature instantly, unlike tracking down a Grafana IP. This ease is critical during setup, saving time for science. I'm excited to collaborate with DESY via HSF, merging my open-source passion with impactful physics tools.

2.4. Proposed Deliverables

- The application will be a Qt Desktop based application, which can be started and closed on the go. It will be an on-the-go visualization application and hence will not have the features of data storage or controlling the satellite. First let's talk about the design of the application.
- The design will be similar to existing modules like the Observatory/MissionControl, but more focus will be on keeping the design of the dashboard much like a grafana dashboard:
  - The dashboard will have a header which will have a toolbar instead of filtering to keep it more interactive, so there will be dropdowns for displaying the discovered satellites, and STAT topics for the discovered satellites.
  - One grouped button would be there for selecting the diagram like bar/line/time series charts.
  - Finally there will be a final button named something like this "Add to dashboard" which will confirm the user selection and will add a graph in the dashboard.
  - Now this graph which is added will be resizable and movable within the grid, and this information will be added in a config file so that whenever a user closes the application and

restarts it, the graphs should be present in there desired location i.e. which is set by the user.

- While designing the User interface, I will also maximize my approach to reuse the existing UI elements from the MissionControl and Observatory, by this I mean I will restructure them and make them as reusable Qt widgets.
- Now upon the start the satellites would be discovered using the CMDPListener, once the satellite is discovered we can make use of the metadata like host_id, identifier etc.
- Once the satellites are discovered, the system then should connect to them via the CMDP(Constellation Monitoring Distribution Protocol) to receive telemetry and status updates.Each discovered CMDP host is identified by the %x03 service ID and publishes metric data under STAT/ topics, and then request a list of metrics via the notification messages(STAT). But a derived class, StatListener, will be implemented to handle metric subscriptions dynamically, filtering and parsing MessagePack-encoded telemetry data. The protocol ensures efficient data distribution via PUB/SUB sockets (29/PUBSUB) while enforcing message integrity with structured headers and timestamps.

2.5. Plan of Action

I will start by understanding the structure of the codebase in detail and analyze where all the new files should be kept for the new application and as well for the reusable Qt widgets. I will understand the code in the CMDPListener.hpp, and analyze the CMDP protocol in detail. I will also scan through the various codebase of the MissionControl and Observatory to find the UI elements which can be reused.

I have already started work on the easier items, as stated above, restructuring of the UI elements, the one I am working now to make it into a reusable Qt widget is the placeholder for the cnstlnName. Also I will establish direct contact with the mentors, we will then again go through the finalized features, making the reusable Qt widgets will consume a significant amount of time and research.

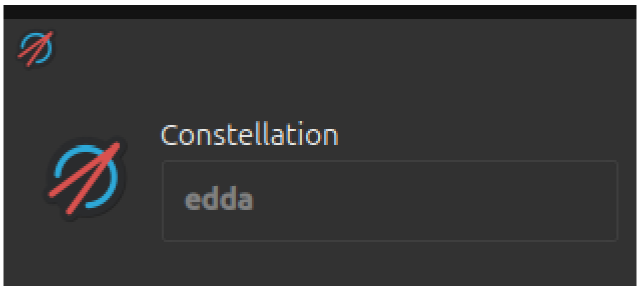
2.5.1 Familiarization, Setup & Restructuring

Expected due date: June 15, 2025
Objective: Gain deep understanding of the Constellation codebase, set up my development environment, and restructure Qt sources into reusable Qt widgets.
Deliverables:

- A working local build of Constellation with MissionControl and Observatory for reference.
- Will understand in detail the key classes like Manager.hpp, CMDPListener.hpp.
- Will add doxygen documentation in the modules of MissionControl and Observatory (Optional).
- A refactored structure of Qt GUI sources, separating shared design elements. In short making reusable Qt widgets.

I will start by looking into the Constellation repo. As a first step, I will set up my development environment by cloning and building the Constellation repository (which I have already tested successfully), allowing me to focus immediately on reading the helper classes. Then I'll dig into the docs and tutorials (single_satellite, missioncontrol) to map out how CHIRP discovers satellites and CMDP handles data.

Next, I'll analyze the Qt GUI code. I'll identify reusable components (eg. headers, styles, the way logos are displayed) and propose a cleaner structure— a shared ui/ directory for common widgets, for example the cnstlnName, here is an image showing the possible reusable UI element.



I'll sketch out how my dashboard fits in as a "listener" (Listener), ensuring it aligns with the ecosystem. By mid-June, I'll have a solid

foundation and a plan to reuse the Observatory's/MissionControl header styles.
Also this would be optional, the header files of the MissionControl and Observatory are well documented, as it states in detail what the function does, but it would be a good idea to add more detailed comments in the source files, like for example

```
/**
 * @brief Constructs the MissionControl main window
 *
 * @param controller_name Name of the controller instance
 * @param group_name Name of the Constellation group to connect to
 *
 * @details Initializes the Mission Control GUI with:
 * - Controller name display
 * - Constellation group connection
 * - Satellite state monitoring
 * - Run control functionality
 * - Configuration management
 *
 * The constructor:
 * 1. Sets up the UI from the .ui file
 * 2. Initializes logging and status displays
 * 3. Configures the satellite connection view
 * 4. Sets up timer-based updates
 * 5. Restores previous window state
 */

MissionControl::MissionControl(std::string controller_name,
std::string_view group_name)
    : runcontrol_(std::move(controller_name)),
      logger_("GUI"),
      user_logger_("OP"),
      run_id_validator_(QRegularExpression("^[\\w-]+$"), this),
      config_file_fs_(&config_file_completer_) {
}
```

2.5.2 Designing the User Interface

Expected due date: July 08, 2025
Objective: Implement satellite discovery and telemetry subscription using existing Constellation protocols.
Deliverables:

- A StaticListener class derived from CMDPListener.cpp to subscribe to STAT/topics.
- A basic Qt widget listing discovered satellites and their STAT topics (only for testing).
- Logs confirming successful subscription to telemetry streams.

Here, I'll build the backbone of the dashboard. I'll start with CMDPListener to auto-discover satellites when the app launches—looping through the network and collecting their IDs. Then, I'll extend CMDPListener.hpp into a new StatListener class. It'll subscribe to STAT/ topics (eg. STAT/CPU, STAT/TEMP) via CMDP (cmdp.html), parsing incoming messages into a usable format (maybe a QMap<QString, QVariant> for key-value pairs).
The StatListener.hpp would look something like this,

```
// StatListener.hpp
class StatListener : public CMDPListener {
public:
   StatListener(std::string_view log_topic);
   std::vector<std::string> getDiscoveredSatellites() const;
   // Callback for telemetry data, using Metric class
   void onTelemetryReceived(const std::string& satellite,
                      const std::string& topic,
                      const Metric& value);
protected:
   void topics_changed(std::string_view sender) override;
   void sender_connected(std::string_view sender) override;
private:
   std::map<std::string, std::set<std::string>> satellites_stat_topics_;
};
```

On the UI side, I'll create a simple QListWidget(Just for testing) to display satellites and their topics, connecting it to StatListener via Qt signals/slots. I'll test this with a mock satellite sending dummy data (eg. random CPU values) to ensure the pipeline works. By July 1, the app will "hear" telemetry—ready for visualization.

2.5.3 Dashboard Design and Diagram Integration

Expected due date: August 25, 2025
Objective: Design the main UI and integrate the real time diagram widgets using QtCharts, for telemetry display, make the grid below the header and allow the charts to be resizable. Store the positions in a config file.
Deliverables:

- A Qt header with dropdowns for satellite, topics, and diagram type, plus an "Add to dashboard" button.
- A grid layout which will host all the diagrams selected by the scientist.
- Reused UI elements from MissionControl/Observatory.

Now it's time to make the dashboard look good and work smooth. I'll design a header inspired by the Observatory—a QToolBar instead of a filter with a dropdown widget for selecting satellites, STAT topics, and diagram types (like line, bar). An "Add to Dashboard" QPushButton will lock in the choice and thus add the diagram in the grid. Below, a QGridLayout will hold the diagrams.
For diagrams, I'll use QtCharts to render live data—eg. a Line graph for CPU usage or a bar graph for temperature. I'll connect these to StatListener signals, updating whenever new data arrives. By August 25, users can add and watch a couple of live graphs ticking along.

For dynamic updates, I'll connect QtCharts to StatListener telemetry callbacks. Each chart (eg. QLineSeries for CPU usage or any other STAT's) will maintain a data buffer (eg. QVector<QPointF> or any other type) storing timestamp-value pairs. When StatListener invokes its onTelemetryReceived callback with a new Metric object, I'll extract the timestamp and value, append them to the buffer via QLineSeries::append(), and trigger a redraw with QChart::update().
To ensure smooth performance with high-frequency data (this is optional, and will be taken forward after consulting with the mentors) (eg. 100 Hz or any other specified type), I'll use a QTimer (set to ~100 ms intervals or relevant) to batch updates. The chart's axes (eg. QValueAxis) will adjust dynamically using setRange() to reflect the latest data span, keeping the display responsive and real-time.

2.5.4 Configuration & Grafana Implementation

Expected due date: September 05, 2025
Objective: Add persistence for dashboard layout and polish the UI for a grafana like design.
Deliverables:

- Dashboard config saved, and resumed when the application is started again.
- Error messages when either no satellites are found or signal is lost.
- Make the UI look like the grafana dashboard, some more minor UI enhancements.

This phase will be dedicated to enhance the user experience. I'll save the dashboard state—storing each diagram's satellite, topic, type, and grid position in a Qt config file. On restart, the grid will load exactly as left. For flexibility, I'll add drag-and-drop to the QGridLayout using QDrag and QDropEvent, letting users place the diagram.

I'll also polish it up: a QMessageBox for errors (eg. if CHIRP finds no satellites), a "Remove" button per diagram, and maybe basic graph tweaks (eg. color picker if time allows, otherwise this would be taken after the GSoC period). By September 05, the dashboard will feel reliable, ready for real lab use.

2.5.5 Qt 5/ Qt 6 Compatibility and Optimization

Expected due date: September 20, 2025
Objective: Ensure dashboard runs on both Qt5 and Qt6 while optimizing the performance.
Deliverables:

- A codebase compiling clearly on Qt5 and Qt6 (tested on both).
- Performance tweaks for high-frequency telemetry.
- Compatibility fixes and bottlenecks will be resolved.

I'll set up dual builds (Qt5 on one VM, Qt6 on another) and hunt down deprecated APIs. I'll use #pragma once QT_VERSION or other relevant versions blocks where needed for better compatibility.
For optimization, we will refactor some of the codebase of the new application to better utilize the memory, for example to use Qt::UniqueConnection to avoid redundant updates.

2.5.6 Documentation & Testing

Expected due date: October 20, 2025
Objective: Fully document the dashboard and test it end-to-end for submission, also write tests for the same.
Deliverables:

- Unit tests for StatListener, diagram updates, and config saving.
- End-to-end tests simulating multiple satellites with telemetry.
- Writing doxygen documentation for the header as well as for the source files, also a user guide (markdown) explaining setup and usage using sphinx.

I'll write unit tests—eg. checking if StatListener parses STAT messages right or if diagrams are at the same position where they are supposed to according to the config file. If bugs arise, they will be resolved.
Documentation will cover installation, code structure, and a quick "How to Use" section—mirroring Constellation's tutorial style. Also a detailed documentation of the headers/source/build files.

I'll submit everything by October 20, leaving some time for mentor feedback before the final evaluation deadline (November 10). This delivers a complete, tested package!

2.6 Timeline

Days	Plan
Community Bonding Period	
May 8 – June 15	• Read the documentation thoroughly • Read the details of the codebase • Join Constellation weekly meetings with mentors • Participate in group discussions • Finalize on any other features
June 15 – July 8	• Use CHIRP's Manager.hpp to auto-discover satellites at launch. • Create StatListener (extending CMDPListener.hpp) to subscribe to STAT/ topics (STAT/CPU, STAT/TEMP). • Parse telemetry messages into QMap<QString, QVariant>. • UI: QListWidget (test only) to display satellites & topics, linked via Qt signals/slots. • Test with a mock satellite sending dummy telemetry data.
Midterm Evaluation (July 14)	
July 9 – August 25	• Header Design: Use QToolBar with dropdowns for satellite, STAT topics, and diagram type (like line, bar). • Dashboard Logic: "Add to Dashboard" (QPushButton) locks choices & adds diagrams to QGridLayout • Live Data Integration: Use QtCharts for telemetry (eg. Line graph for CPU, Bar graph for temperature • Real-time Updates: Connect StatListener signals to update charts dynamically.
August 26 – September 5	• Save & Load Layout: Store diagram settings in a Qt config file. • Drag & Drop: Enable repositioning in QGridLayout via QDrag & QDropEvent. • Error Handling: Use QMessageBox for CHIRP errors (eg. no satellites found).

	● "Remove" button per diagram. ● Possible graph tweaks (e.g., color picker if time allows).
September 6 – September 20	● Dual Builds: Test on Qt5 & Qt6. ● API Compatibility: Use #ifdef, QT_VERSION to handle deprecated APIs. ● Refactor code to improve memory usage. ● Use Qt::UniqueConnection to prevent redundant updates.
September 21 – October 20	● Write unit tests to verify telemetry parsing and config-based diagram positions. ● Conduct end-to-end testing. ● Document installation, code structure, and usage. ● Provide detailed header/source/build file documentation. ● Submit everything by October 20, allowing time for mentor feedback.
Final Evaluation (November 10)	

3. Commitments

I am flexible with my working hours and am open to working at night as well. This summer, I have my University exams from June till August. Although my college has already started, I am willing to dedicate 5–6 hours per day during that period. Additionally, my end-semester exams will not be a source of pressure for me. I check my emails multiple times a day, so communication via email would be convenient. You can expect my response within 24 hours.

4. Contributions

4.1. Evaluation tasks:

I made a demo Qt application for the warm exercise. It basically has a chart, and 2 buttons, one button changes the chart and the other resets

it, generated the documentation using doxygen and hosted it on github pages, applied the clang formatting.
Link: Demo Qt Warm Up Exercise

4.2. Merged PR:

I have made all the imports within the python files of the codebase sorted using the isort, also made a CI job for the same and added it in the pre-commit hook.
Link: Using isort in the codebase

5. Post GSoC

Post the GSoC period, I will be continuing contributing to this amazing project and implementing more features requests for **Constellation** that may have not been covered in GSoC in the short time interval. I will also try to provide consistent patches and minor bug fixes to the projects. I will also try to contribute to the community by taking part in development other than the scope of this project. I am truly grateful to get the opportunity to contribute to this amazing project which is at the forefront of technical innovation.