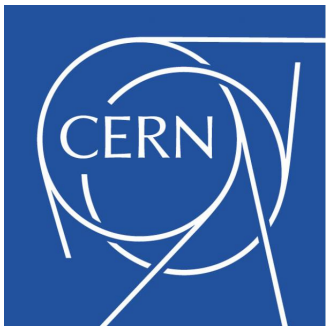
Implementation of a Slack chatbot to support ProtoDUNE workflow

Neil Micallef

Supervisor(s): Manuel Jesus Alonso Rodriguez, Giovanna Lehmann Miotto



CERN Summer Studentship

EP-DT-DI

August 2018

Contents

1 Introduction

This project was implemented within the CERN EP-DT-DI section as part of the CERN Summer Studentship. The task involves the ProtoDUNE experiments currently taking place at CERN. ProtoDUNE stands for Prototype Deep Underground Neutrino Experiment. Essentially, these experiments are performed on two large neutrino detectors, one being single-phase (protoDUNE-SP) and the other dual-phase (protoDUNE-DP). The single-phase detector uses only liquid argon, whilst the dual-phase will make use of dual-state argon in liquid and gas form. An image of one of these detectors can be seen in the figure below, with a person on a lift next to it for scale[1]:

Figure 1: One of the 10x10x10m detector cubes used in protoDUNE

As stated previously, the detectors are Liquid Argon Time Projection Chambers, similarly to the ICARUS detector[2] used between 2010 and 2014 at the INFN Gran Sasso Laboratory in Italy. Six rectangular Anode Plane Assembly(APA) modules line the detector, which aim to detect signals emitted by passing particles. Important parts of the model include the cryostat and the field cage, which monitor the temperature of the detector, as well as keeping the electric field constant so as not to introduce noisy signals into the system. Within this complex system are a multitude of devices which relay live data from the detector onto a server, and may be adjusted accordingly by authorized personnel.

[1]https://www.symmetrymagazine.org/sites/default/files/images/standard/protoDUNEconstruction.jpg (accessed 28/07/2018)

[2]https://home.cern/cern-people/updates/2016/06/new-wings-give-icarus-flight-second-neutrino-hunt (accessed 28/07/2018)

2 Motivation

Various operations form part of the day-to-day proceedings surrounding the single and dual-phase detectors. For the sake of scope, this report will only speak about the EP-DT-DI section's involvement in the experiment. Slack is the main means of communication for the team when speaking about the detector. As it stands, Slack can be used by members to inform others when reserving a particular partition of the system, as well as to obtain any other form of information not readily available on the server. Thus, the Slack channel can become very time-consuming and cluttered with messages as follows:

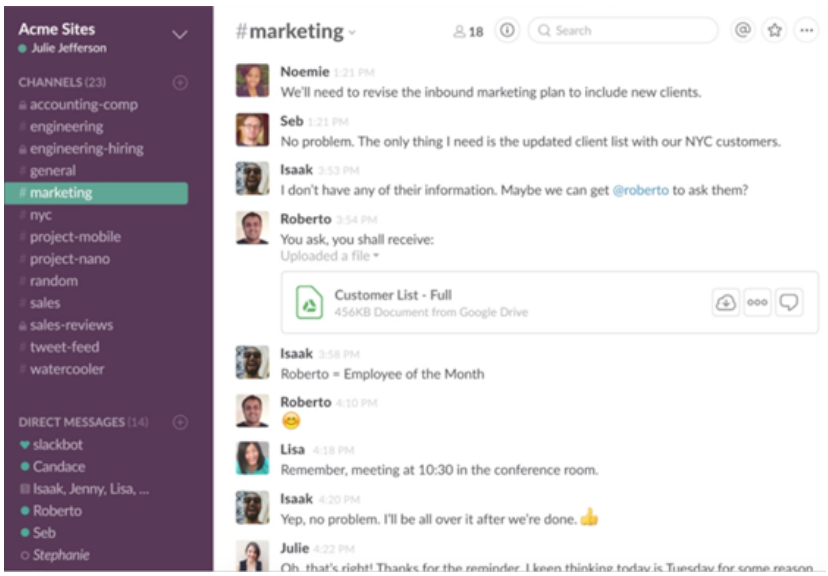


Figure 2: Cluttered Slack Channel

The goal of the task is to introduce a chatbot, which is allocated its own channel into the team's Slack workspace. The bot can post messages when detecting changes in the server instead of the workers. It would also keep track of important information such as partition reservations, as in the figure below:

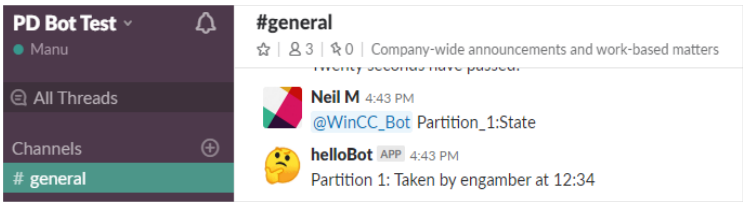


Figure 3: Slack bot posts

3 Implementation

3.1 Pipeline

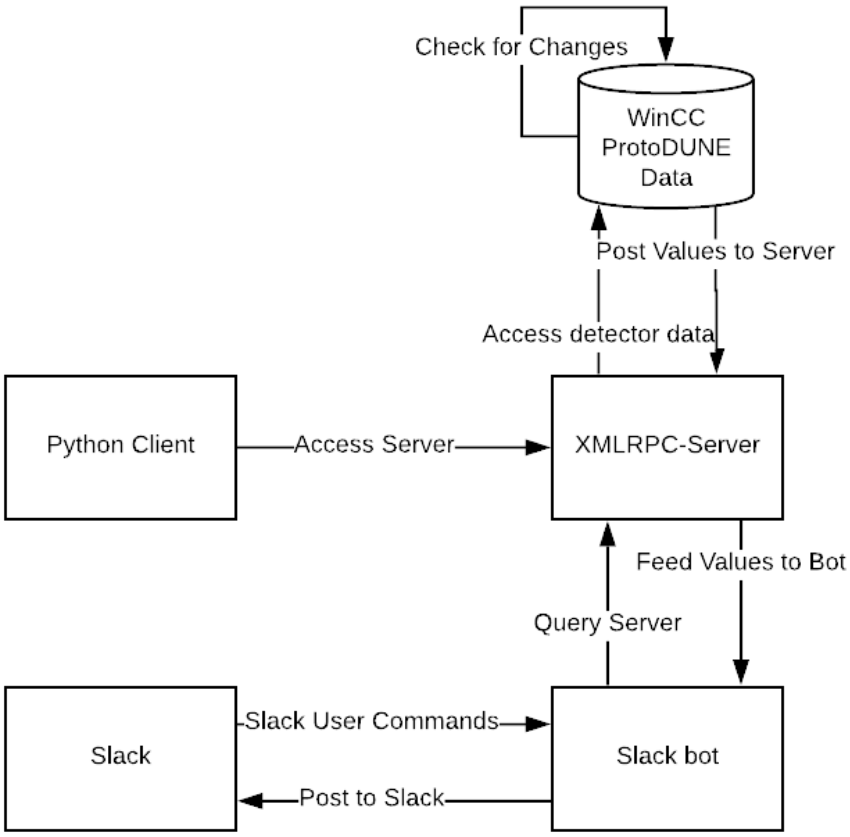


Figure 4: Slack bot system process flow

The above pipeline shows the slack bot will interact with the XMLRPC Server, with the latter being connected to WinCC to receive data, also taking note of any changes. The bot is connected to Slack using the *slack-client*library and the Slack Real-Time-Messaging API. The bot is invoked when a user uses the syntax <*botName*> <*command*>. The RTM API allows the bot to consistently check the Slack channel and see if there have been any commands input by users, with an adjustable 1-second delay. Should a command be input and validated by the bot, the corresponding function will be executed.

3.2 Xmlrpc Server

The server used for this implementation holds the data type of Asynchronous XMLRPC. Essentially, this implies that threading mix-ins may be applied to the server, allowing for multi-threading and multiprocessing libraries to be used freely. XMLRPC Servers are capable of registering instances of functions or even entire classes and their functions within themselves. The *register_instance* command allows for a class of methods to be stored and recognized by the server. This is also applicable to any class parameters such as the dictionary used by the server to store name-value pairs. The below figures show data being input to the server via WinCC and Python:

Figure 5: WinCC Additions to XMLRPC Server

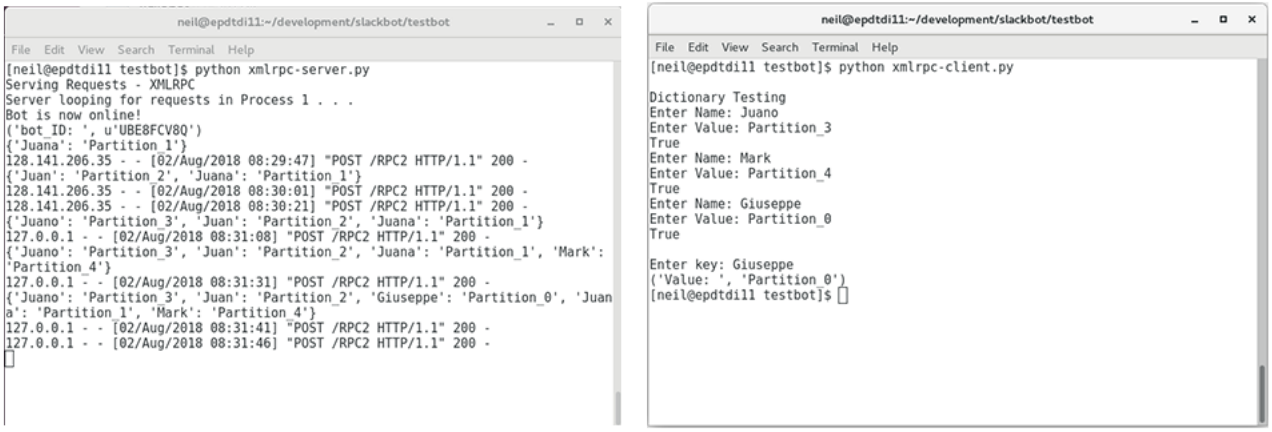


Figure 6: Server maintains WinCC data and appends client additions

The figure above shows how the server will maintain its dictionary following multiple client calls from different sources. Data will only be removed from the server if it is switched off, which may be avoided by making use of file I/O, or better yet online storage, to minimize read/write overhead. The section below will elaborate on the slack bot's connection to the server, and how this link may be used by the team members. As of writing this report (15/8/2018), this system has been shifted from a panel to a standalone script, which is running on the production machine and successfully communicating with the server.

3.3 Slackbot implementation

The RTM API allows the bot to monitor the chat for any user commands, with minimal timeout between reads. As an example the *#var <User>* command will ask the bot to check if the respective user currently owns a resource on the server. The bot will then access the XMLRPC Server dictionary to check for this user, and return the held resources (when applicable). Thus, the server acts as the link between the bot and the hardware readouts available on WinCC. This system has also been implemented onto the production machine, and runs consistently as a service script connecting to the previously mentioned XMLRPC Server. The commands *#owner <Partition ID>* and *#partition <Owner Name>* have also been introduced as a quality-of-life update. The figure below shows the bot publishing a value via WinCC, as well as a user's var command:

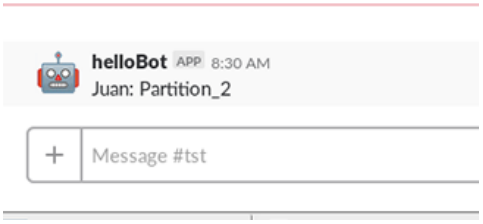


Figure 7: Slack bot publishing on channel from WinCC

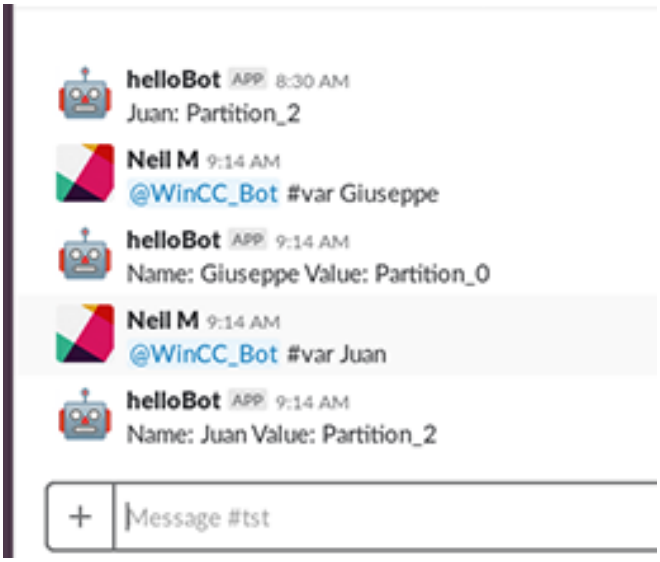


Figure 8: Slack bot using var command to access server

As described briefly in the pipeline section above, the *var* command invokes the bot to communicate with the server, which will transitively allow the users a way to see server-side information such as the resources held by a particular user. This is very helpful in reducing time wasted by users posting on Slack to inform others of holding a partition or resource. It is also beneficial when other users may wish to see what resources a particular user holds, as there is no waiting on that user involved. Another noteworthy feature of the bot is that can follow the FSM Tree of any of the Partitions (e.g. $Booted -> Configured -> Running$) and publish them onto the Slack channel. This saves time for users which may otherwise be spent checking on the run summary, or skimming through very long debug logs produced by the system.

4 Appendix: WinCC Panel Design for NP04

Another task in this studentship was to design a panel in WinCC to represent the NP04 detector. The design would include all 6 APAs, the top and bottom of the Time Projection Chamber, as well as the dynamic and static temperature gradients. Other useful information provided in the panel includes cryogenic system information such as the level of Liquid Argon and ambient conditions for the detector such as humidity, pressure, and temperature. The figure below shows the panel as it is before execution:

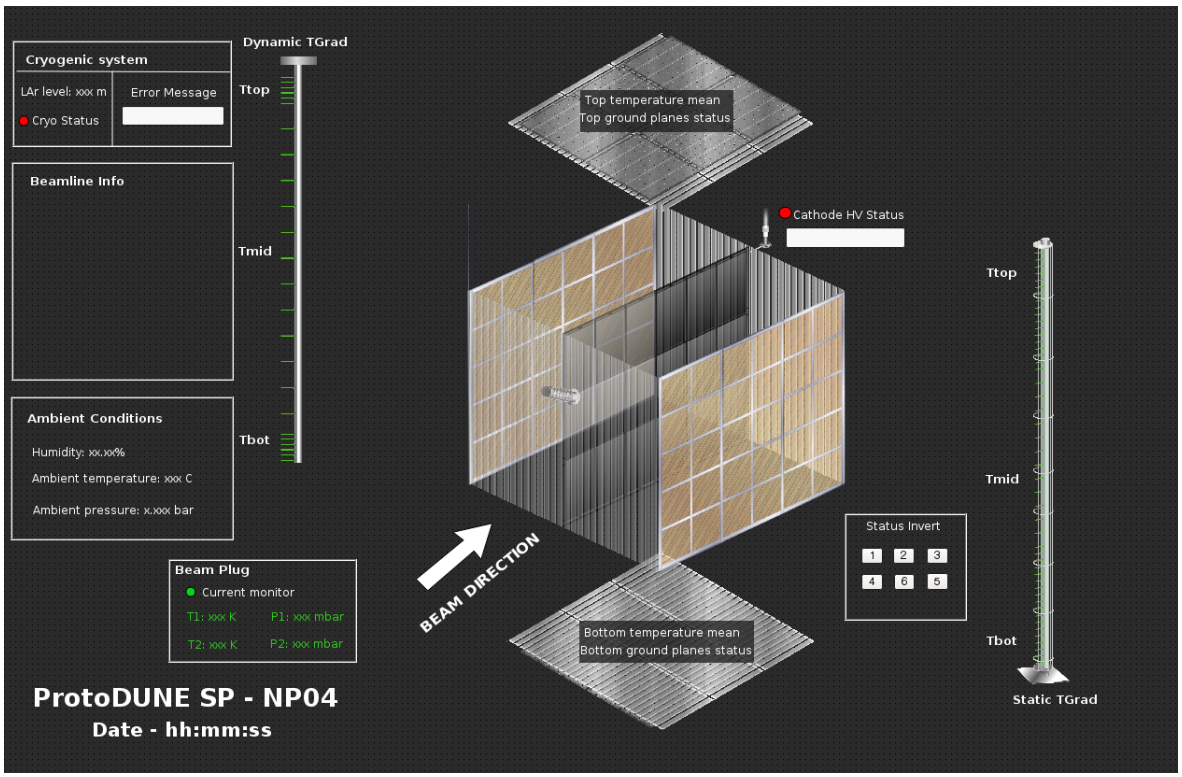


Figure 9: NP04 panel view

The cube represents the detector itself within the cryostat, with the grid-like dark yellow parts being the APAs (3 2-column APA on each side). The values are merely set as placeholders and may easily be replaced by real-time numbers from the detector interface online. The top and bottom TPCs are they grey grids, which also have their own values being monitored. The 'status invert' dialog is solely for testing purposes for a feature of the APAs which is available during run-time as may be observed in the following figure:

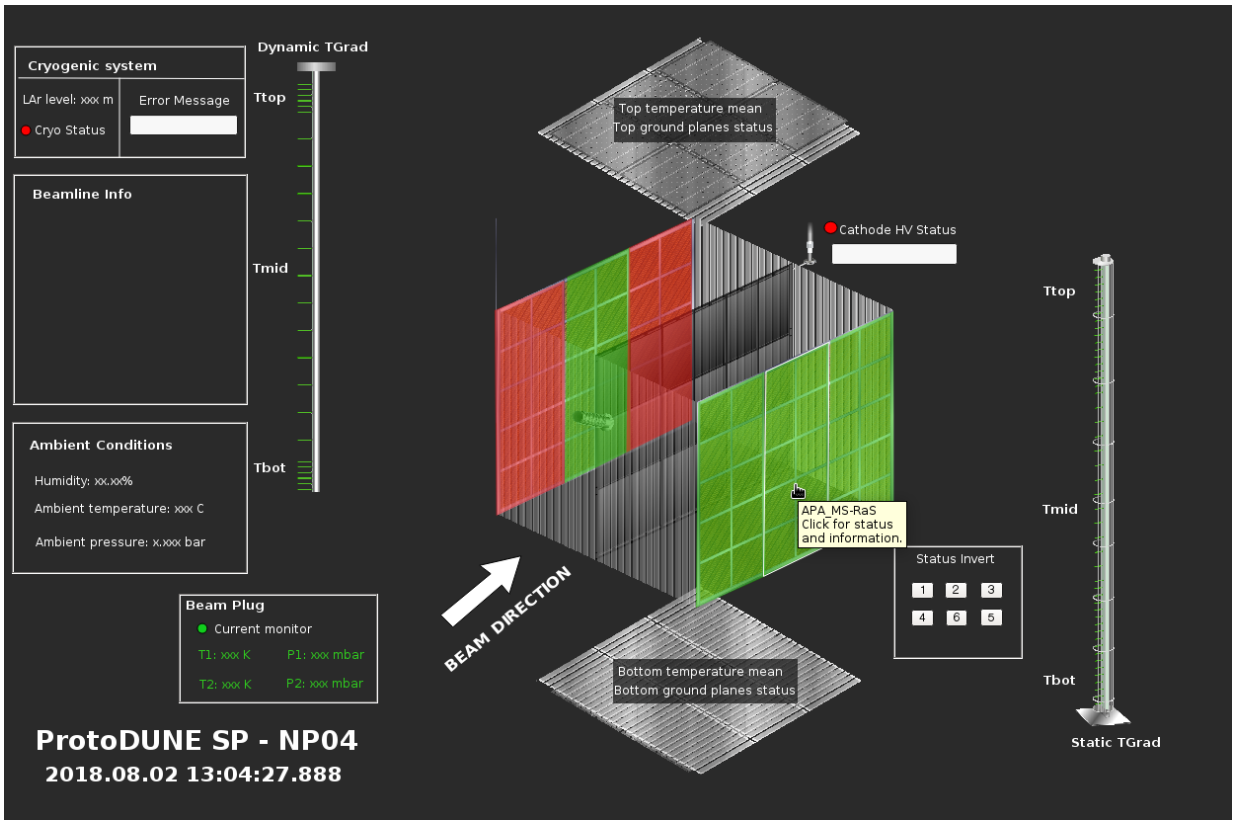


Figure 10: NP04 panel during runtime

The APAs have been allocated a strong presence on the panel due to the fact that they are one of the more critical parts of the TPC. They are attributed this level of importance since they are the charge sensing devices, as well as the photon detector carriers. Each two APAs carry modules of different design where the Argon scintillation light is trapped and collected in square volumes along each bar[1]. These 10 photon detector bars per two APAs are what gives the front and back of the TPC its grid-like shape, which can even be seen in the panel. lit up in different colours depending on the 'state' of each one. Dummy states are set for the APAs by making use of previously mentioned invert dialog. When applied with real-time data, it could instead signify irregular conditions such as an extreme temperature, pressure or voltage. Each APA can also be hovered over to display its name, and clicked to generate a separate panel specifically for that APA. The temperature gradients also have their own panel generator scripts when clicked, as it provides a more intuitive view of the data without further cluttering up the main panel in Figure 10. When clicking on the panel, as in the figure, the following APA status panel would be shown:

Figure 11: Separate APA view after click on previous image area

The APA and name in this window is dynamically generated depending on which APA was clicked. That APA status is then mirrored onto this one, as may be any statistics and conditions such as temperature and pressure from the live server.

References

[1] C. Touramanis and F. Cavanna, "2018 Status Report of NP04 (ProtoDUNE-SP)," Tech. Rep. CERN-SPSC-2018-011. SPSC-SR-230, CERN, Geneva, Apr 2018.